KIẾN TRÚC MÁY TÍNH VÀ HỢP NGỮ

ĐỒ ÁN MÔN HỌC #1

BIỂU DIỄN VÀ TÍNH TOÁN SỐ NGUYÊN LỚN

TH2015/1
Khoa Công nghệ Thông tin
Đại học Khoa học Tự nhiên TP HCM
Tháng 04/2017

1512561 – Hoàng Thị Hoài Thương

1512681 – Hứa Văn Vũ

1512682 – Lê Hoàng Vũ

Mục lục

I. GIỚI THIỆU

1.1 Nội dung đồ án

Xây dựng một chương trình hướng đối tượng để mô tả kiểu dữ liệu số nguyên lớn.

1.2 Yêu cầu đồ án

Kiểu dữ liệu số nguyên lớn có dấu gọi là QInt có độ lớn 16 byte gồm một số chức năng:

- Chuyển đổi số QInt từ hệ thập phân sang hệ nhị phân và ngược lại
- Chuyển đổi số QInt từ hệ nhị phân sang hệ thập lục phân và ngược lại
- Các operator=, operator+, operator-, operator*, operator/
- Các toán tử AND "&", OR "|", XOR "^", NOT "~"
- Các toán tử: dịch trái "<<", dịch phải ">>"
- Các phép xoay trái "rol", xoay phải "ror" mỗi lần xoay chỉ xử lý cho đúng 1 bit

1.3 Thông tin nhóm

1.3.1 Thông tin thành viên

MSSV	Họ và tên	Email	Vai trò
1512682	Lê Hoàng Vũ	elhoangvu@gmail.com	Leader, Developer
1512681	Hứa Văn Vũ	huavanvu2812@gmail.com	Developer
1512561	Hoàng Thị Hoài Thương	hoaithuong23101997@gmail.com	Developer

Bảng 1.1: Bảng thông tin thành viên nhóm

1.3.2 Phân công công việc

Họ tên	Công việc	Các hàm / phương thức cần viết	Mô tả
Lê Hoàng Vũ	Thiết kế lớp		Thiết kế thuộc tính, các hàm, phương thức của lớp
	Viết báo cáo		Viết file báo cáo
	Viết hàm đọc/ xuất file	int main()	Viết trong hàm main và đọc xuất file trong hàm main
	Viết hàm nhập dữ liệu từ chuỗi số	QInt(int type, string strOfNum)	Hàm dựng với tham số chuỗi số và loại của hệ số Nhập dữ liệu chuỗi số vào class ứng theo hệ

		void inputUnsignedDecimal(string decimal)	Nhập dữ liệu vào class theo chuỗi số nguyên dương ở hệ 10
		void inputHexa (string hexa)	Nhập dữ liệu vào class theo chuỗi số Hexa
		void inputBinary (string binary)	Nhập dữ liệu vào class theo chuỗi số Binary
	Viết hàm xuất chuỗi số	string toDecimal()	Xuất ra chuỗi số ở dạng nhị phân
		string toDecimal()	Xuất ra chuỗi số ở dạng thập phân
		string toHexa()	Xuất ra chuỗi số ở dạng thập lục phân
Hứa Văn Vũ	Viết toán tử cộng	QInt operator+(QInt x)	Toán tử cộng 2 QInt
	Viết toán tử trừ	QInt operator-(QInt x)	Toán tử trừ 2 QInt
	Viết toán tử nhân	QInt operator*(QInt x)	Toán tử nhân 2 QInt
	Viết toán tử chia	QInt operator/(QInt x)	Toán tử chia 2 QInt
Hoàng Thị Hoài Thương	Viết toán tử dịch trái	QInt operator<<(int n)	Toán tử dịch trái n bit
	Viết toán tử dịch phải	QInt operator>>(int n)	Toán tử dịch phải n bit
	Viết phương thức xoay trái	QInt rol()	Phương thức xoay trái 1 bit
	Viết phương thức xoay phải	QInt ror()	Phương thức xoay phải 1 bit
	Viết toán tử AND	QInt operator&(QInt x)	Toán tử AND 2 QInt
	Viết toán tử OR	QInt operator|(QInt x)	Toán tử OR 2 QInt
	Viết toán tử XOR	QInt operator^(QInt x)	Toán tử XOR 2 QInt
	Viết toán tử NOT	QInt operator~()	Toán tử NOT QInt

Bảng 1.2: Bảng phân công các công việc cho các thành viên nhóm

II. Ý TƯỞNG THỰC HIỆN

Xây dựng lớp QInt để chứa dữ liệu, thành phần thuộc tính là 1 mảng gồm 2 phần tử thuộc kiểu long long (__int64) tên là arrBit. Mỗi một phần tử mảng arrBit là 8 byte tương ứng 64 bit nên lớp QInt sẽ là lớp dữ liệu chứa được số 16 byte tương ứng 128 bit.

2.1 Cách lưu trữ:

Xem mảng arrBit 2 phần tử là 1 dãy 128 bit liên tục, đánh số thứ tự bit theo tự nhiên tức là từ phải qua trái tương ứng với bit thứ 0 đến bit thứ 127

Mảng chứa dữ liệu	arrBit[0]					arrBit[1]				
Thứ tự bit trong mỗi phần tử mảng	63	62	…	1	0	63	62	…	1	0
Thứ tự bit được xem trong 1 số QInt 128 bit	127	126	125	…	…	…	3	2	1	0

Bảng 1: Bảng mô tả các bit trong thiết kế lưu trữ của lớp QInt

2.2 Phạm vi lưu trữ:

- Số nhị phân: có độ dài tối đa 128 bit
- Số thập phân: từ -2^{127} đến $2^{128} - 1$
- Số thập lục phân: số hexa có độ dài tối đa là 32 ký tự hexa

2.3 Cách nhập liệu:

Dữ liệu được đưa vào dưới dạng chuỗi (*string strOfNum*) kèm theo kiểu hệ số cần nhập (*int type*).

Xử lý chuỗi chuyển về hệ nhị phân và thực hiện lưu từng bit vào 127 bit của mảng arrBit, bao gồm:

- Hệ thập phân: Thực hiện chia 2 và lấy dư làm bit và đặt vào lớp QInt, đối với số âm chuyển sang số bù 2
- Hệ thập lục phân: Thực hiện chuyển từng ký tự hexa sang nhị phân 4 bit và đặt vào lớp QInt
- Hệ nhị phân: Thực hiện đặt từng ký tự nhị phân vào bit tương ứng của lớp QInt

2.4 Cách xuất:

Khi dữ liệu của một lớp QInt là 1 dãy 127 bit thì việc xuất ra hệ 2, 10 hay 16 làm tương tự như cách tự nhiên

- Hệ nhị phân: Xuất chuỗi nhị phân ứng với giá trị của từng bit
- Hệ thập phân: Xuất chuỗi thập phân theo dãy 127 bit
- Hệ thập lục phân: Xuất chuỗi hexa bằng cách gom 4 bit thành 1 ký tự hexa

2.5 Thuật toán sử dụng:

- Toán tử nhân (operator *): Thuật toán nhân cải tiến – xử lý số âm

- Toán tử chia (operator /): Thuật toán chia – chỉ xử lý được số dương nên thêm vào thao tác bỏ dấu và xét dấu
- Toán tử cộng (operator +): Thuật toán cộng Half Adder

III. CÀI ĐẶT ĐỒ ÁN

3.1 Môi trường lập trình

Visual Studio 2015

3.2 Tên lớp đối tượng

QInt

3.3 Thuộc tính dữ liệu trong lớp QInt

long long arrBit[128];

3.4 Nhập dữ liệu vào

Khởi tạo mặc định dữ liệu có sẵn trong QInt là 0 (arrBit[0] = 0 và arrBit[1] = 0)

3.4.1 Nhập số nhị phân

Kiểu truyền vào là một dãy nhị phân ở dạng chuỗi, tối đa 128 ký tự nhị phân: *string strOfNum*

Thực hiện: duyện chuỗi nhị phân từ phải sang trái và lấy từng ký tự nhị phân thứ i chuyển sang số nguyên, sau đó gọi hàm *setBit1(i)*[1]để đặt bit 1 ngay vị trí thứ i trong QInt

Hình 3.1: Nhập chuỗi nhị phân và xuất biểu diễn 128 bit đã được lưu

3.4.2 Nhập số thập phân

Kiểu truyền vào là một dãy thập phân ở dạng chuỗi: *string strOfNum*

[1] setBit1(i) là hàm gán bit 1 vào vị trí thứ i trong QInt

Thực hiện: Kiểm tra chuỗi số thập phân là âm hay dương

- Nếu là dương: (không có dấu trừ ‘-‘ ở đầu)

  Chạy vòng lặp cho i từ 0 đến 127

  {

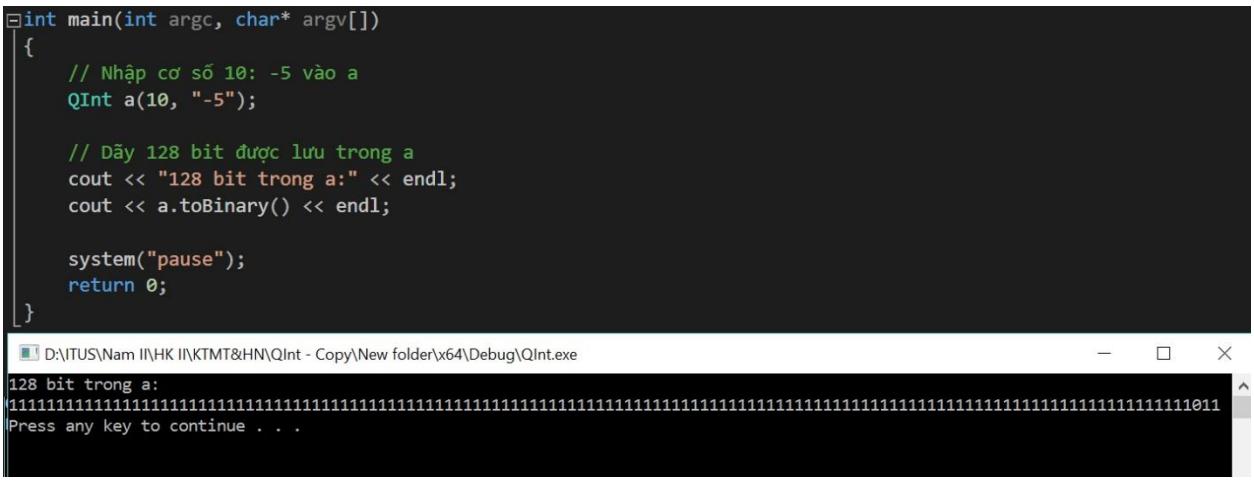
  Gọi hàm divTwo(strOfNum) để chia 2 chuỗi và lấy dư. Số dư là 1 hoặc 0, nếu là 0 thì bỏ qua, nếu là 1 thì setBit1(i)

  }

Hình 3.2: Nhập chuỗi thập phân dương và xuất biểu diễn 128 bit đã được lưu

- Nếu là âm: (có dấu trừ ‘-‘ ở đầu)

  Xóa dấu trừ, nhập như số dương, đảo dấu QInt và cộng thêm bit 1



Hình 3.3: Nhập chuỗi thập phân âm và xuất biểu diễn 128 bit đã được lưu

3.4.3 Nhập số thập lúc phân

Kiểu truyền vào là một dãy thập lục phân ở dạng chuỗi, tối đa 32 ký tự hexa: *string strOfNum*

Thực hiện: Duyệt chuỗi hexa từ phải sang trái, chuyển từng ký tự hexa sang số nhị phân 4 bit bằng cách đổi ký tự hexa sang số thập phân và lấy bit. Gán từng bit vào QInt bằng hàm *setBit(i)*

```cpp
int main(int argc, char* argv[])
{
    // Nhập cơ số 16: 15ABFF vào a
    QInt a(16, "15ABFF");

    // Dãy 128 bit được lưu trong a
    cout << "128 bit trong a:" << endl;
    cout << a.toBinary() << endl;

    system("pause");
    return 0;
}
```

```
D:\ITUS\Nam II\HK II\KTMT&HN\QInt - Copy\New folder\x64\Debug\QInt.exe
128 bit trong a:
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000101011010101111111111
Press any key to continue . . .
```

Hình 3.4: Nhập chuỗi thập lục phân và xuất biểu diễn 128 bit đã được lưu

3.5 Xuất dữ liệu ra

Dữ liệu ra là 1 chuỗi số thuộc kiểu string, ngoài ra còn 1 số thủ thuật để loại bỏ số không ở đầu mà không được đề cặp.

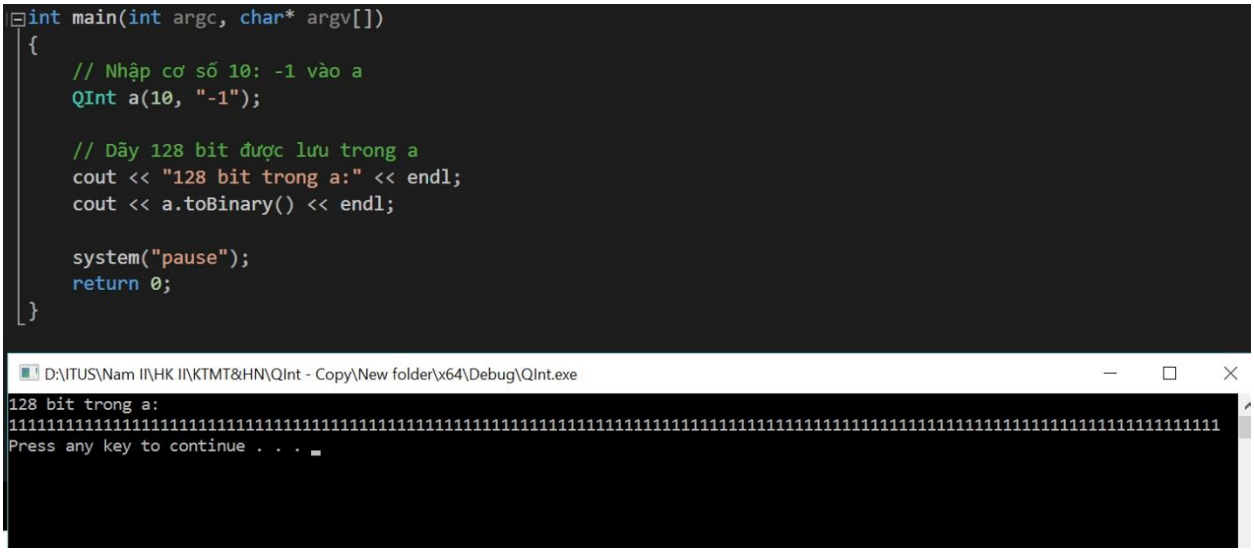
3.5.1 Xuất số nhị phân

Tạo một biến thuộc kiểu chuỗi để chứa kết quả: *string result*

Chạy vòng lặp từ 127 đến 0. Gọi hàm *getBit(i)*[2] để lấy bit ở vị trí thứ i và chuyển bit đó thành ký tự để cộng dồn vào chuỗi kết quả *result*

Kết quả: chuỗi *result*

[2] getBit(i) là hàm lấy bit thứ i trong QInt, kết quả trả về là 0 hoặc 1



Hình 3.5: Xuất biểu diễn nhị phân của số QInt a

3.5.2 Xuất số thập phân

Tạo một biến thuộc kiểu chuỗi để chứa kết quả: *string result*

Chạy vòng lặp i từ 126 đến 0, gọi hàm *getBit(i)*. Nếu *getBit(i)* bằng 0 thì bỏ qua, nếu bằng 1 thì tính 2^i bằng hàm *powOfTow(i)*[3] và cộng vào chuỗi result theo cách cộng số nguyên bằng hàm *sumUnsignedDecimal(result, powOfTow(i))*[4]

Bước cuối: nếu *getBit(127)* là 1 thì số đang ở dạng bù 2, kết quả là 2^127 – *result* → *result* (cần 1 hàm *subUnsignedDecimal(string, string)*[5] để trừ 2 chuỗi theo cách trừ số nguyên). Ngược lại thì result là kết quả

Kết quả: chuỗi *result*

[3] powOfTwo(i) là hàm tính 2^i ở hệ thập phân và trả về chuỗi kết quả kiểu string

[4] sumUnsignedDecimal(string a, string b) là hàm cộng chuỗi số nguyên ở hệ thập phân của chuỗi a cho chuỗi b, kết quả trả về là 1 chuỗi số kiểu string

[5] subUnsignedDecimal(string a, string b) là hàm trừ chuỗi số nguyên ở hệ thập phân của chuỗi a cho chuỗi b, kết quả trả về là 1 chuỗi số kiểu string

```
int main(int argc, char* argv[])
{
    // Nhập cơ số 2: 127 ký tự '1' vào a
    QInt a(2, "11111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111");

    // Dãy 128 bit được lưu trong a
    cout << "128 bit trong a:" << endl;
    cout << a.toBinary() << endl;

    // Xuất cơ số 10
    cout << "Decimal" << endl;
    cout << a.toDecimal() << endl;

    system("pause");
    return 0;
}
```

```
D:\ITUS\Nam II\HK II\KTMT&HN\QInt - Copy\New folder\x64\Debug\QInt.exe
128 bit trong a:
01111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111
Decimal
170141183460469231731687303715884105727
Press any key to continue . . .
```

Hình 3.6: Xuất biểu diễn thập phân của số QInt a

3.5.3 Xuất số thập lục phân

Tạo một biến thuộc kiểu chuỗi để chứa kết quả: *string result*

Chạy vòng lặp I từ 127 đến 0, dùng hàm *getBit(i)* để lấy bit thứ i sau đó gom cứ mỗi 4 bit chuyển sang hệ thập phân. Nếu kết quả chuyển nhỏ hơn 10 thì đó là ký tự hexa, cộng thêm 48 và chuyển sang ký tự, thêm vào *result.* Ngược lại chuyển sang ký tự hexa bằng cách cộng cho 55 và chuyển sang ký tự (tận dụng bảng mã ASCII: 10 + 55 = 65 (A), 11 + 55 = 66 (B),…)

Kết quả: chuỗi *result*

```
int main(int argc, char* argv[])
{
    // Nhập cơ số 10: -15 vào a
    QInt a(10, "-15");

    // Dãy 128 bit được lưu trong a
    cout << "128 bit trong a:" << endl;
    cout << a.toBinary() << endl;

    // Xuất cơ số 16
    cout << "Hexa: " << endl;
    cout << a.toHexa() << endl;

    system("pause");
    return 0;
}
```

```
D:\ITUS\Nam II\HK II\KTMT&HN\QInt - Copy\New folder\x64\Debug\QInt.exe
128 bit trong a:
11111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111110001
Hexa:
FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF1
Press any key to continue . . .
```

Hình 3.7: Xuất biểu diễn thập lục phân của số QInt a

3.6 Toán tử luận lý

3.6.1 Toán tử & (AND)

Khi thực hiện toán tử AND giữa 2 kiểu QInt ta thực hiện AND giữa các phần tử mảng tương ứng của chúng

Ví dụ: a & b với a, b thuộc kiểu QInt

	arrBit[0]					arrBit[1]				
QInt a	1	0	…	1	1	1	1	…	1	1
Toán tử	**&**									
QInt b	1	0	…	0	0	1	0	…	1	0
Kết quả	1	0	…	0	0	1	0	…	1	0

Bảng 3.1: Bảng mô tả phép AND

```
// Xuất 128 bit của a
cout << "128 bit cua a: " << endl;
cout << a.toBinary() << endl;

// Xuất 128 bit của b
cout << "128 bit cua b: " << endl;
cout << b.toBinary() << endl;

// Xuất 128 bit của a & b
cout << "a & b:" << endl;
cout << (a & b).toBinary() << endl;
```

```
D:\ITUS\Nam II\HK II\KTMT&HN\QInt - Copy\New folder\x64\Debug\QInt.exe
128 bit cua a:
11111111111111111111111111111111111111110000000010110010100000101001001010000111101001000001011100111100111111111000000000000000000000
128 bit cua b:
00000000000000000000000000000000100111111001000001101100111000010011101101110100000010101011111111111111000010000000000000000000000
a & b:
00000000000000000000000000000000100111100000000001100100000000000000010100001010000000000000111001111001100001000000000000000000000
Press any key to continue . . .
```

Hình 3.8: Xuất biểu diễn 128 bit của phép a AND b

3.6.2 Toán tử | (OR)

Khi thực hiện toán tử OR giữa 2 kiểu QInt ta thực hiện OR giữa các phần tử mảng tương ứng của chúng

Ví dụ: a | b với a, b thuộc kiểu QInt

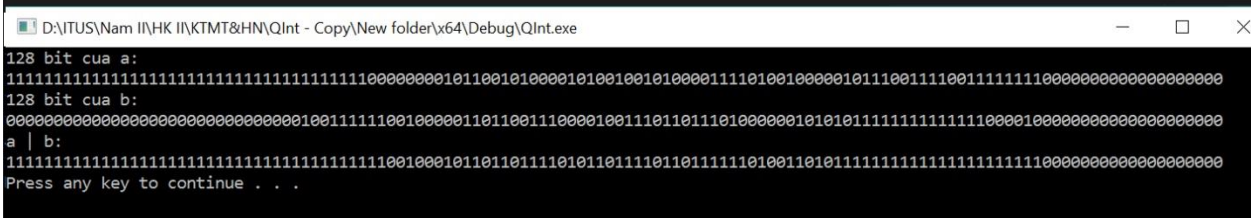
	arrBit[0]					arrBit[1]				
QInt a	1	0	…	1	1	1	1	…	1	1
Toán tử	|									
QInt b	1	0	…	0	0	1	0	…	1	0
Kết quả	1	0	…	1	1	1	1	…	1	1

Bảng 3.2: Bảng mô tả phép OR

```
// Xuất 128 bit của a
cout << "128 bit cua a: " << endl;
cout << a.toBinary() << endl;

// Xuất 128 bit của b
cout << "128 bit cua b: " << endl;
cout << b.toBinary() << endl;

// Xuất 128 bit của a | b
cout << "a | b:" << endl;
cout << (a | b).toBinary() << endl;
```



Hình 3.9: Xuất biểu diễn 128 bit của phép a OR b

3.6.3 Toán tử ^ (XOR)

Khi thực hiện toán tử XOR giữa 2 kiểu QInt ta thực hiện XOR giữa các phần tử mảng tương ứng của chúng

Ví dụ: a ^ b với a, b thuộc kiểu QInt

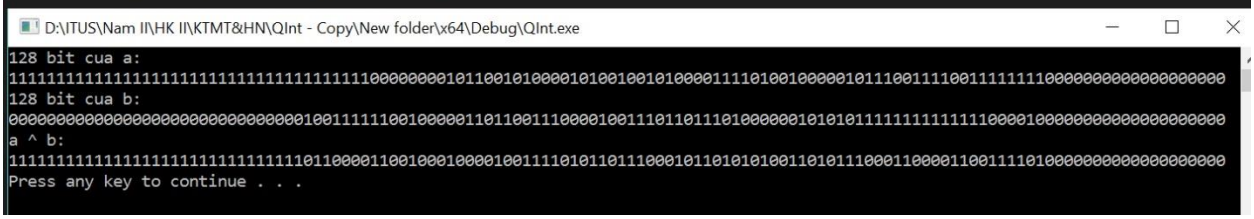
	arrBit[0]					arrBit[1]				
QInt a	1	0	…	1	1	1	1	…	1	1
Toán tử	^									
QInt b	1	0	…	0	0	1	0	…	1	0
Kết quả	0	0	…	1	1	0	1	…	0	1

Bảng 3.3: Bảng mô tả phép XOR

```
// Xuất 128 bit của a
cout << "128 bit cua a: " << endl;
cout << a.toBinary() << endl;

// Xuất 128 bit của b
cout << "128 bit cua b: " << endl;
cout << b.toBinary() << endl;

// Xuất 128 bit của a ^ b
cout << "a ^ b:" << endl;
cout << (a ^ b).toBinary() << endl;
```



Hình 3.10: Xuất biểu diễn 128 bit của phép a XOR b

3.6.4 Toán tử ~ (NOT)

Khi thực hiện toán tử NOT 1 QInt ta thực hiện NOT tường phần tử mảng của chúng

Ví dụ: ~a với a thuộc kiểu QInt

	arrBit[0]					arrBit[1]				
Toán tử	~									
QInt a	1	0	…	0	0	1	0	…	1	0
Kết quả	0	1	…	1	1	0	1	…	0	1

Bảng 3.4: Bảng mô tả phép NOT

```
// Xuất 128 bit của a
cout << "128 bit cua a: " << endl;
cout << a.toBinary() << endl;

// Xuất 128 bit của ~a
cout << "~a:" << endl;
cout << (~a).toBinary() << endl;
```

```
D:\ITUS\Nam II\HK II\KTMT&HN\QInt - Copy\New folder\x64\Debug\QInt.exe
128 bit cua a:
11111111111111111111111111111111111111110000000010110010100000101001001010000111101001000001011100111100111111111000000000000000000
~a:
00000000000000000000000000000000000000001111111101001101011110101101101011110000101101111101000110000110000000011111111111111111
Press any key to continue . . .
```

Hình 3.11: Xuất biểu diễn 128 bit của phép NOT a

3.7 Toán tử dịch

3.7.1 Toán tử << (dịch trái)

Xét 2 trường hợp: n < 64 và n >= 64

- **Trường hợp 1: n < 64**

  Dịch trái arrBit[0] n bit, lấy (64 - n) bit cao nhất của arrBit[1] lấp vào (64 - n) bit thấp nhất của arrBit[0] bằng cách OR với arrBit[1] sau khi dịch phải luận lý 64 - n bit

  arrBit[0] << n | (unsigned long long) arrBit[1] >> (64 - n);

  Dịch trái arrBit[1] n bit

  arrBit[1] << n;

Ví dụ: a << 1 với a thuộc kiểu QInt

	arrBit[0]					arrBit[1]				
Thứ tự bit	63	62	…	1	0	63	62	…	1	0
QInt a	0	1	…	1	0	1	0	…	0	1
Toán tử	<< 1									
Kết quả	1	…	1	0	1	0	…	0	1	0

Bảng 3.5: Bảng mô tả phép dịch trái 1 bit

```
int main(int argc, char* argv[])
{
    // Nhập cơ số 10: -123456789000000000000000000 vào a
    QInt a(10, "-123456789000000000000000000");

    // Xuất 128 bit của a
    cout << "128 bit cua a: " << endl;
    cout << a.toBinary() << endl;

    // Xuất 128 bit của a khi dịch trái 20
    cout << "Dich trai 20:" << endl;
    cout << (a << 20).toBinary() << endl;

    system("pause");
    return 0;
}
```

```
D:\ITUS\Nam II\HK II\KTMT&HN\QInt - Copy\New folder\x64\Debug\QInt.exe
128 bit cua a:
11111111111111111111111111111111111111100000000101100101000010100100101000011110100100000101110011110011111111000000000000000000000
Dich trai 20:
11111111111111111110000000010110010100000101001001010000111101001000001011100111100111111111000000000000000000000000000000000000000
Press any key to continue . . .
```

Hình 3.12: Xuất biểu diễn 128 bit của phép dịch trái 20 bit

- **Trường hợp 2: n >= 64**

  arrBit[0] sẽ là arrBit[1] sau khi dịch trái (n – 64) bit

  arrBit[0] << (n - 64);

  arrBit[1] sẽ toàn bit 0

  Ví dụ: a << 64 với a thuộc kiểu QInt

	arrBit[0]					arrBit[1]				
Thứ tự bit	63	62	…	1	0	63	62	…	1	0
QInt a	0	1	…	1	0	1	0	…	0	1
Toán tử	<< 64									
Kết quả	1	0	…	0	1	0	…	0	0	0

Bảng 3.6: Bảng mô tả phép dịch trái 64 bit

Hình 3.13: Xuất biểu diễn 128 bit của phép dịch trái 80 bit

3.7.2 Toán tử >> (dịch phải)

Xét 2 trường hợp

- **Trường hợp 1: n < 64**

  Dịch phải luận lý arrBit[1] n bit, lấy (64 - n) bit thấp nhất nhất của arrBit[0] lấp vào (64 - n) bit cao nhất của arrBit[1] bằng cách OR với arrBit[0] sau khi dịch trái (64 – n) bit

  (unsigned long long) arrBit[1] >> n | arrBit[0] << (64 - n);

  Dịch phải arrBit[0] n bit

  arrBit[0] >> n;

  Ví dụ: a >> 1 với a thuốc kiểu QInt

	arrBit[0]					arrBit[1]				
Thứ tự bit	63	62	…	1	0	63	62	…	1	0
QInt a	0	1	…	1	0	1	0	…	0	1
Toán tử	>> 1									
Kết quả	0	0	1	…	1	0	1	0	…	0

Bảng 3.7: Bảng mô tả phép dịch phải 1 bit

Hình 3.14: Xuất biểu diễn 128 bit của phép dịch phải 20 bit

- **Trường hợp 2: n >= 64**

  arrBit[1] sẽ là arrBit[0] sau khi dịch phải (n – 64) bit

  arrBit[1] >> (n - 64);

  arrBit[0] sẽ toàn bit 0 nếu nó lớn hơn hoặc bằng 0 (dịch luận lý) và toàn bit 1 nếu nhỏ hơn 0 (dịch số học)

  Ví dụ: a >> 64 với a thuộc kiểu QInt

	arrBit[0]					arrBit[1]				
Thứ tự bit	63	62	…	1	0	63	62	…	1	0
QInt a	1	0	…	1	0	1	0	…	0	1
Toán tử	>> 64									
Kết quả	1	1	1	…	1	1	0	…	1	0

Bảng 3.8: Bảng mô tả phép dịch phải 64 bit

```
int main(int argc, char* argv[])
{
    // Nhập cơ số 10: -123456789000000000000000000 vào a
    QInt a(10, "-123456789000000000000000000");

    // Xuất 128 bit của a
    cout << "128 bit cua a: " << endl;
    cout << a.toBinary() << endl;

    // Xuất 128 bit của a khi dịch phải 60
    cout << "Dich phai 80:" << endl;
    cout << (a >> 80).toBinary() << endl;

    system("pause");
    return 0;
}
```

```
D:\ITUS\Nam II\HK II\KTMT&HN\QInt - Copy\New folder\x64\Debug\QInt.exe
128 bit cua a:
11111111111111111111111111111111111110000000010110010100001010010010100001111010010000010111001111001111111100000000000000000000
Dich phai 80:
11111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111110000000010
Press any key to continue . . .
```

Hình 3.15: Xuất biểu diễn 128 bit của phép dịch phải 80 bit

3.8 Phương thức xoay

3.8.1 Phương thức ror (xoay phải)

Dịch phải luận lý arrBit[0] 1 bit và lấy 1 bit thấp nhất (bit thứ 0) của arrBit[1] lấp vào 1 bit cao nhất (bit thứ 63) của arrBit[0] bằng cách OR với arrBit[1] sau khi dịch trái 63 bit

(unsigned long long) arrBit[0] >> 1 | arrBit[1] << 63;

Dịch phải luận lý arrBit[1] 1 bit và lấy 1 bit thấp nhất (bit thứ 0) của arrBit[0] lấp vào 1 bit cao nhất (bit thứ 63) của arrBit[1] bằng cách OR với arrBit[0] sau khi dịch trái 63 bit

(unsigned long long) arrBit[1] >> 1 | arrBit[0] << 63;

```
int main(int argc, char* argv[])
{
    // Nhập cơ số 10: -123456789 vào a
    QInt a(10, "-123456789");

    // Xuất 128 bit của a
    cout << "128 bit cua a: " << endl;
    cout << a.toBinary() << endl;

    // Xuất 128 bit của a khi xoay phải 1
    cout << "Xoay phai: " << endl;
    cout << a.ror().toBinary() << endl;

    system("pause");
    return 0;
}
```

```
D:\ITUS\Nam II\HK II\KTMT&HN\QInt - Copy\New folder\x64\Debug\QInt.exe
128 bit cua a:
11111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111000101001000011001011101011
Xoay phai:
11111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111000101001000011001011101011
Press any key to continue . . .
```

Hình 3.16: Xuất biểu diễn 128 bit của phép xoay phải 1 bit

3.8.2 Phương thức rol (xoay trái)

Dịch trái arrBit[0] 1 bit và lấy 1 bit cao nhất (bit thứ 63) của arrBit[1] lấp vào 1 bit thấp nhất (bit thứ 0) của arrBit[0] bằng cách OR với arrBit[1] sau khi dịch phải luận lý 63 bit

(unsigned long long) arrBit[0] << 1 | arrBit[1] >> 63;

Dịch trái arrBit[1] 1 bit và lấy 1 bit cao nhất (bit thứ 63) của arrBit[0] lấp vào 1 bit thấp nhất (bit thứ 0) của arrBit[1] bằng cách OR với arrBit[0] sau khi dịch phải luận lý 63 bit

(unsigned long long) arrBit[1] << 1 | arrBit[0] >> 63;

```
int main(int argc, char* argv[])
{
    // Nhập cơ số 10: -123456789 vào a
    QInt a(10, "-123456789");

    // Xuất 128 bit của a
    cout << "128 bit cua a: " << endl;
    cout << a.toBinary() << endl;

    // Xuất 128 bit của a khi xoay trái 1
    cout << "Xoay trai: " << endl;
    cout << a.rol().toBinary() << endl;

    system("pause");
    return 0;
}
```

```
D:\ITUS\Nam II\HK II\KTMT&HN\QInt - Copy\New folder\x64\Debug\QInt.exe
128 bit cua a:
11111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111000101001000011001011101011
Xoay trai:
11111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111000101001000011001011101011
Press any key to continue . . .
```

Hình 3.17: Xuất biểu diễn 128 bit của phép xoay trái 1 bit

3.9 Toán tử tính toán

3.9.1 Toán tử + (cộng)

Sử dụng thuật toán cộng: Half Adder

Với a + b:

Lặp (a & b khác 0)

{

a = a ^ b;

b = (a & b) << 1;

}

Kết quả a | b

```
int main(int argc, char* argv[])
{
    // Nhập cơ số 10: 12345678912345678912345678 vào a
    QInt a(10, "12345678912345678912345678");

    // Nhập cơ số 10: -123 vào b
    QInt b(10, "-123");

    cout << a.toDecimal() << " + "
        << b.toDecimal() << " = ";
    // Xuất số tổng dưới dạng cơ số 10
    cout << (a + b).toDecimal() << endl;

    system("pause");
    return 0;
}
```

```
D:\ITUS\Nam II\HK II\KTMT&HN\QInt - Copy\New folder\x64\Debug\QInt.exe
12345678912345678912345678 + -123 = 12345678912345678912345555
Press any key to continue . . .
```

Hình 3.18: Xuất kết quả của phép cộng a + b ở hệ thập phân

3.9.2 Toán tử - (trừ)

Với a – b

Đảo dấu b: đảo bit b và cộng cho bit 1

Kết quả là a + b

```
int main(int argc, char* argv[])
{
    // Nhập cơ số 10: 123456789123456789123456789 vào a
    QInt a(10, "123456789123456789123456789");

    // Nhập cơ số 10: -123 vào b
    QInt b(10, "-123");

    cout << a.toDecimal() << " - "
        << b.toDecimal() << " = ";
    // Xuất số hiệu dưới dạng cơ số 10
    cout << (a - b).toDecimal() << endl;

    system("pause");
    return 0;
}
```

```
D:\ITUS\Nam II\HK II\KTMT&HN\QInt - Copy\New folder\x64\Debug\QInt.exe
123456789123456789123456789 - -123 = 123456789123456789123456912
Press any key to continue . . .
```

Hình 3.19: Xuất kết quả của phép trừ a - b ở hệ thập phân

3.9.3 Toán tử * (nhân)

Sử dụng thuật toán nhân cải tiến: Booth's Multiplication Algorithm

Với a * b

Khởi tạo: A = 0; k = n; Q_{-1} = 0 (thêm 1 bit = 0 vào cuối Q)

Lặp k lần

{

Nếu 2 bit cuối của Q_0Q_{-1}

{

= 10 thì A – M ⟶ A

= 01 thì A + M ⟶ A

= 00, 11 thì A không thay đổi

}

Dịch phải số học [A, Q, Q_{-1}]

}

Kết quả: [A, Q]

Do QInt chỉ có 128 bit nên chỉ lấy Q làm kết quả

```
int main(int argc, char* argv[])
{
    // Nhập cơ số 10: 12345678912345678912345678 vào a
    QInt a(10, "12345678912345678912345678");

    // Nhập cơ số 10: -123 vào b
    QInt b(10, "-123");

    cout << a.toDecimal() << " * "
        << b.toDecimal() << " = ";
    // Xuất số tích dưới dạng cơ số 10
    cout << (a * b).toDecimal() << endl;

    system("pause");
    return 0;
}
```

```
D:\ITUS\Nam II\HK II\KTMT&HN\QInt - Copy\New folder\x64\Debug\QInt.exe
12345678912345678912345678 * -123 = -1518518506218518506218518504 7
Press any key to continue . . .
```

*Hình 3.20: Xuất kết quả của phép nhân a * b ở hệ thập phân*

3.9.4 Toán tử / (chia)

Sử dụng thuật toán chia: Restoring Division Algorithm

Với a / b (b khác 0)

Do thuật toán chỉ chia đúng khi số dương chia số dương nên chuyển a, b sang dương, tạo 1 biến dấu để xét dấu cho thương sau khi chia xong

Khởi tạo: A = n bit 0 nếu Q > 0; A = n bit 1 nếu Q < 0; k = n

Lặp k lần

{

Shift left (SHL) [A, Q]

$A - M \rightarrow A$

 # Nếu $A < 0$*:* $Q_0 = 0$ *và* $A + M \rightarrow A$

 # Ngược lại: $Q_0 = 1$

}

Kết quả: Q là thương, A là số dư

Ta lấy Q làm kết quả

Cuối cùng dựa vào biến xét dấu mà ta xét dấu cho Q, nếu kết quả ra âm thì ta đảo dấu Q ngược lại thì trả về Q

```cpp
int main(int argc, char* argv[])
{
    // Nhập cơ số 10: 12345678912345678912345678 vào a
    QInt a(10, "12345678912345678912345678");

    // Nhập cơ số 10: -123 vào b
    QInt b(10, "-123");

    cout << a.toDecimal() << " / "
        << b.toDecimal() << " = ";
    // Xuất số thương dưới dạng cơ số 10
    cout << (a / b).toDecimal() << endl;

    system("pause");
    return 0;
}
```

```
D:\ITUS\Nam II\HK II\KTMT&HN\QInt - Copy\New folder\x64\Debug\QInt.exe
12345678912345678912345678 / -123 = -100371373271103080588176 2
Press any key to continue . . .
```

Hình 3.21: Xuất kết quả của phép chia a / b ở hệ thập phân

3.9.5 Toán tử = (gán bằng)

Gán 2 QInt cho nhau cũng như gán 2 phần tử mảng của QInt này cho QInt kia

Với a = b

a.arrBit[0] = b.arrBit[0];

b.arrBit[1] = b.arrBit[1];

```cpp
int main(int argc, char* argv[])
{
    // Nhập cơ số 10: 123456789 vào a
    QInt a(10, "123456789");

    // Gán a cho b
    QInt b = a;

    // Xuất số a dưới dạng cơ số 10
    cout << "a: ";
    cout << a.toDecimal() << endl;

    // Xuất số b dưới dạng cơ số 10
    cout << "b: ";
    cout << b.toDecimal() << endl;

    system("pause");
    return 0;
}
```

```
D:\ITUS\Nam II\HK II\KTMT&HN\QInt - Copy\New folder\x64\Debug\QInt.exe
a: 123456789
b: 123456789
Press any key to continue . . .
```

Hình 3.18: Xuất kết quả của phép gán a = b ở hệ thập phân

3.10 Xử lý chương trình kiểm tra

Sử dụng lớp *fstream* để đọc và xuất file, mỗi lần đọc 1 dòng trong file sau đó tách ra thành tối đa 4 chuỗi để xử lý.

Nếu tách được 3 chuỗi thì các thao tác có thể thực hiện là: đổi cơ số, toán tử ~, phép xoay trái, xoay phải

Nếu tách được 4 chuỗi thì các thao tác có thể thực hiện là: phép dịch trái, phép dịch phải, toán tử +, -, *, /, &, |, ^

IV. CHẠY THỬ - KIỂM TRA

4.1 Chạy thử - kiểm tra

Trong quá trình viết các toán tử, phương thức và các hàm thì đã được kiểm trong quá trình viết một cách tương đối

4.2 Tạo tệp tin INPUT để kiểm tra – sửa lỗi

4.2.1 Tạo tệp INPUT

Nhóm đưa ra một phương án, viết một chương trình tạo tập tin INPUT.TXT theo định dạng giống mô tả trong đồ án và phát sinh n dòng test (số rất lớn dòng test). Mục đích tìm lỗi và sửa lỗi đọc ghi file, nếu có thể kết hợp các nhóm khác sử dụng tệp INPUT.TXT được tạo ra để tổng hợp tệp OUTPUT.TXT so sánh, tìm điểm sai sót và sửa chữa code.

Họ Tên	Công việc	Hàm cần viết	Mô tả
Hoàng Thị Hoài Thương	Viết hàm tạo chuỗi nhị phân ngẫu nhiên	*string randBinary();*	Tạo 1 dãy nhị phân ngẫu nhiên ở dạng chuỗi và độ dài <= 128
	Viết hàm tạo chuỗi hexa ngẫu nhiên	*string randHexa();*	Tạo 1 dãy hexa ngẫy nhiên ở dạng chuỗi có độ dài <= 32
	Viết hàm tạo chuỗi thập phân ngẫu nhiên	*string randDecimal(string max);*	Tạo 1 dãy thập phân ngẫu nhiên dạng chuỗi và có độ dài <= max với max cũng là 1 chuỗi ở hệ 10. Nếu là số âm thì có trị tuyệt đối < max

Hứa Văn Vũ	Viết hàm phát sinh tệp INPUT.TXT	*void randFileTest(int n);*	Tạo random 1 file test giống file INPUT.TXT mô tả trong đồ án. Với n là số lượng dòng.
Lê Hoàng Vũ	Hướng dẫn, phân công, tổng hợp, chạy thử - kiểm tra và sửa lỗi		

Bảng 4.1: Bảng phân công công việc tạo chương trình phát sinh tệp INPUT.TXT

4.2.2 Lỗi gặp phải – sửa lỗi

Một số lỗi gặp phải và đã sửa:

- Lỗi xuất số '0': Khi QInt == 0 thì hàm xuất hệ nhị phân và thập phân không xuất số '0' mà rỗng
- Lỗi phép xoay: Phép xoay 1 số trường hợp đưa ra kết quả sai
- Lỗi đọc đối số: Một số trường hợp đọc sai 3 và 4 đối số, cuối dòng có khoảng trắng

V. Đánh giá

5.1 Đánh giá mức độ hoàn thành đồ án

Yêu cầu đồ án	Mức độ hoàn thành
Chuyển đổi số QInt từ hệ thập phân sang hệ nhị phân và ngược lại	100%
Chuyển đổi số QInt từ hệ nhị phân sang hệ thập lục phân và ngược lại	100%
Các operator=, operator+, operator-, operator*, operator/	100%
Các toán tử AND "&", OR "|", XOR "^", NOT "~"	100%
Các toán tử: dịch trái "<<", dịch phải ">>"	100%
Các phép xoay trái "rol", xoay phải "ror" mỗi lần xoay chỉ xử lý cho đúng 1 bit	100%

Chuyển đổi số QInt từ hệ thập phân sang hệ nhị phân và ngược lại	100%
TỔNG ĐỒ ÁN	100%

Bảng 5.1: Bảng đánh giá mức độ hành thành đồ án

5.2 Đánh giá thành viên trong nhóm

Họ Tên	Khối lượng công việc	Mức độ hoàn thành
Lê Hoàng Vũ	40%	100%
Hứa Văn Vũ	30%	100%
Hoàng Thị Hoài Thương	30%	100%

Bảng 5.2: Bảng đánh giá thành viên trong nhóm

TÀI LIỆU THAM KHẢO

1. Slide **CTT104_Ch02_Bieu dien so nguyen.pptx**, Khoa Công nghệ thông tin Trường Đại học Khoa học Tự nhiên, ĐHQG Tp.HCM.

2. **Adder (electronics)**: https://en.wikipedia.org/wiki/Adder_(electronics)