TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN TPHCM

KHOA CÔNG NGHỆ THÔNG TIN

**BÁO CÁO MÔN HỌC**

**KIẾN TRÚC MÁY TÍNH VÀ HỢP NGỮ**

Đồ án 1

**BIỂU DIỄN VÀ TÍNH TOÁN SỐ HỌC TRÊN MÁY TÍNH**

Nhóm sinh viên thực hiện: Vũ Lê Thế Anh 1612838

Nguyễn Lê Hồng Hạnh 1612849

Phan Quốc Tuấn 1612902

**Mục lục**

[1. QInt – Biểu diễn số nguyên lớn 128 bit 3](#_Toc510185899)

[1.1. Tổ chức dữ liệu 3](#_Toc510185900)

[1.1.1. Tổ chức các lớp đối tượng 3](#_Toc510185901)

[1.1.2. Tổ chức kiểu dữ liệu Qbit 3](#_Toc510185902)

[1.1.3. Tổ chức kiểu dữ liệu Int 3](#_Toc510185903)

[1.1.4. Tổ chức kiểu dữ liệu QInt 4](#_Toc510185904)

[1.2. Chuyển đổi số nguyên 16 byte sang kiểu dữ liệu QInt 4](#_Toc510185905)

[1.3. Chuyển đổi giữa các hệ cơ số: 5](#_Toc510185906)

[1.3.1. Nhóm hàm nhận vào các hệ cơ số dưới dạng chuỗi 6](#_Toc510185907)

[1.3.2. Nhóm hàm chuyển đổi và biểu diễn ở các hệ cơ số dưới dạng chuỗi 7](#_Toc510185908)

[1.4. Các operator tính toán trên QInt 7](#_Toc510185909)

[1.4.1. Các toán tử trên bit 7](#_Toc510185910)

[1.4.2. Phép cộng (operator +) 8](#_Toc510185911)

[1.4.3. Phép trừ (operator -) 8](#_Toc510185912)

[1.4.4. Phép nhân (operator \*) 8](#_Toc510185913)

[1.4.5. Phép chia (operator /) 9](#_Toc510185914)

[2. QFloat – Biểu diễn số thực lớn 128 bit 11](#_Toc510185915)

[2.1. Tổ chức dữ liệu 11](#_Toc510185916)

[2.1.1. Tổ chức các lớp đối tượng 11](#_Toc510185917)

[2.1.2. Tổ chức kiểu dữ liệu Float 11](#_Toc510185918)

[2.1.3. Tổ chức kiểu dữ liệu QFloat 11](#_Toc510185919)

[2.2. Chuyển đổi số thực lớn giữa các hệ cơ số 13](#_Toc510185920)

[2.2.1. Nhóm hàm nhận vào các hệ cơ số dưới dạng chuỗi 13](#_Toc510185921)

[2.2.2. Nhóm hàm chuyển đổi và biểu diễn các hệ cơ số dưới dạng chuỗi 14](#_Toc510185922)

[2.3. Các operator tính toán trên số thực lớn 15](#_Toc510185923)

[2.3.1. Phép cộng (operator +) 15](#_Toc510185924)

[2.3.2. Phép trừ (operator -) 16](#_Toc510185925)

[2.3.3. Phép nhân (operator \*) 16](#_Toc510185926)

[2.3.4. Phép chia (operator /) 17](#_Toc510185927)

[3. Đánh giá mức độ hoàn thành 18](#_Toc510185928)

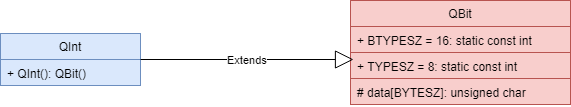
[4. Giao diện chương trình ứng với từng testcase 18](#_Toc510185929)

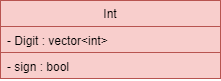
[5. Tài liệu tham khảo 18](#_Toc510185930)

# **QInt – Biểu diễn số nguyên lớn 128 bit**

## **Tổ chức dữ liệu**

### **Tổ chức các lớp đối tượng**





***Hình 1. Tổ chức class***

Class QInt được thừa kế từ 2 class QBit. Class QBit dùng để tổ chức và lưu trữ số nguyên lớn (cụ thể là 16 byte), cùng với các phép tính, phép luận lý liên quan đến bit. QInt lưu trữ các phép chuyển đổi giữa các hệ của số nguyên lớn và các phép tính + - \* /. Còn Int được tổ chức để xử lý số nguyên lớn.

### **Tổ chức kiểu dữ liệu Qbit**

Để biểu diễn số nguyên với độ lớn 16 byte (tương ứng 128 bit), chúng ta có nhiều cách thức khác nhau. Trong đồ án này, chúng tôi chọn cách biểu diễn như sau:



Cấu trúc trên sử dụng mảng một chiều kiểu unsigned char tên là *data,* gồm 16 phần tử, để lưu trữ tổng cộng 128bit của số nguyên 16 byte. Như vậy, với cách tổ chức trên, một phần tử sẽ lưu trữ 8bit *[TYPESZ]*, và tương ứng 16 phần tử *[BYTESZ]*sẽ là 128bit.

Lớp QBit sẽ cài đặt các phương thức trên bit như AND (&), OR (|), XOR (^), NOT (~).

### **Tổ chức kiểu dữ liệu Int**

Lớp Int dùng để nhận và xử lý trên chuỗi cơ số thập phân. Cấu trúc Int gồm có 2 thuộc tính: *Digit* để lưu trữ chuỗi thập phân dưới dạng một vector các số nguyên, và *sign* để lưu trữ dấu.



Lớp Int sẽ cài đặt các phương thức:

* Phép toán +, -, \*, /
* Các phép kiểm tra, so sánh

### **Tổ chức kiểu dữ liệu QInt**

QInt sẽ được thừa kế từ QBit để thuận tiện hơn trong việc thao tác trên bit, cũng như trên số nguyên lớn.



Do QInt biểu diễn số nguyên kích thước bit dưới dạng bù 2nên phạm vi biểu diễn của QInt là:

Với

Lớp QInt sẽ cài đặt các phương thức:

* Nhập dữ liệu từ các hệ cơ số dưới dạng chuỗi
* Biểu diễn dữ liệu trong các hệ cơ số dưới dạng chuỗi
* Phép toán +, -, \*, /
* Các phép kiểm tra, so sánh

## **Chuyển đổi số nguyên 16 byte sang kiểu dữ liệu QInt**

Do số nguyên lớn 16 byte nên ta không thể dùng các lệnh nhập/ xuất trong thư viện C/ C++ để đọc giá trị vừa nhập. Vậy nên, ta chỉ có thể đọc dạy số vừa nhập bằng cách đọc theo chuỗi (string). Để làm được điều đó, ta cần chuyển lần lượt các chuỗi số vừa nhập (được thể hiện dưới dạng chuỗi số nhị phân (binary), hoặc thập lục phân (hexadecimal), hoặc thập phân (decimal) thành kiểu QInt. Khi đó, giá trị vừa nhập sẽ được biểu diễn dưới dạng các bit và được lưu trữ trong *data.*

Một điều cần lưu ý ở đây là các bit sẽ được lưu trữ ngược lại với chiều đọc tự nhiên của con người, nói cách khác, bit quan trọng nhất (most significant bit, MSB) sẽ được lưu ở cuối và ngược lại, bit kém quan trọng nhất (least significant bit, LSB) sẽ được lưu ở đầu.

Để làm việc dễ dàng hơn với bit, 2 hàm hỗ trợ sau đây là rất quan trọng:

* Hàm *setBit* dùng để thay đổi giá trị phần tử tại vị trí *pos* trong QBit thành *bit* (0 hoặc 1)



* Hàm *getBit* dùng để lấy bit tại vị trí *pos* trong QBit

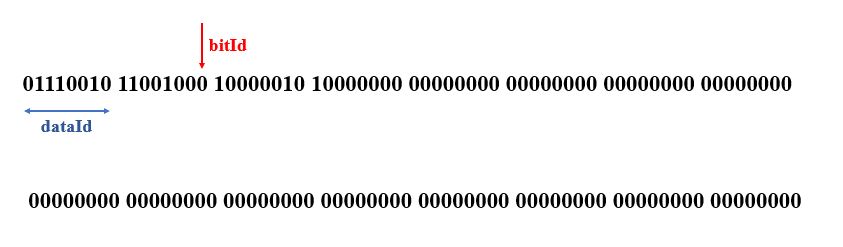


Các bit trong QBit được đánh số từ trái sang phải, bắt đầu từ 0. Vị trí *pos* nhập vào sẽ giúp xác định *dataId* là chỉ số của data và *bitId* là vị trí của bit đó trong *data[dataId]:*

dataId = pos / TYPESZ;

bitId = pos % TYPESZ;

Để dễ dàng hiểu về *dataId, và bitId* biểu thị, chúng ta xét cụ thể một số nhị phân sau:



***Hình 2. Minh họa dataId và bitId***

Một kiểu dữ liệu QInt, gồm 128 bit, sẽ được chia thành 16 vị trí theo thứ tự từ trái sang phải (đánh số từ 0 đến 15), *dataId* là biến chứa chỉ số của các vị trí đó. Trong mỗi dataId gồm 8 bit, được biểu thị bằng bitId, nghĩa là bitId sẽ chỉ đến vị trí cụ thể từng bit trong dataId.

Các hệ cơ số sẽ được nhận vào dưới dạng chuỗi, sau đó được chuyển đổi sang hệ cơ số mong muốn khác, và được biểu diễn ngược lại dưới dạng chuỗi. Để dễ dàng chuyển đổi giữa các hệ cơ số, chúng tôi tổ chức 2 nhóm hàm:

* Nhóm hàm nhận/đọc vào từ các hệ cơ số dưới dạng chuỗi;
* Nhóm hàm biểu diễn các hệ cơ số sau khi chuyển đổi dưới dạng chuỗi.

## **Chuyển đổi giữa các hệ cơ số:**

Đối với QInt, có 3 hệ cơ số xem xét là hệ nhị phân (binary), thập phân (decimal) và thập lục phân (hexadecimal).

Các hệ cơ số sẽ được nhận vào dưới dạng chuỗi, lưu trữ trong thuộc tính data của lớp QInt dưới dạng nhị phân, và khi chuyển đổi sẽ được biểu diễn ngược lại dưới dạng chuỗi. Để dễ dàng chuyển đổi giữa các hệ cơ số, chúng tôi tổ chức 2 nhóm hàm: 1 – nhóm hàm nhận/ đọc vào từ các hệ cơ số dưới dạng chuỗi; 2 – nhóm hàm biểu diễn các hệ cơ số sau khi chuyển đổi dưới dạng chuỗi.

### **Nhóm hàm nhận vào các hệ cơ số dưới dạng chuỗi**

* **Hàm nhận vào chuỗi nhị phân:**



*fromBin* là hàm nhận vào chuỗi nhị phân, sau đó lưu trữ vào QInt. Thứ tự lưu vào QInt như sau, lưu xuống từng dataId một của QInt (từ trái qua), nhưng vị trí bit của chuỗi vừa nhận sẽ được lưu ngược (từ phải qua trái) trong từng dataId. Để lưu được chuỗi nhị phân trong QInt, ta thực hiện phép dịch phải (theo vị trí *bitId*), sau đó thực hiện phép OR.

Ví dụ: Chuỗi binary nhập vào s = ‘10111001’, sẽ cho kết quả dưới dạng QInt là res = ‘10011101 00000000…’, nghĩa là, ứng với từng giá trị trong s, sẽ được lưu ở dataId thứ 0, với thứ tự ngược từ phải sang trái trong dataId (tức bitId duyệt từ 7 trở về 0).

* **Hàm nhận vào chuỗi thập phân**



Hàm *fromDec* được dùng để nhận chuỗi thập phân, được thực hiện như sau:

1. Lưu trữ chuỗi thập phân vào kiểu dữ liệu Int để chuẩn hóa và xử lý
2. Lấy từng giá trị được lưu trong Int chia lần lượt cho 2 🡪 lấy số dư
3. Số dư trên tương ứng với từng bit trong kiểu QInt

Trước tiên, cần lưu trữ chuỗi thập phân vào Int. Chuỗi nhận vào sẽ được kiểm tra là số âm hay số dương; tiếp đến lưu trữ từng giá trị vào *vector Digit* theo chiều từ phải sang.

Một hàm hỗ trợ khác để chuyển chuỗi thập phân sang kiểu dữ liệu QInt là *halved* – dùng để chia 2 số thập phân, lấy số dư (tương ứng với các bit 1 hoặc 0). Hàm này nhận vào là một tham trị, trỏ đến biến *r* – lưu phần dư trong phép tính chia một số cho 2. Trong hàm trên, thay vì dùng phép chia cho 2 (/2), chúng tôi sử dụng phép dịch phải trực tiếp trên bit để tăng tốc độ phép tính. Ví dụ: số 12, lần đầu, lấy 1 chia 2 thì kết quả được 0 và dư 1; lần thứ hai, lấy 12 chia cho 2 (nghĩa là (10 \* dư 1 + 2) / 2).



Nếu số nhập vào là số âm, chuyển sang dạng bù 2 của nó bằng cách nghịch đảo các bit (bù 1) rồi cộng thêm 1.

Cuối cùng, sau khi lấy được phần dư (biến r giữ giá trị), các giá trị ấy sẽ lần lượt được chuyển vào QInt, theo thứ tự từ phải sang trái.

* **Hàm nhận vào chuỗi thập lục phân**



Chuỗi thập lục phân sau khi nhận vào sẽ được chuẩn hóa, chuyển ký tự từ ‘a’ đến ‘f’ thành kí tự in hoa ‘A’ đến ‘F’.

Tiếp đến, lần lượt chuyển từng số trong chuỗi thập lục phân thành QInt dưới dạng nhị phân. Để ý thấy rằng, từng giá trị trong hex chỉ phân bố từ ‘0’ cho đến ‘F’, tương ứng với độ dài bit là 4, nên ta sẽ có vòng lặp for để đặt giá trị bit cho giá trị thập lục phân. Lấy phần dư lần lượt từng giá trị của chuỗi thập lục phân, theo thứ tự từ trái sang phải. Cuối cùng, các bit trong mỗi dataId sẽ được lưu theo thứ tự từ phải sang trái.

### **Nhóm hàm chuyển đổi và biểu diễn ở các hệ cơ số dưới dạng chuỗi**

* **Hàm chuyển từ QInt sang chuỗi nhị phân**



Hàm *toBin()* dùng để chuyển từ kiểu dữ liệu QInt sang chuỗi nhị phân. Đầu tiên, cần một biến *i* để đếm số lượng bit trống (tức không dùng để biểu diễn số nào cả). Theo một nghĩa khác, giá trị của biến *i* chính là vị trí cuối cùng của bit (nhập vào lúc đầu) được biểu diễn trong kiểu dữ liệu QInt, hay còn được hiểu là vị trí bắt đầu để lấy giá trị bit, chuyển sang chuỗi nhị phân. Tiếp đến, lần lượt chuyển QInt thành chuỗi nhị phân, tính từ vị trí *i.*

Ví dụ: Với một chuỗi nhập vào là s = ‘2104’, chuyển sang dạng QInt là res = ‘00011100 00010000 0000000000 …’. Sau khị thực hiện vòng lặp để đếm, i sẽ có giá trị là 116, như vậy, chuỗi nhị phân in ra sẽ được lật ngược lại (tính từ i đến 128) 🡪 kết quả chuỗi nhị phân là: **‘100000111000’.**

* **Hàm chuyển từ QInt sang chuỗi thập phân**



Hàm *toDec()* dùng để chuyển đổi từ QInt sang chuỗi thập phân. Để tính số thập phân, dùng kiểu Int, với ý tưởng như sau: tính giá trị thập phân của biểu diễn dạng bù 2, sử dụng công thức sau:

Ví dụ:

Do phép tính có thể trả về số lớn nên sử dụng lớp Int để hỗ trợ cho việc này.

* **Hàm chuyển từ QInt sang chuỗi thập lục phân**



Hàm *toHex()* dùng để chuyển đổi từ QInt sang chuỗi thập lục phân. Như ở trên đã nói, do những giá trị của thập lục phân khi chuyển sang nhị phân chỉ có độ dài 4 bit, nên ta dùng vòng lặp, lặp lại 32 lần để chuyển QInt thành chuỗi thập lục phân.

## **Các operator tính toán trên QInt**

### **Các toán tử trên bit**

Để thực hiện các phép tính + - \* / giữa 2 số kiểu QInt dễ dàng, ta cần đến các operator của các phép luận lý trên bit: phép dịch phải (>>), phép dịch trái (<<), phép and (&), phép or (|), phép xor (^), phép not (~).

Các phép AND, OR, XOR, NOT có thể thực hiện bằng cách AND, OR, XOR, NOT từng phần tử của thuộc tính data.

Các phép dịch trái, dịch phải k lần được thực hiện bằng cách lấy bit ở vị trí cách đó k đơn vị getBit() (trả về nếu ngoài mảng) và đặt ở vị trí tương ứng setBit().

### **Phép cộng (operator +)**

Để cộng 2 số QInt, ta thực hiện cộng từng bit, đồng thời có một biến để lưu trữ giá trị bit dư (tạm gọi là biến leftover). Ví dụ, để thực hiện phép tính: 10 + 15, ta làm như sau

1. Chuyển cả 5 và 3 về kiểu dữ liệu QInt: 1010 = 1010, 1510 = 1111
2. Thực hiện cộng từng bit một từ phải qua trái, lưu trữ vào biến leftover (nếu có phần dư khi cộng):

* 0 + 1 = 1, leftover = 0, result = 1
* 1 + 1 = 0, leftover = 1, result = 01
* 1 + 0 + 1 (leftover) = 0, leftover = 1, result = 001
* 1 + 1 + 1 (leftover) = 1m leftover = 1, result = 1001

Result = 11001 (cộng thêm leftover dư ở đầu)

Do đó, ta cần dùng phép dịch phải và phép AND với 1 để thực hiện cộng 2 bit.

### **Phép trừ (operator -)**

Phép toàn trừ, trên thực tế là ngược lại với phép cộng 2 bit, do đó thay vì lấy A trừ B, thì ta thực hiện phép tính A + (-B). Có nhiều cách thức biểu diễn số âm trong nhị phân, ở đây chúng tôi sử dụng số bù 2 để biểu diễn.

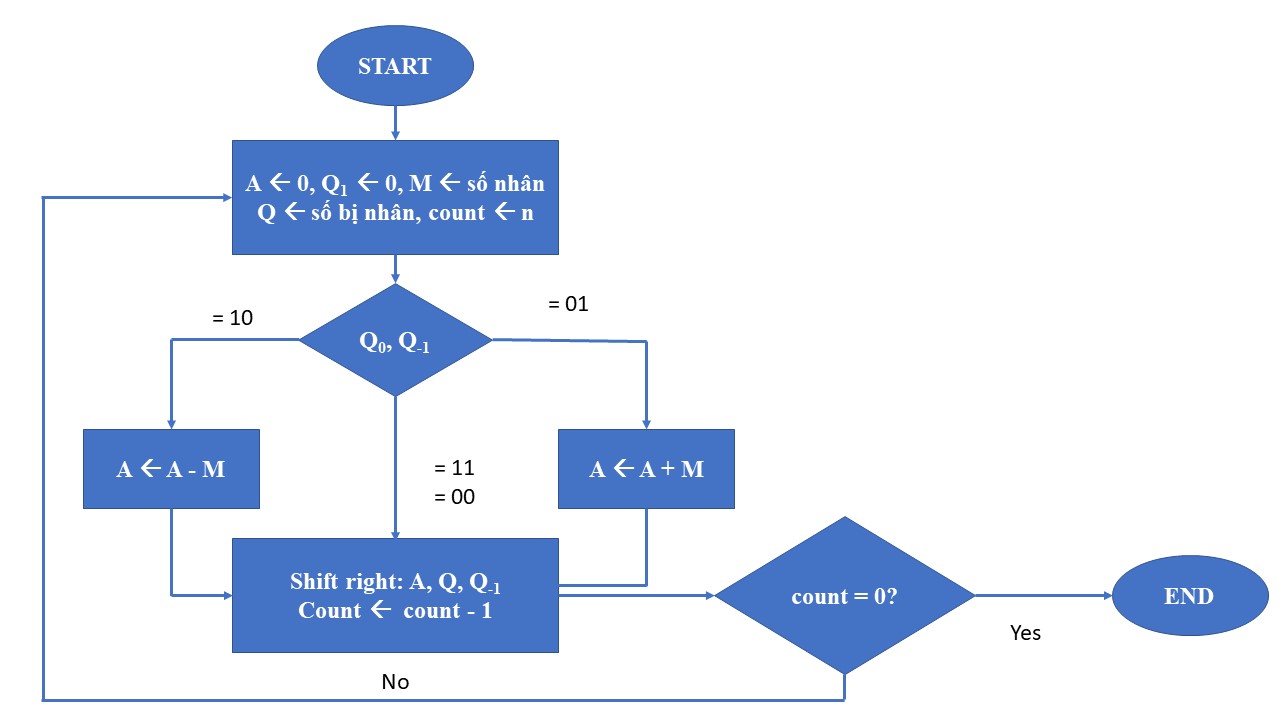
Ví dụ: để thực hiện phép tính 15 – 10, ta làm như sau

1. Thay vì thực hiện phép trừ, ta thực hiện phép cộng 15 + (-10)
2. Biểu diễn cả 2 số dưới dạng nhị phân. Đặc biệt số -10 biểu diễn dưới dạng bù 2: 1510 = 1111, -1010 = 0110
3. Thực hiện cộng 2 số tương tự như trên.

Do đó, ta chỉ cần dùng phép NOT để lật ngược bit, sau đó cộng thêm 1, rồi thực hiện phép tính cộng bit như trên.

### **Phép nhân (operator \*)**

Sử dụng thuật toán Booth để nhân 2 số QInt. Ưu điểm của thuật toán Booth là biểu diễn số âm một cách chính xác.



***Hình 3. Lưu đồ thuật toán Booth***

Ví dụ: thực hiện phép nhân 12 x -5

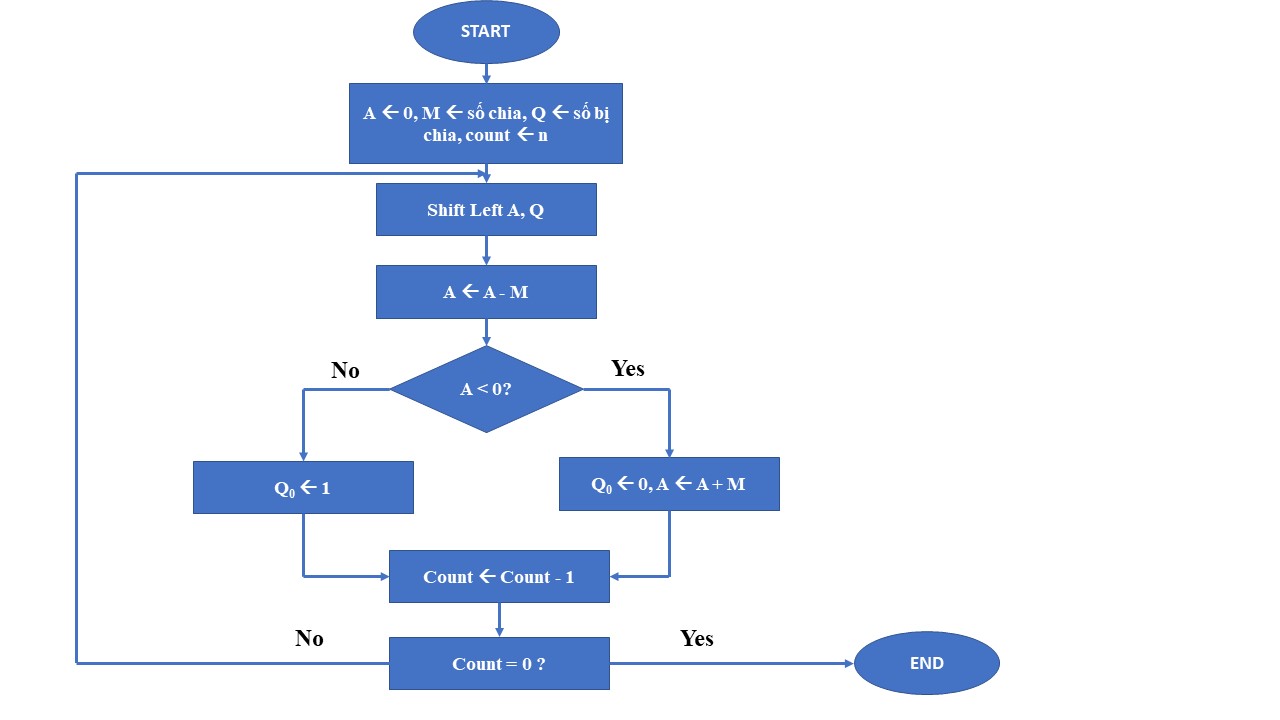
1. Chuyển đổi lần lượt 2 số ra nhị phân: 12 = 01100; -5 = 11011
2. Xác định các giá trị: A = 00000, M = 01100, Q = 11011, Q-1 = 0, count = 5
3. Thực hiện phép nhân Booth theo lưu đồ thuật toán:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| A | Q | Q-1 | M | Thực hiện |
| 00000 | 11011 | 0 | 01100 | QQ-1 = 10, count = 5 |
| 10100 | 11011 | 0 | 01100 | A = A - M |
| 11010 | 01101 | 1 | 01100 | Shift Right, count = 4, QQ-1 = 11 |
| 11101 | 00110 | 1 | 01100 | Shift Right, count = 3, QQ-1 = 01 |
| 01001 | 00110 | 1 | 01100 | A = A + M |
| 00100 | 10011 | 0 | 01100 | Shift Right, count = 2, QQ-1 = 10 |
| 11000 | 10011 | 0 | 01100 | A = A – M |
| 11100 | 01001 | 1 | 01100 | Shift Right, count = 1, QQ-1 =11 |
| 11110 | 00100 | 1 | 01100 | Shift Right, count = 0 🡪 Kết thúc |

Như vậy, kết quả của phép nhân là: 1111000100.

### **Phép chia (operator /)**

Thực hiện phép chia không dấu (nếu số chia hoặc số bị chia là số âm, ta dùng một một biến lưu dấu, và lấy trị tuyệt đối của nó) giữa 2 số QInt.



***Hình 4. Lưu đồ thuật toán cho phép chia không dấu.***

Ví dụ: thực hiện phép tính 7 / 3

1. Đổi lần lượt các số ra dạng nhị phân: 7 = 0111, 3 = 0011
2. Xác định các giá trị: A 🡨 0000, Q 🡨 0111, M 🡨 0011
3. Thực hiện phép chia theo lưu đồ thuật toán:

|  |  |  |  |
| --- | --- | --- | --- |
| A | Q | M | Thực hiện |
| 0000 | 0111 | 0011 | Khởi tạo giá trị ban đầu |
| 0000 | 1110 | 0011 | Shift left |
| 1101 |  |  | A 🡨 A - M |
| 0000 | 1110 | 0011 | Phục hồi |
| 0001 | 1100 | 0011 | Shift left |
| 1110 |  |  | A 🡨 A – M |
| 0001 | 1100 |  | Phục hồi |
| 0011 | 1000 |  | Shift left |
| 0000 |  |  | A 🡨 A – M |
| 0000 | 1001 |  | Set Q0= 1 |
| 0001 | 0010 |  | Shift left |
| 1110 |  |  | A 🡨 A – M |
| 0001 | 0010 |  | Phục hồi, count = 0 |

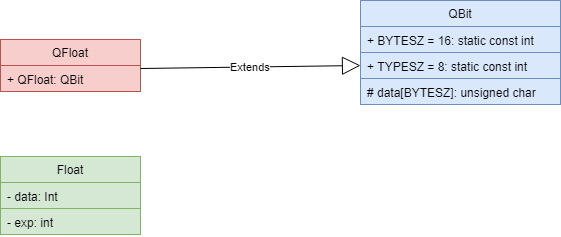
Như vậy kết quả phép chia là: **Thương là: Q = 0010, Số dư là: A = 0001**

Nếu phép chia là âm, thì cứ chia như không dấu, khi ra kết quả, chỉ việc thêm dấu trừ (-) vào trước kết quả.

# **QFloat – Biểu diễn số thực lớn 128 bit**

## **Tổ chức dữ liệu**

### **Tổ chức các lớp đối tượng**



***Hình 5. Tổ chức class để biểu diễn số thực lớn***

Cũng giống như class QInt, class QFloat được thừa kế từ class QBit, do đó số thực lớn sẽ sử dụng các phép tính, phép luận lý liên quan đến bit, trong khi đó QFloat lưu trữ các phép chuyển đổi giữa các hệ của số thực lớn và các phép tính + - \* /.

Còn Float được tổ chức để xử lý số nguyên lớn và vị trí dấu chấm động của số thực. Trong Float, thuộc tính *data* sử dụng kiểu dữ liệu Int, nghĩa là phần số nguyên được lưu trữ trong vector.

### **Tổ chức kiểu dữ liệu Float**

Lớp Float dùng để xử lý phần nguyên lớn và lưu trữ vị trí dấu chấm động ‘.’ của số thực. Cấu trúc Float gồm 2 thuộc tính: *data* có kiểu dữ liệu Int, nên các giá trị phần nguyên sẽ được lưu vào vector; *exp* để lưu trữ vị trí dấu chấm động



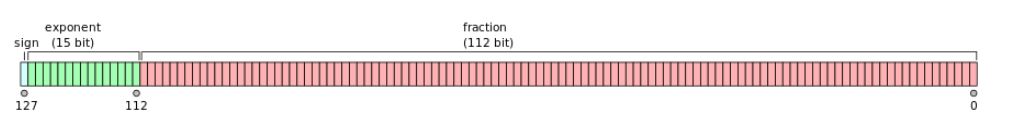
Lớp Float sẽ cài đặt các phương thức:

* Phép toán +, -, nhân đôi, chia đôi
* Các phép kiểm tra, so sánh

### **Tổ chức kiểu dữ liệu QFloat**

Số thực lớn 128 bit (Quadruple-precision floating-point format) gồm

* Bit biểu diễn dấu (sign): 1 bit (1 nếu âm, 0 nếu dương)
* Bit biểu diễn mũ (): 15 bit
* Bit biểu diễn phần thập phân : 112 bit



***Hình 6. Biểu diễn số thực 128 bits***

Số thực biểu diễn như trên trong hệ thập phân có dấu phụ thuộc vào bit đầu và độ lớn:

Trong đó, số mũ nằm trong phạm vi

Sở dĩ có phạm vi trên là đã xét đến các trường hợp đặc biệt như sau:

* Số 0: , (tức toàn bộ bit đều là 0, trừ bit dấu)
* Số không chuẩn: (tức toàn bộ bit mũ là 0, bit trị khác 0)
* Số không chuẩn có dạng thập phân:
* Số vô cực : , (tức toàn bộ bit mũ là 1, bit trị bằng 0)
* Số báo lỗi : , (tức toàn bộ bit mũ là 1, bit trị khác 0)

Do tính đối xứng trên trục số của giới hạn biểu diễn, ta chỉ xét phần dương, phần âm sẽ tương tự và ngược lại (đổi dấu, tối đa và tối thiểu). Giới hạn biểu diễn của số thực chuẩn trên như sau:

* Số mũ tối đa là và phần thập phân tối đa khi toàn bộ bit đều bật lên 1, do đó tối đa của nó là
* Số mũ tối thiểu là và phần thập phân tối thiểu bằng 0, do đó tối thiểu của nó là

Xét trường hợp số không chuẩn, số mũ luôn bằng :

* Phần thập phân tối đa khi toàn bộ bit bật lên 1 nên tối đa của nó là
* Phần thập phân tối thiểu của nó khi toàn bộ bit trị bằng 0 trừ bit cuối cùng, do đó tối thiểu của nó là

Vậy phạm vi biểu diễn của số thực như trên là (số âm tương tự):

Như vậy, cũng giống như QInt, để biểu diễn số thực lớn, kiễu dữ liệu QFloat cũng được thừa kế từ QBit:



Lớp QFloat sẽ cài đặt các phương thức:

* Nhập dữ liệu từ các hệ cơ số dưới dạng chuỗi
* Biểu diễn dữ liệu trong các hệ cơ số dưới dạng chuỗi
* Phép toán +, -, \*, /
* Các phép kiểm tra, so sánh

## **Chuyển đổi số thực lớn giữa các hệ cơ số**

Các hệ cơ số sẽ được nhận vào dưới dạng chuỗi, sau đó được chuyển đổi sang hệ cơ số mong muốn khác, và được biểu diễn ngược lại dưới dạng chuỗi. Để dễ dàng chuyển đổi giữa các hệ cơ số, chúng tôi tổ chức 2 nhóm hàm: 1 – nhóm hàm nhận/ đọc vào từ các hệ cơ số dưới dạng chuỗi; 2 – nhóm hàm biểu diễn các hệ cơ số sau khi chuyển đổi dưới dạng chuỗi.

### **Nhóm hàm nhận vào các hệ cơ số dưới dạng chuỗi**

* **Hàm nhận vào chuỗi nhị phân**



Hàm *QFloatfromBin* nhận vào chuỗi nhị phân, sau đó trả về kiểu QFloat bằng cách dùng hàm setbit.

Có hai dạng biểu diễn có thể nhập vào:

1. Chuẩn IEEE: 128 bit gồm 1 bit dấu, 15 bit mũ, 112 bit trị
2. Dạng nhị phân với dấu chấm

* **Hàm nhận vào chuỗi thập phân**

****

Hàm *strDecToQFloat* nhận vào chuỗi thập phân và trả về kiểu QFloat, các bit sẽ được lưu theo chiều thuận (từ trái sang phải). Khi chuyển chuỗi thập phân thành các bit, cần lưu ý 2 trường hợp đặc biệt sau:

TH1: chuỗi thập phân là số âm, khi đó cần xét kí tự đầu tiên có phải là ‘-‘ hay không.

TH2: chuỗi thập phân có dạng sau: **0.0**.

Để chuyển chuỗi thập phân sang kiểu dữ liệu QFloat, ta thực hiện như sau:

1. Kiểm tra các dạng đặc biệt của chuỗi thập phân.

* Nếu đó là số âm, thì đặt bit 1 vào vị trí đầu tiên của QFloat (bằng cách dùng hàm setbit đã nói ở trên)
* Để kiểm tra số đầu vào có dạng **0.0** hay không, ta dùng hàm isZeroStr để kiểm tra; nếu phải thì đặt tất cả các bit là 0.



1. Khi không thuộc 2 trường hợp đặc biệt trên, tiến hành chuyển đổi số nhị phân thành bit, dùng hàm:



Tách chuỗi đầu vào thành 2 phần: phần số nguyên *(intPart)* và phần thập phân *(fracPart)*.

Đối với intPart, lần lượt chuyển thành nhị phân giống với số nguyên lớn, bằng cách lấy số dư khi chia 2.

Đối với fracPart (có dạng ), lần lượt lấy phần thập phân nhân cho 2, lấy số trước dấu chấm (0 hoặc 1) chính là bit tương ứng rồi bỏ số này đi. Ví dụ: cho phần thập phân tương ứng là

Tuy nhiên, cần chú ý tới 2 trường hợp: có phần intPart, và không có phần intPart, trong khi chuyển thành nhị phân của fracPart.

* Có phần intPart, chuyển phần fracPart thành nhị phân như bình thường.
* Không có phần intPart, cần tìm số bit phần thập phân lặp lại, sau đó chuyển thành nhị phân như bình thường.

1. Cuối cùng, cộng intPart và fracPart lại với nhau, và chuyển về kiểu dữ liệu QFloat.

### **Nhóm hàm chuyển đổi và biểu diễn các hệ cơ số dưới dạng chuỗi**

* **Hàm chuyển QFloat thành chuỗi nhị phân**

****

Hàm *DecToBinary* chuyển thành chuỗi nhị phân bằng cách chuyển lần lượt các bit thành chuỗi khi dùng phép dịch phải và phép AND với 1.

Có hai dạng biểu diễn:

1. Chuẩn IEEE: 128 bit gồm 1 bit dấu, 15 bit mũ, 112 bit trị
2. Dạng nhị phân với dấu chấm

* **Hàm chuyển QFloat thành chuỗi thập phân**

****

Hàm *toDec* trả về một chuỗi thập phân. Mã giả sau thể hiện cách chuyển đổi thành chuỗi thập phân:

Gọi là số thập phân cần tìm

Tìm i là vị trí bit 1 đầu tiên

// Tìm

Nếu ()

Ngược lại

//

Lặp ():

//

if (bit[i])

## **Các operator tính toán trên số thực lớn**

### **Phép cộng (operator +)**



Xét một số trường hợp đặc biệt:

Mã giả dưới đây mô tả ý tưởng thực hiện cộng 2 số thực lớn:

Gọi A và B lần lượt là 2 số thực lớn, expA và expB là số mũ của A và B, signA và signB là dấu của A và B:

1. Kiểm tra A hoặc B có bằng 0? 🡪 trả về số khác 0
2. Lấy giá trị expA và expB, signA và signB
3. Lấy fracPart1 và fracPart2 là phần trị của A, B đưa vào vector
4. Nếu (exp­1 < exp2): đưa mũ lớn hơn lên trước
5. Nếu (exp­1 = exp2):

Nếu (sign1 != sign2): thực hiện trừ 2 số, trả về *str*

Nếu (str là số bù 2): lấy bù 2 lần nữa để trả về ban đầu, rồi shift left để đưa

về dạng chuẩn

Nếu (str không là số bù 2):

While (str[i] != 1) shift left để đưa về dạng chuẩn

if (i > str.size) exp = 0 (số không chuẩn)

else: (exp có dạng 0.1xxx) shift left để đưa về 0.xxx

Dấu bằng dấu số lớn hơn

Nếu (sign1 == sign2): cộng A và B như bình thường

1. Nếu (exp­1 != exp2):

Shift right để cân bằng số mũ

If (sign1 != sign2) trừ 2 số, còn nhớ thì bỏ số 1 trong phần nguyên

Else: cộng 2 số, còn nhớ thì shift right

1. Trả về kết quả

### **Phép trừ (operator -)**



Bản chất của phép trừ chính là phép cộng. Ví dụ, khi lấy 15.45 – 12.1, nghĩa là 15.45 + (-12.1). Do đó, để thực hiện phép trừ, ta chỉ việc đổi dấu số trừ, sau đó gọi là hàm phép cộng.

### **Phép nhân (operator \*)**



Xét một số trường hợp đặc biệt:

Hai toán hạng trong phép nhân được biểu diễn dưới dạng (bỏ qua dấu) với nếu là số chuẩn và nếu là số ở dạng không chuẩn.

Ta dời dấu chấm sang trái 2 đơn vị đồng thời tăng số mũ lên nhằm cho số có dạng . Mục đích của việc này là đồng nhất cách trình bày giữa hai chuỗi bit và thêm ở đầu để thuật toán Booth (là thuật toán nhân có dấu) chạy đúng. Tách phần sau dấu chấm, ta có hai chuỗi bit độ dài để thực hiện phép nhân theo thuật toán Booth như trình bày trong phần số nguyên.

Số mũ kết quả bằng tổng hai số mũ: , phép cộng được thực hiện ở dạng biểu diễn trên Bias nên số mũ thực tế là với .

Kết quả phép tính là số có dạng . Số này có thể được chuẩn hóa bằng cách tìm đến bit đầu tiên, đồng thời giảm đi.

Khi tìm thấy bit , bỏ bit này và lấy bit sau đó. Nếu đã xuống đến (tức ) mà vẫn chưa tìm thấy, đây là số không chuẩn, lấy tiếp tục bit tiếp theo.

### **Phép chia (operator /)**



Xét một số trường hợp đặc biệt:

Ta thực hiện phép chia bằng cách:

Đầu tiên, số mũ bằng hiệu hai số mũ, do đây là phép trừ ở dạng biểu diễn bias nên cần thêm phép cộng ngay sau:

Cân bằng số mũ ở hai số bằng cách dịch trái (gấp đôi) tử số cho đến khi hai số mũ bằng nhau.

Thực hiện lần lượt phép chia hai phần trị cho đến khi lấy đủ 112 bit. Ở mỗi lượt chia lấy phần thương số ghép vào phần trị, phần dư được nhân đôi và tiếp tục chia cho số chia.

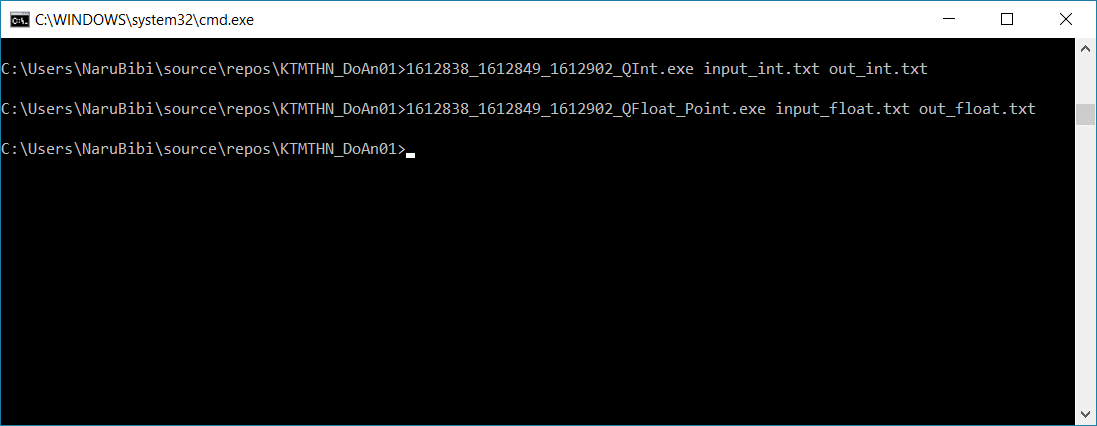
Trong trường hợp chuỗi kết quả vẫn chưa có bit 1 nào, tiếp tục thực hiện phép chia cho đến khi phần mũ trở về 0 (dạng không chuẩn), lấy 112 bit kế tiếp làm kết quả.

# **Đánh giá mức độ hoàn thành**

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Yêu cầu** | **Tỉ lệ hoàn thành (%)** | **Thiếu sót** |
| **QInt** | Nhập, xuất | 100 |  |
| Chuyển đổi giữa các hệ cơ số | 100 |  |
| Toán tử +, -, \*, / | 100 |  |
| Toán tử &, |, ^, ~ | 100 |  |
| Toán tử <<, >> | 100 |  |
| **QFloat** | Nhập, xuất | 100 |  |
| Chuyển đổi giữa các hệ cơ số | 100 |  |
| Operator +, -, \*, / | 100 |  |
| Xử lý số không chuẩn | 90 | Chưa xử lý trong trường hợp các phép tính +, - |

# **Giao diện chương trình ứng với từng testcase**

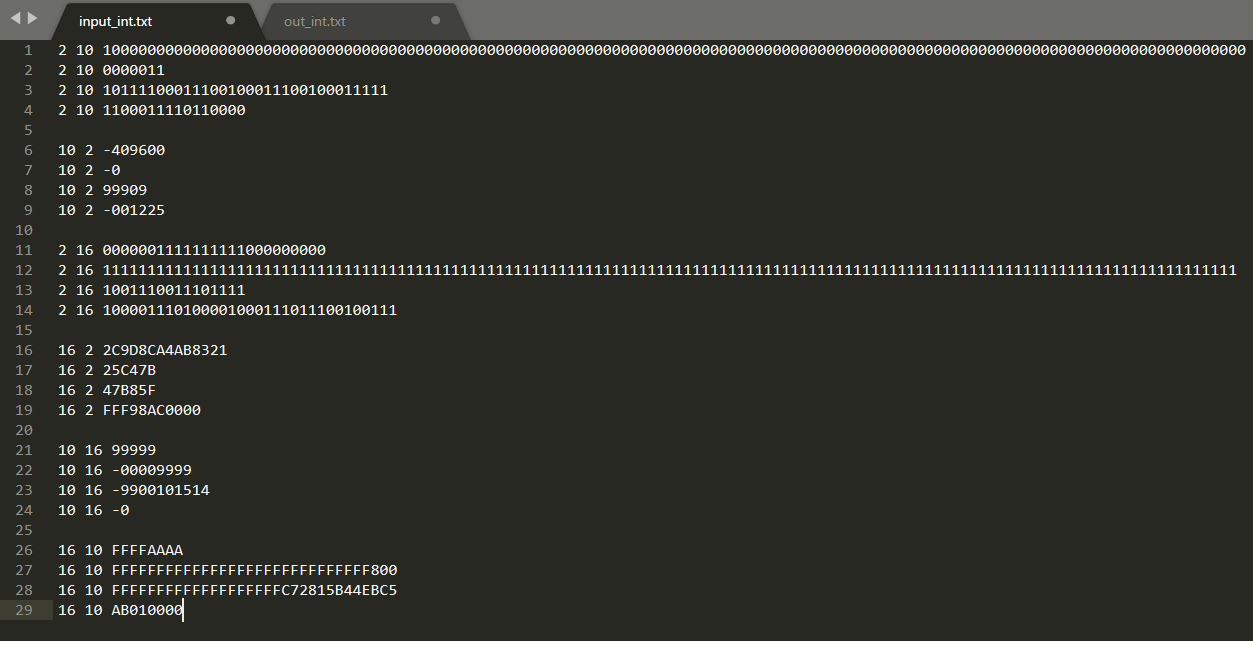
**Console**



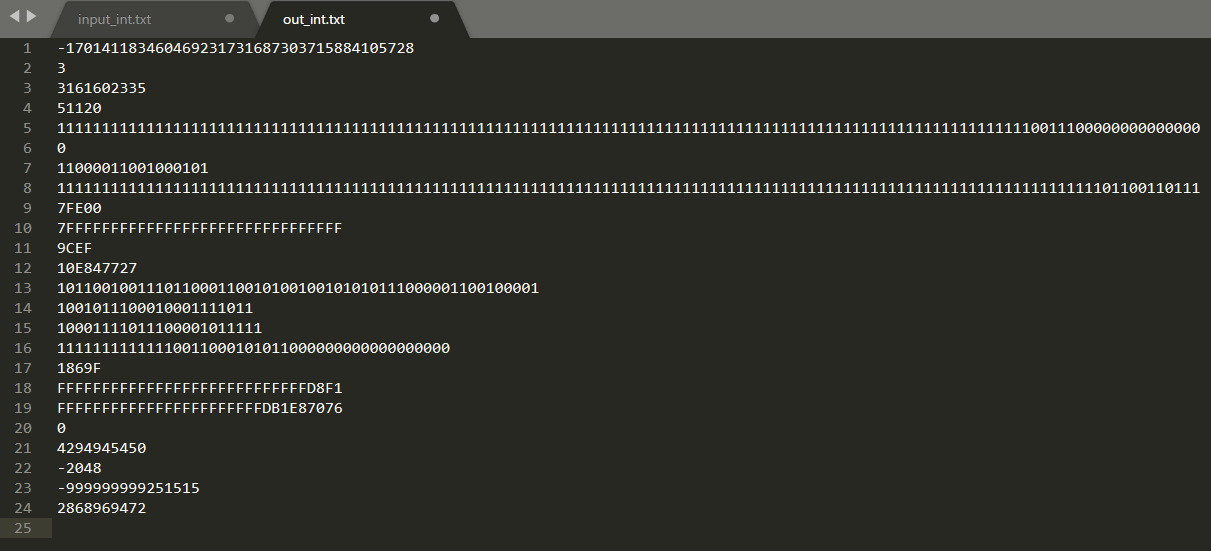
***Hình 7. Giao diện và lệnh console***

**QInt**

**Chuyển đổi giữa các hệ cơ số**

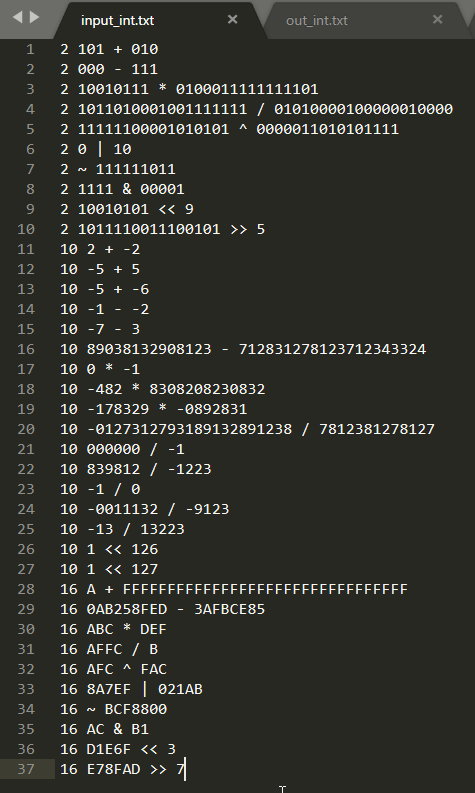
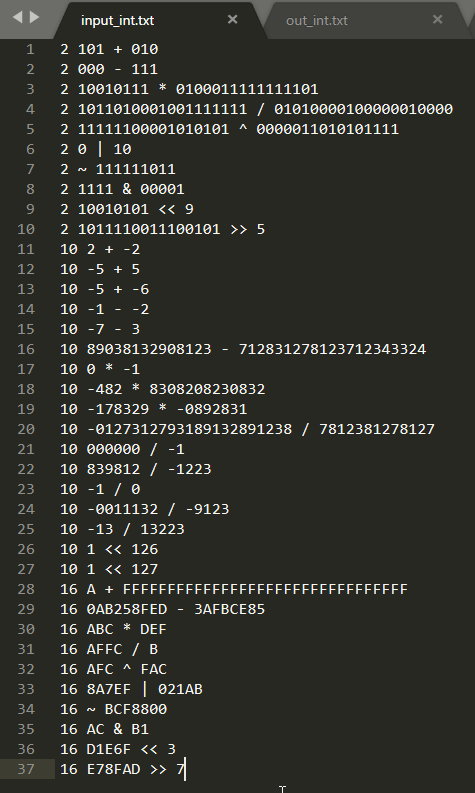


***Hình 8. Input chuyển đổi cơ số QInt***

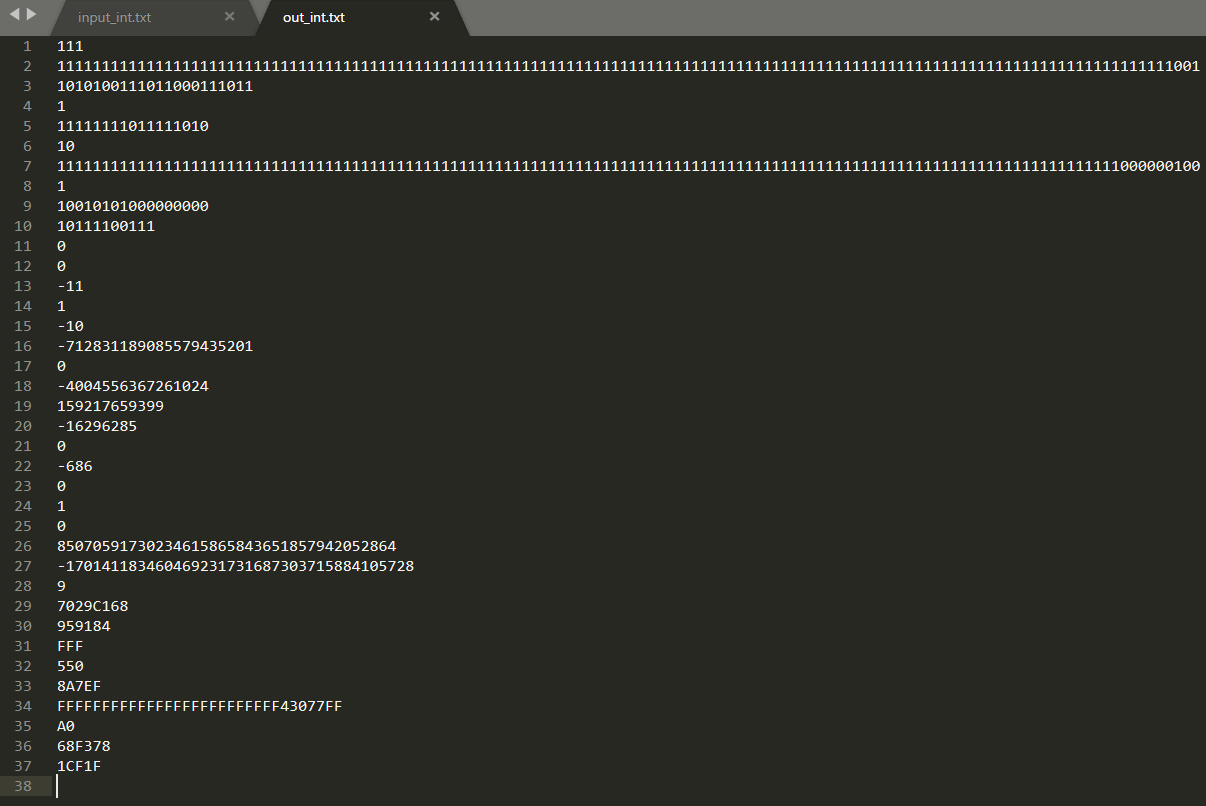


***Hình 9. Kết quả chuyển đổi cơ số QInt***

**Toán tử**

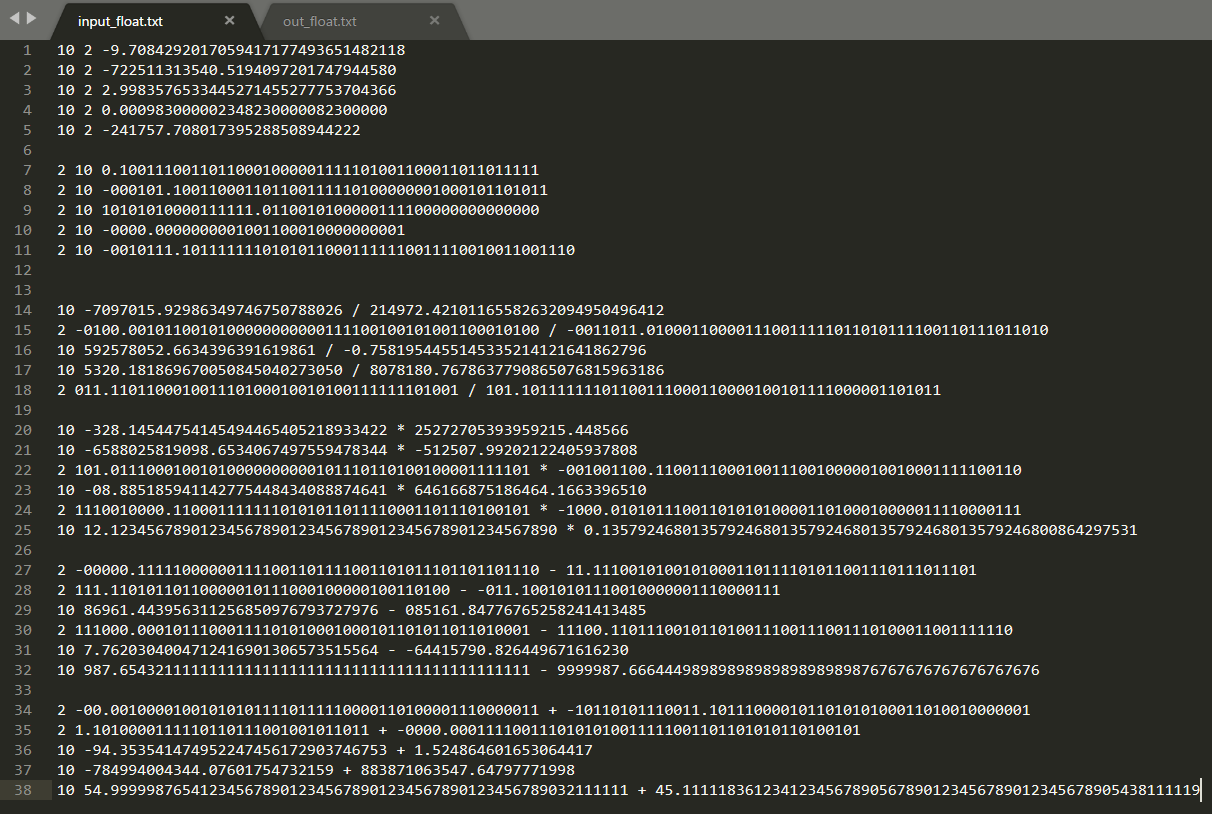


***Hình 10. Input các toán tử QInt***

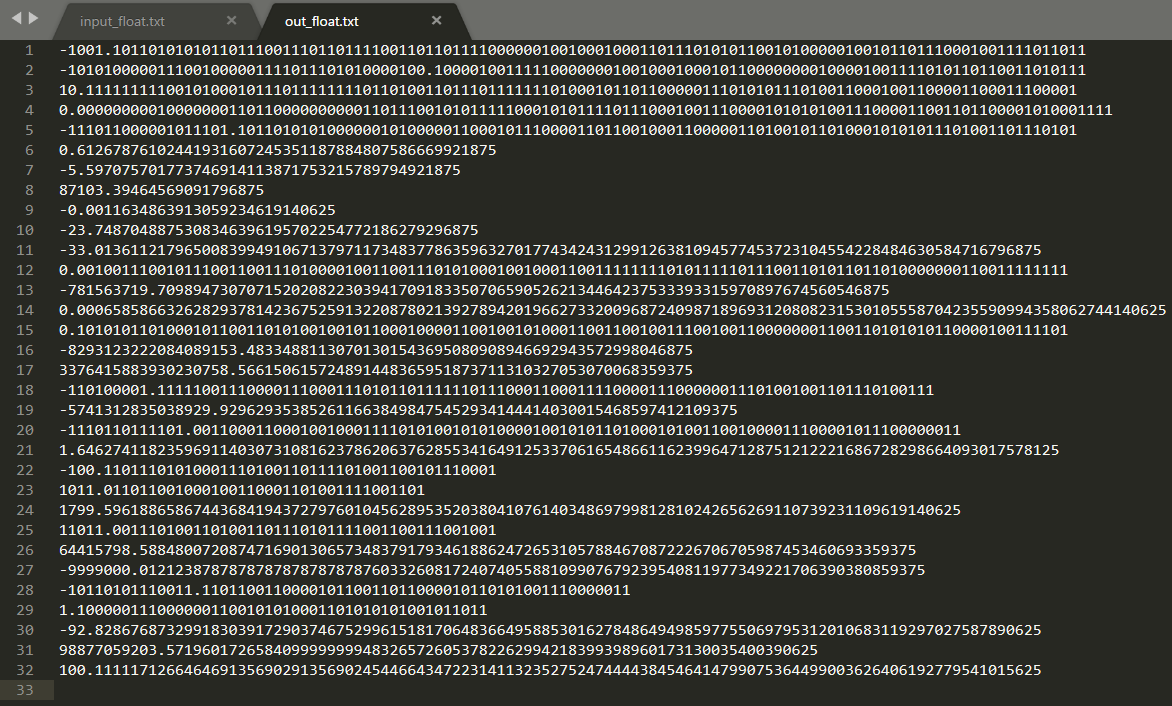


***Hình 11. Kết quả các toán tử QInt***

**QFloat**



***Hình 12. Input QFloat***



***Hình 6. Output QFloat***

# **Tài liệu tham khảo**

1. <https://en.wikipedia.org/wiki/Scientific_notation>
2. <https://en.wikipedia.org/wiki/Quadruple-precision_floating-point_format>
3. <https://www.facebook.com/BigOCoding/>
4. Slide môn học Kiến trúc Máy tính & Hợp ngữ, Phạm Tuấn Sơn, ĐHKHTN TPHCM