**ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH**

**TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN**

**KHOA CÔNG NGHỆ THÔNG TIN**



**ĐỒ ÁN 01 MÔN KIẾN TRÚC MÁY TÍNH VÀ HỢP NGỮ**

Đề tài:

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

Người thực hiện:

Nguyễn Hoàng Thái Dương (18120336)

Trần Thanh Quang (18120230)

Trương Trọng Lộc (18120197)

Giảng viên hướng dẫn:

Thầy ThS. Lê Viết Long

Thành phố Hồ Chí Minh – Tháng 03, 2020

MỤC LỤC

[MỤC LỤC 2](#_Toc40444798)

[1. THÀNH VIÊN VÀ PHÂN CHIA CÔNG VIỆC: 4](#_Toc40444799)

[2. BIỂU DIỄN SỐ NGUYÊN LỚN 128 BIT 4](#_Toc40444800)

[2.1. Thiết kế cấu trúc dữ liệu QInt: 4](#_Toc40444801)

[2.2. Chuyển đổi giữa các hệ cơ số: 4](#_Toc40444802)

[2.2.1. Ý tưởng chuyển đổi: 4](#_Toc40444803)

[2.2.2. Nhóm hàm SetBit, GetBit: 4](#_Toc40444804)

[2.2.3. Chuyển đổi từ chuỗi nhị phân (string) sang QInt 5](#_Toc40444805)

[2.2.4. Chuyển đổi từ chuỗi thập phân (string) sang QInt 5](#_Toc40444806)

[2.2.5. Chuyển đổi từ chuỗi thập lục phân (string) sang QInt 6](#_Toc40444807)

[2.2.6. Chuyển đổi từ chuỗi ở hệ cơ số 2, 10, 16 (string) sang QInt 6](#_Toc40444808)

[2.2.7. Chuyển đổi từ QInt sang chuỗi nhị phân (string) 6](#_Toc40444809)

[2.2.8. Chuyển đổi từ QInt sang chuỗi thập phân (string) 6](#_Toc40444810)

[2.2.9. Chuyển đổi từ QInt sang chuỗi thập lục phân (string) 7](#_Toc40444811)

[2.2.10. Chuyển đổi từ QInt sang chuỗi ở hệ cơ số 2, 10, 16 (string) 7](#_Toc40444812)

[2.2.11. Chuyển đổi từ chuỗi ở hệ cơ số a sang chuỗi ở hệ cơ số b (a, b = 2, 10, 16) 7](#_Toc40444813)

[2.3. Các nhóm toán tử: 7](#_Toc40444814)

[2.3.1. Toán tử cộng: 7](#_Toc40444815)

[2.3.2. Toán tử trừ: 7](#_Toc40444816)

[2.3.3. Toán tử nhân: 7](#_Toc40444817)

[2.3.4. Toán tử chia: 7](#_Toc40444818)

[2.3.5. Toán tử so sánh và gán: 7](#_Toc40444819)

[2.3.6. Toán tử AND, OR, XOR, NOT: 7](#_Toc40444820)

[2.3.7. Toán tử dịch trái, dịch phải: 7](#_Toc40444821)

[2.3.8. Toán tử xoay trái, xoay phải: 7](#_Toc40444822)

# THÀNH VIÊN VÀ PHÂN CHIA CÔNG VIỆC:

|  |  |  |  |
| --- | --- | --- | --- |
| **STT** | **MSSV** | **Họ và tên** | **Công việc** |
| 1 | 18120336 | Nguyễn Hoàng Thái Dương |  |
| 2 | 18120230 | Trần Thanh Quang |  |
| 3 | 18120197 | Trương Trọng Lộc |  |

# BIỂU DIỄN SỐ NGUYÊN LỚN 128 BIT

## Thiết kế cấu trúc dữ liệu QInt:

* Số nguyên lớn QInt bao gồm 128 bit được lưu trữ bởi mảng *data* gồm 4 phần tử kiểu số nguyên. Ở phương thức khởi tạo mặc định, tất cả 128 bit này đều là bit 0.
* Vì 1 phần tử kiểu số nguyên có kích thước 4 bytes nên ta sử dụng 4 phần tử để biểu diễn (thỏa yêu cầu 16 bytes như đề bài).

## Chuyển đổi giữa các hệ cơ số:

### Ý tưởng chuyển đổi:

Đầu vào sẽ là một chuỗi String ở dạng nhị phân / thập lục phân / thập phân. Sau đó ta lưu chuỗi String này vào QInt bằng các hàm chuyển đổi. Từ QInt ta chuyển sang lại dạng chuỗi ở các dạng nhị phân / thập lục phân / thập phân. Như vậy, ta có thể chuyển từ chuỗi String từ hệ cơ số a sang chuỗi String ở hệ cơ số b dựa vào trung gian là QInt.

### Nhóm hàm SetBit, GetBit:

* Lưu ý về việc lưu các dãy bit. Ta xét ví dụ sau: Giả sử ta có chuỗi 8 bit. Khi đó, đối với dạng chuỗi (string), vị trí từng bit, lúc này mỗi bit là phần tử của string nên sẽ được đánh số thứ tự từ trái sang phải. Tuy nhiên, khi lưu trữ dãy bit, ta đánh số từ phải sang trái. Giả sử ta lưu dãy 8 bit này vào 1 mảng gồm 2 phần tử có kích thước 4 bit, ta lưu:

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Vị trí bit (dạng string) | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| Vị trí bit (khi lưu trữ) | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| Mảng 2 phần tử (mỗi phần tử 4 bit) | A[0] | | | | A[1] | | | |
| Vị trí bit ở mỗi phần tử trong mảng | 3 | 2 | 1 | 0 | 3 | 2 | 1 | 0 |

* Để dễ hiểu, ta xét ví dụ, dãy 8 bit 1100 1011 ở dạng chuỗi (string), khi đó phần tử str[0] = ‘1’ sẽ tương ứng với vị trí bit thứ 7 khi lưu trữ. Khi đó bit này sẽ được lưu trữ vào A[1] ở vị trí thứ 3. Sở dĩ là A[1] vì bit lưu trữ ở vị trí thứ 7 chia cho kích thước một phần tử là 4, lấy phần nguyên ta được 1; ở vị trí thứ 3 trong A[1] thì 7 chia 4 lấy phần dư là 3. Từ đó, ta thấy với dãy 1100 1011 thì A[0] gồm dãy 4 bit 1011 và A[1] gồm dãy 4 bit 1100.
* Tổng quát để xác định vị trí bit lưu ở đâu trong QInt dựa vào vị trí bit khi lưu trữ, ta làm như sau:
  + Xác định *block* dựa vào vị trí bit khi lưu trữ bằng cách lấy vị trí đó chia cho kích thước của một *block*. Với QInt, mỗi *block* là kiểu số nguyên nên kích thước là 32 bit.
  + Xác định vị trí bit trong *block* bằng cách lấy vị trí bit khi lưu trữ chia lấy dư cho kích thước của một *block.*
* Sau khi xác định được vị trí bit trong QInt, ta tiến hành GetBit hoặc SetBit như sau:
  + Với GetBit, ta thực hiện phép AND với 1 tại bit đó.
  + Với SetBit, nếu là bit 1, ta thực hiện phép OR với 1 tại bit đó (vì OR với 1 sẽ ra 1); nếu là bit 0, ta thực hiện phép AND với 0 tại bit đó (vì AND với 0 sẽ ra 0)

### Chuyển đổi từ chuỗi nhị phân (string) sang QInt:

Ta lần lượt duyệt qua từng phần tử của chuỗi string. Cứ mỗi lần duyệt ta gọi hàm SetBit để lưu vào QInt. Ở đây, ta duyệt từ đầu đến cuối của chuỗi string, nên khi SetBit vị trí sẽ là str.length() – 1 – i, với i là biến đếm duyệt chuỗi string, str.length() là độ dài chuỗi string (vì vị trí set bit là vị trị lưu trữ - sẽ được đánh số thứ tự ngược với chuỗi string như đã trình bày ở trên).

### Chuyển đổi từ chuỗi thập phân (string) sang QInt:

* Đầu tiên ta xét chuỗi thập phân biểu thị số âm hay dương, nếu có dấu ‘-’ ta thực hiện xóa dấu này ra khỏi chuỗi và có lưu lại sự thay đổi này bởi biến *sign* kiểu bool để làm cơ sở xét dấu lúc sau.
* Ta thực hiện chuyển chuỗi thập phân sang QInt bằng cách chia chuỗi này cho 2, phần dư sẽ được SetBit vào QInt, còn phần nguyên ta tiếp tục chia cho 2. Lặp lại như vậy cho đến khi chuỗi bằng “0” hoặc chuỗi rỗng.
* Ta dựa vào tình trạng biến *sign*, nếu biến *sign* biểu thị số âm ta thực hiện chuyển sang dạng bù 2 (ta thực hiện đổi bit kết quả tìm được ở trên, sau đó cộng cho 1).

### Chuyển đổi từ chuỗi thập lục phân (string) sang QInt:

* Do ở hệ thập lục phân, biểu diễn bởi 16 ký tự là 0 đến 9 và A đến F. Nên trước tiên ta thực hiện chuẩn hóa các ký tự đầu vào thành kí tự in hoa (nếu đầu vào là chữ thường).
* Điểm đặc biệt ở hệ thập lục phân là mỗi ký tự biểu diễn của nó sẽ được biễu diễn bởi 4 bit nhị phân (Vì mỗi ký tự của nó chạy từ 0 đến 15). Vì vậy ta sẽ duyệt lần lượt từng ký tự ở hệ thập lục phân và chuyển nó về dạng 4 bit nhị phân.
* Lưu ý khi chuyển, nếu gặp ký tự chữ ta phải chuyển đó về dạng số tương ứng rồi sau đó mới chuyển về dạng 4 bit. Cụ thể A là 10, B là 11, C là 12, D là 13, E là 14 và F là 15.

### Chuyển đổi từ chuỗi ở hệ cơ số 2, 10, 16 (string) sang QInt:

Ta sử dụng câu lệnh *if* trên tham số đầu vào (đầu vào sẽ được truyền vào biểu thị hệ cơ số cần được chuyển đổi vào QInt). Sau đó dựa vào đầu vào ta lần lượt gọi các hàm đã trình bày ở [2.2.3](#_Chuyển_đổi_từ_5), [2.2.4](#_Chuyển_đổi_từ_6) và [2.2.5](#_Chuyển_đổi_từ_7) để lưu trữ vào QInt.

### Chuyển đổi từ QInt sang chuỗi nhị phân (string):

Ta thực hiện lấy từng bit trong QInt và cộng vào chuỗi kết quả. Sau đó ta loại bỏ các số 0 thừa ở đầu. Lưu ý giữ lại 1 bit 0 nếu 128 bit đều là 0.

### Chuyển đổi từ QInt sang chuỗi thập phân (string):

* Ta thực hiện lấy từng bit trong QInt theo đúng thứ tự lưu trữ (từ 0 đến 126, trừ bit dấu). Tại mỗi bit nếu giá trị bit là 1 ta thực hiện cộng với 2^i với i là số thứ tự của bit.
* Ta xét bit dấu, nếu bit dấu là 1, ta thực hiện phép trừ giữa 2^127 và kết quả tìm được ở trên, sau đó thêm ‘-’ vào trước chuỗi kết quả.

### Chuyển đổi từ QInt sang chuỗi thập lục phân (string):

* Đầu tiên, ta tạo kiểu dữ liệu *map* để lưu trữ các ký tự biểu diễn hệ thập lục phân ở dạng 4 bit. Sở dĩ chọn kiểu dữ liệu *map* vì nó dễ dàng truy xuất phần tử với index dạng chuỗi.
* Sau đó ta duyệt lần lượt bộ 4 bit trong QInt từ vị trí i = 0 và dựa vào chuỗi 4 bit đó ta cộng vào chuỗi kết quả ký tự ở hệ thập lục phân tương ứng thông qua kiểu dữ liệu *map*.
* Sau đó ta thực hiện xóa các số 0 thừa (nếu có) và thực hiện đảo ngược lại kết quả (vì i = 0 nên ta phải đảo ngược kết quả).

### Chuyển đổi từ QInt sang chuỗi ở hệ cơ số 2, 10, 16 (string):

Ta sử dụng câu lệnh *if* trên tham số đầu vào (đầu vào sẽ được truyền vào biểu thị hệ cơ số cần được chuyển đổi ra string từ QInt). Sau đó dựa vào đầu vào ta lần lượt gọi các hàm đã trình bày ở [2.2.7](#_Chuyển_đổi_từ_2), [2.2.8](#_Chuyển_đổi_từ_3) và [2.2.9](#_Chuyển_đổi_từ_4) để chuyển từ QInt sang string.

### Chuyển đổi từ chuỗi ở hệ cơ số a sang chuỗi ở hệ cơ số b (a, b = 2, 10, 16):

Ta sử dụng câu lệnh *if* trên tham số đầu vào (đầu vào sẽ được truyền vào biểu thị hệ cơ số a – hệ cơ số cần phải chuyển đổi và hệ cơ số b – sau khi chuyển đổi). Sau đó dựa vào đầu vào ta lần lượt gọi các hàm đã trình bày ở [2.2.6](#_Chuyển_đổi_từ) và [2.2.10](#_Chuyển_đổi_từ_1) để thực hiện việc chuyển đổi.

## Các nhóm toán tử:

### Toán tử cộng:

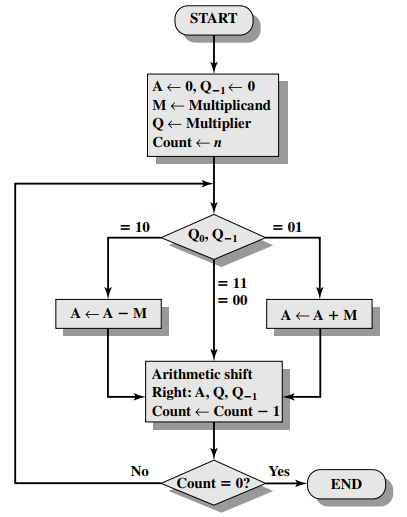
Phép cộng 2 số QInt được thực hiện dựa trên nguyên tắc cộng bit. Ta cộng lần lượt các bit theo thứ tự từ phải sang trái. Trong quá trình cộng có biến *carry* để lưu lại bit nhớ. Ta cộng bit ở số QInt trên với bit tương ứng ở số QInt dưới với bit nhớ ta được kết quả lưu ở biến *tmp*, sau đó ta thực hiện SetBit bằng cách lấy kết quả vừa tìm lấy dư cho 2, đồng thời cập nhật biến nhớ bằng cách lấy *tmp* chia cho 2.

### Toán tử trừ:

Ta thực hiện chuyển số trừ sang dạng bù 2 (đổi bit và cộng 1), sau đó thực hiện phép cộng giữa số bị trừ và số trừ ở dạng bù 2.

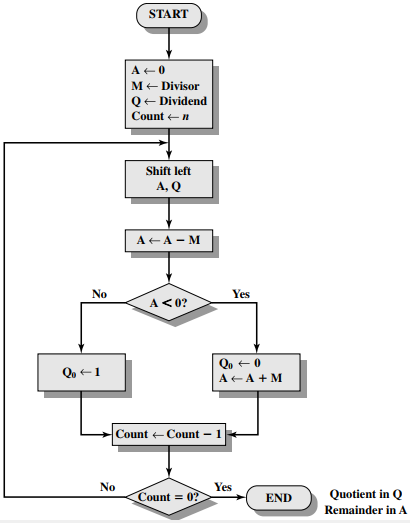
### Toán tử nhân:

Ta thực hiện theo thuật toán Booth. Lưu đồ thuật toán Booth sau đây được trích từ Chương 9, sách Computer Organization and Architecture Designing for performance (8th edition) của William Stallings.



### Toán tử chia và lấy dư:

* Phép chia và lấy dư được thực hiện theo thuật toán chia không dấu. Nếu số bị chia hay số chia là số âm, ta thực hiện chuyển các số âm đó sang dạng bù 2 để thực hiện chia không dấu. Sau khi chia không dấu ta dựa vào dấu của số bị chia và số chia để xác định dấu của thương và số dư cho phù hợp.
* Ta lưu ý về thương và số dư. Gọi D là số bị chia, V là số chia, Q là thương, R là số dư. Ta có biểu thức sau: D = Q × V + R. Dấu của R chính là dấu của D. Dấu của Q chính là dấu của D nhân với dấu của V. Xét ví dụ sau để hiểu rõ:
* D = 7, V = 3 => Q = 2, R = 1
* D = -7, V = 3 => Q = -2, R = -1
* D = 7, V = -3 => Q = -2, R = 1
* D = -7, V = -3 => Q = 2, R = -1
* Lưu đồ thuật toán chia không dấu sau đây được trích từ Chương 9, sách Computer Organization and Architecture Designing for performance (8th edition) của William Stallings.



### Toán tử so sánh và gán:

* Phép >: Kiểm tra xem 2 số có cùng dấu hay không, nếu khác dấu, ta kiểm tra số đầu tiên nếu là âm sẽ return false, ngược lại return true. Nếu cùng dấu, ta lần lượt so sánh các bit theo thứ tự từ bit kế bit dấu đến bit thứ 0 trong vị trí lưu trữ. Nếu bit tại vị trí *i* ở số thứ nhất nhỏ hơn bit ở số thứ 2, ta trả về bit dấu của số thứ nhất. Nếu bit bit tại vị trí *i* ở số thứ 1 lớn hơn bit ở số thứ 2, ta đổi bit dấu của số thứ nhất (lấy NOT) và trả về.
* Phép ==: Nếu hai số có bit nào khác nhau thì trả về false, ngược lại trả về true.
* Phép !=: Nếu hai số không bằng nhau thì return true, ngược lại return false.
* Phép <: Nếu số thứ nhất không lớn hơn và không bằng số thứ 2 return true; ngược lại return false
* Phép >=: Nếu số thứ nhất không nhỏ hơn số thứ 2 thì return true, ngược lại return false.
* Phép <=: Nếu số thứ nhất không lớn hơn số thứ 2 thì return true, ngược lại return false.
* Toán tử gán =: Nếu 2 địa chỉ của 2 biến khác nhau ta thực hiện gán thuộc tính của biến này sang biến khác.

### Toán tử AND, OR, XOR, NOT:

* Ở các toán tử AND, OR, XOR, ta thực hiện trên từng cặp bit của 2 số QInt.
* Ở toán tử NOT, ta đổi từng bit của số đó.
* Cụ thể:
* Ở phép AND, nếu cả 2 bit của 2 số QInt là 1 ta SetBit 1 vào số QInt kết quả, ngược lại SetBit 0.
* Ở phép OR, nếu cả 2 bit của 2 số QInt là 0 ta SetBit 0 vào số QInt kết quả, ngược lại SetBit 1.
* Ở phép XOR, nếu 2 bit của 2 số QInt bằng nhau, ta SetBit 0 vào số QInt kết quat, ngược lại SetBit 1.
* Ở phép NOT, ta thực hiện đổi từng bit (từ 0 sang 1 và từ 1 sang 0).

### Toán tử dịch trái, dịch phải:

* Ta truyền vào tham số là số lượng bit cần dịch.
* Ở đây ta thực hiện phép dịch số học. Khi dịch trái ta thêm các bit 0 vào cuối dãy bit sau khi dịch. Khi dịch phải ta thêm các bit dấu vào đầu dãy bit sau khi dịch.

### Toán tử xoay trái, xoay phải:

* Ta truyền vào tham số là số lượng bit cần xoay.
* Khi xoay trái / xoay phải *x* bit, ta lấy số lượng bit cần xoay bằng cách chia lấy dư với 128 (vì phép xoay sẽ thực hiện tuần hoàn với chu kỳ 128 bit).
* Mỗi lần xoay ta dịch trái / dịch phải 1 bit, lặp lại cho đến khi đủ *x* lần.

# BIỂU DIỄN SỐ CHẤM ĐỘNG CHÍNH XÁC CAO 128 BIT

## Thiết kế kiểu dữ liệu QFLoat:

* Số chấm động chính xác cao QFloat bao gồm 128 bit được lưu trữ bởi mảng data gồm 4 phần tử kiểu số nguyên. Ở phương thức khởi tạo mặc định, tất cả 128 bit này đều là bit 0.
* Vì 1 phần tử kiểu số nguyên có kích thước 4 bytes nên ta sử dụng 4 phần tử để biểu diễn (thỏa yêu cầu 16 bytes như đề bài).
* Trong 128 bit, có 1 bit biểu diễn dấu, 15 bit biểu diễn phần mũ và 112 bit biểu diễn phần định trị.

## Chuyển đổi giữa các hệ cơ số:

### Ý tưởng chuyển đổi:

Đầu vào sẽ là một chuỗi String ở dạng nhị phân / thập phân. Sau đó ta lưu chuỗi String này vào QFloat bằng các hàm chuyển đổi. Từ QFloat ta chuyển sang lại dạng chuỗi ở các dạng nhị phân / thập phân. Như vậy, ta có thể chuyển từ chuỗi String từ hệ cơ số a sang chuỗi String ở hệ cơ số b dựa vào trung gian là QFloat.

### Nhóm hàm SetBit, GetBit:

Ý tưởng và phương pháp thực hiện giống ở mục [2.2.2](#_Nhóm_hàm_SetBit,).

### Nhóm hàm kiểm tra các trường hợp đặc biệt:

* bool CheckZero() const;: Hàm kiểm tra có phải QFloat là 0 hay không. Nếu có bất kỳ bit nào trong 127 bit (trừ bit dấu) khác bit 0, return false.
* bool CheckDenormalized() const;: Hàm kiểm tra dạng số không chuẩn. Số không chuẩn là số có phần mũ toàn bit 0, và phần trị khác 0.
* bool CheckNaN() const;: Hàm kiểm tra có phải là dạng NaN không. Dạng NaN khi phần mũ toàn bit 1, và phần trị khác 0.
* bool CheckInf() const;: Hàm kiểm tra có phải là dạng Infinity không. Dạng Infinity khi phần mũ toàn bit 1, và phần trị toàn bit 0.

### Chuyển đổi từ chuỗi nhị phân (string) sang QFloat:

Ta lần lượt duyệt qua từng phần tử của chuỗi string. Cứ mỗi lần duyệt ta gọi hàm SetBit để lưu vào QFloat. Ở đây, ta duyệt từ đầu đến cuối của chuỗi string, nên khi SetBit vị trí sẽ là str.length() – 1 – i, với i là biến đếm duyệt chuỗi string, str.length() là độ dài chuỗi string (vì vị trí set bit là vị trị lưu trữ - sẽ được đánh số thứ tự ngược với chuỗi string như đã trình bày ở trên).

### Chuyển đổi từ chuỗi thập phân (string) sang QFloat:

* Đầu tiên, ta xét dạng chuỗi thập phân nhập vào có phải dạng “0.0” hoặc “0” hay không, nếu có ta trả về phương thức khởi tạo mặc định của QFloat.
* Ta xét chuỗi thập phân biểu thị số âm hay dương, nếu có dấu ‘-’ ta thực hiện xóa dấu này ra khỏi chuỗi và ta thực hiện SetBit dấu vào kết quả. Lưu ý bit dấu là vị trí thứ 127 như đã trình bày ở những phần trên.
* Ta thực hiện phân tách phần nguyên và phần thập phân của chuỗi đầu vào nhờ việc tìm kiếm dấu “.” trong chuỗi đầu vào. Tuy nhiên, một số trường hợp chuỗi đầu vào không có dấu “.”, ta thực hiện thêm vào sau chuỗi thập phân đầu vào “.0”.
* Ta thực hiện đổi phần nguyên và phần thập phân của chuỗi đầu vào thành dãy bit nhị phân. Lưu ý, khi đổi phần thập phân thành các dãy bit, ta có các trường hợp sau đây:
* Nếu phần nguyên khác 0, nghĩa là số này có dạng 1.F, đây là dạng chuẩn, nên ta thực hiện lấy phần thập phân nhân cho 2, và lưu lại phần nguyên. Sau đó lặplại việc này cho đến khi đủ bit. Trong quá trình lặp lại, khi đạt được giá trị 1.0 ta thực hiện lưu lại tình trạng này để hỗ trợ việc làm tròn sau này. Đồng thời, giá trị phần mũ được tính như sau: *exp = intPartBit.size() - 1 + BIAS* trong đó *exp* là giá trị phần mũ, *intPartBit.size()* là chiều dài dãy bit nhị phân của phần nguyên, *BIAS* trong trường hợp QFloat là 2^14 – 1 = 16383. Sở dĩ có cách tính này là vì phần nguyên khác 0 nên tối thiểu là length intPartBit = 1 nên số lần dời dấu chấm là *intPartBit.size() – 1.*
* Nếu phần nguyên bằng 0, ta có 2 trường hợp: số chuẩn hóa được và số không thể chuẩn hóa. Ta cũng thực hiện lấy phần thập phân nhân cho 2, và lưu lại phần nguyên. Sau đó lặp lại việc này cho đến khi đạt được giá trị 1.0. *exp* được tính bằng *BIAS* trừ đi số lần lặp. Nếu số lần lặp để tìm được giá trị 1.0 nhỏ hơn *BIAS* thì đây là trường hợp số có thể chuẩn hóa được, ta tiếp tục thực hiện nhân phần thập phân cho 2 và lưu lại phần nguyên cho đến khi đủ bit. Ngược lại là dạng số không thể chuẩn hóa, ta thực hiện reset lại chuỗi nhị phân của phần thập phân và nhân phần thập phân cho 2 và lưu lại phần nguyên và tăng *exp* lên 1. Lặp lại cho đến khi *exp = 0.* Trong quá trình lặp lại, khi đạt được giá trị 1.0 ta thực hiện lưu lại tình trạng này để hỗ trợ việc làm tròn sau này.
* Nếu ta chưa tìm được giá trị 1.0 khi lặp lại việc nhân phần thập phân cho 2. Ta tiếp tục lặp lại việc này thêm 3 lần, để thực hiện việc làm tròn theo nguyên tắc *round to the nearest; ties to even*.
* Ta làm tròn phần nguyên: khi thập phân toàn bit 1, ta cộng 1 vào *exp* và set lại phần thập phân là 0. Ví dụ: 10.1111 -> làm tròn: 11.0000.
* Nếu tổng bit nhị phân phần nguyên và phần thập phân chưa đủ 112 bit ta tiến hành thêm các bit 0 cho đủ bit.
* Sau cùng, ta đổi *exp* sang dãy bit nhị phân và tiến hành SetBit cho *exp* và phần trị vào kết quả.

### Chuyển đổi từ chuỗi ở hệ cơ số 2, 10 (string) sang QFloat:

Ta sử dụng câu lệnh *if* trên tham số đầu vào (đầu vào sẽ được truyền vào biểu thị hệ cơ số cần được chuyển đổi vào QFloat). Sau đó dựa vào đầu vào ta lần lượt gọi các hàm đã trình bày ở [3.2.4](#_Chuyển_đổi_từ_8) và [3.2.5](#_Chuyển_đổi_từ_9) để lưu trữ vào QFloat.

### Chuyển đổi từ QFloat sang chuỗi nhị phân (string):

Ta thực hiện lấy từng bit trong QFloat và cộng vào chuỗi kết quả. Lưu ý thứ tự khi GetBit và cộng vào chuỗi. Ta thực hiện GetBit từ bit dấu (bit thứ 127) về bit thứ 0.

### Chuyển đổi từ QFloat sang chuỗi thập phân (string):

* Ta kiểm tra QFloat có phải là 0 hay dạng NaN hay không, nếu phải ta thực hiện return chuỗi báo tương ứng.
* Ta lưu lại bit dấu của QFloat (bit thứ 127).
* Ta kiểm tra QFloat có phải dạng Infinity hay không, nếu phải ta xét bit dấu và return Inf hay -Inf cho phù hợp.
* Nếu QFloat không nằm trong các dạng đặc biệt trên, ta thực hiện:
* Tách phần mũ và phần trị từ QFloat (ta chuyển QFloat sang chuỗi nhị phân cho dễ thao tác).
* Ta tính giá trị *exp* bằng cách lấy giá trị của dãy bit – *BIAS*. Với số không chuẩn, phần mũ bằng 0 và *exp = -16382* và có dạng 0.F*.* Với số bình thường, *exp* được tính theo công thức trên và có dạng 1.F.
* Dời dấu chấm động khi *exp* != 0. *exp* dương thì ta dời qua phải. Mỗi lần dịch, phần nguyên cộng thêm bit từ phần thập phân chuyển qua và phần thập phân xóa bớt. Nếu đã dịch hết phần thập phân, ta thêm 0 vào phần nguyên. Sau đó, giảm *exp* đi 1 đơn vị. Ngược lại ta dời qua trái. Mỗi lần dịch, phần thập phân cộng thêm bit từ phần nguyên chuyển qua và phần nguyên xóa bớt bit. Nếu đã dịch hết phần nguyên, ta thêm 0 vào phần thập phân. Sau đó, tăng *exp* lên 1 đơn vị.
* Chuyển phần nguyên về dạng thập phân (hệ 10)
* Chuyển phần thập phân về dạng thập phân (hệ 10). Khi duyệt bit ở phần này, tại mỗi bit, nếu là bit 1 ta thực hiện cộng với 2^-n, với n là vị trí bit, tính từ 1.

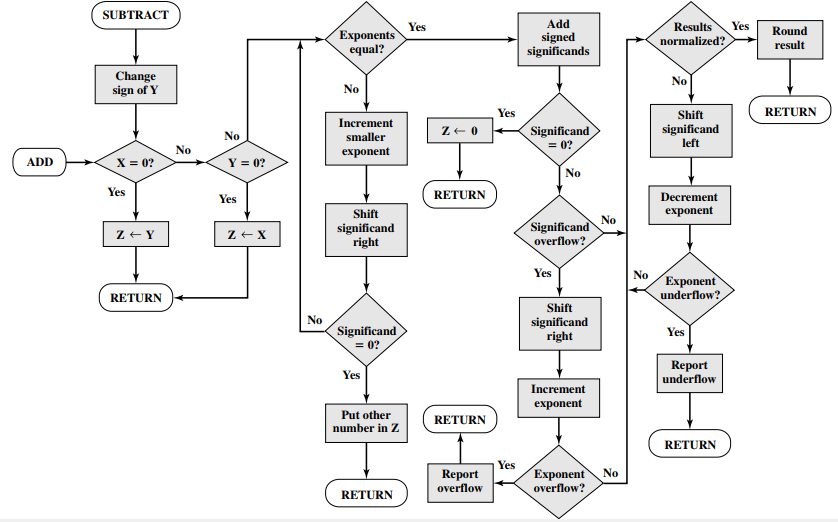
### Chuyển đổi từ QFloat sang chuỗi ở hệ cơ số 2, 10 (string):

Ta sử dụng câu lệnh *if* trên tham số đầu vào (đầu vào sẽ được truyền vào biểu thị hệ cơ số cần được chuyển đổi ra string từ QFloat). Sau đó dựa vào đầu vào ta lần lượt gọi các hàm đã trình bày ở [3.2.7](#_Chuyển_đổi_từ_10) và [3.2.8](#_Chuyển_đổi_từ_11) để chuyển từ QFloat sang string.

## Các nhóm toán tử:

### Toán tử cộng:

Ta thực hiện cộng theo lưu đồ thuật toán sau đây được trích từ Chương 9, sách Computer Organization and Architecture Designing for performance (8th edition) của William Stallings.



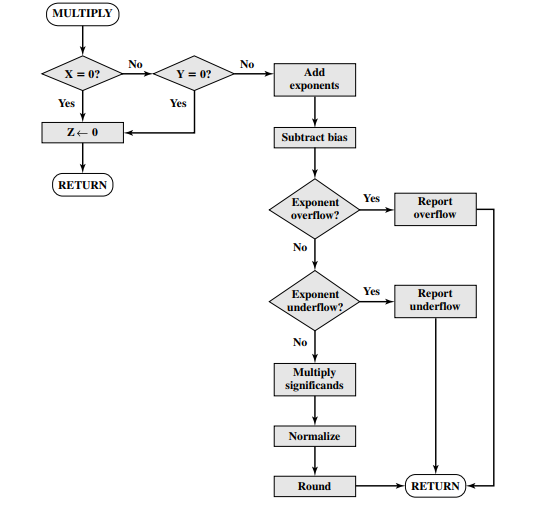
* Ta có thể tóm gọn sơ đồ trong 4 bước chính:
* Kiểm tra phép cộng với NaN, Infinity và 0.
* Điều chỉnh phần mũ cho 2 số bằng nhau.
* Thực hiện cộng có dấu phần định trị của 2 số.
* Chuẩn hóa kết quả (normalize result).

### Toán tử trừ:

Ta thực hiện đổi số trừ sang số đối của nó (thực hiện đối dấu) và thực hiện phép cộng (vì a – b = a + (-b)).

### Toán tử nhân:

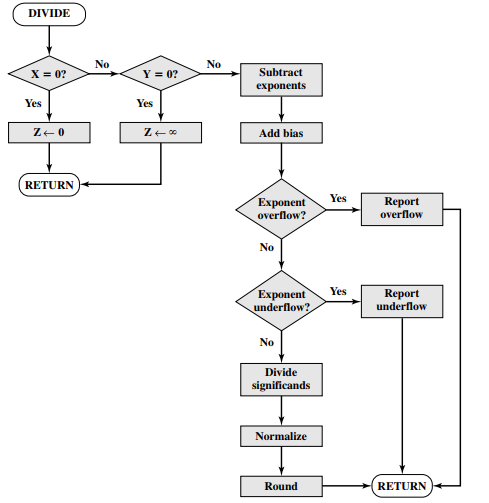
Ta thực hiện nhân theo lưu đồ thuật toán sau đây được trích từ Chương 9, sách Computer Organization and Architecture Designing for performance (8th edition) của William Stallings.



* Ta có thể tóm gọn sơ đồ trong 4 bước chính:
* Kiểm tra phép nhân với NaN, Infinity và 0.
* Tính toán phần mũ
* Thực hiện nhân phần định trị
* Chuẩn hóa kết quả (normalize result).

### Toán tử chia:

Ta thực hiện chia theo lưu đồ thuật toán sau đây được trích từ Chương 9, sách Computer Organization and Architecture Designing for performance (8th edition) của William Stallings.



* Ta có thể tóm gọn sơ đồ trong 4 bước chính:
* Kiểm tra phép chia với NaN, Infinity và 0.
* Tính toán phần mũ
* Thực hiện chia phần định trị
* Chuẩn hóa kết quả (normalize result).

# PHẠM VI BIỂU DIỄN CỦA CÁC KIỂU DỮ LIỆU ĐÃ THIẾT KẾ

## Kiểu số nguyên lớn QInt 128 bit:

Do lưu trữ QInt trên 128 bit, nên phạm vi biểu diễn của QInt là từ -2127 đến 2127 – 1.

## Kiểu số chấm động chính xác cao:

Kiểu QFloat được lưu trữ bởi 1 bit dấu (s), 15 bit phần mũ (E), 112 bit phần định trị (F). Ta xét:

* Số chuẩn:
* Min (+): 1.[112 bit 0] × 2-16382 = 2-16382
* Max (+): 1.[112 bit 1] × 216383 = (20 + 2-1 + 2-2 +…+ 2-112) × 216383

= (2 - 2-112) × 216383

= 216384 - 216271

* Min (-): -1.[112 bit 1] × 216383 = -(216384 - 216271)
* Max (-): -1.[112 bit 0] × 2-16382 = -2-16382
* Số không chuẩn:
* Min (+): 0.[111 bit 0]1 × 2-16382 = 2-112 × 2-16382 = 2-16494
* Max (+): 0.[112 bit 1] × 2-16382 = (2-1 + 2-2 +…+ 2-112) × 2-16382

= (1 - 2-112) × 2-16382

= 2-16382 - 2-16494

* Min (-): -0.[112 bit 1] × 2-16382 = -(2-16382 - 2-16494)
* Max (-): -0.[111 bit 0]1 × 2-16382 = -2-16494
* Vậy phạm vi biểu diễn của QFloat là:

[ -(216384 - 216271); -2-16382 ] [ -(2-16382 - 2-16494); -2-16494 ] { 0 } [ 2-16494; 2-16382 - 2-16494] [ 2-16382; 216384 - 216271 ]

* Ngoài ra QFloat còn có dạng Infinity / -Infinity (phần mũ toàn bit 1, phần trị bằng 0)
* Nếu vượt ra ngoài các khoảng trên, thì QFloat sẽ biểu diễn dạng số báo lỗi NaN (phần mũ toàn bit 1, phần trị khác 0).

# GIAO DIỆN CHƯƠNG TRÌNH ỨNG VỚI TEST CASE

## Tham số dòng lệnh:

## Giao diện người dùng:

# ĐÁNH GIÁ MỨC ĐỘ HOÀN THÀNH

|  |  |  |
| --- | --- | --- |
| **Kiểu dữ liệu / Công việc chính** | **Công việc** | **Mức độ hoàn thành** |
| **QInt** | Nhập, xuất | 100% |
| Chuyển đổi giữa hệ cơ số a sang hệ cơ số b (với a, b thuộc {2, 10, 16}) | 100% |
| Toán tử cộng, trừ, nhân, chia, **chia lấy dư** | 100% |
| Các toán tử so sánh và gán (<, >, <=, >=, ==, **!=**, =) | 100% |
| Các toán tử AND, OR, XOR, NOT | 100% |
| Toán tử dịch trái, dịch phải, xoay trái, xoay phải | 100% |
| **QFloat** | Nhập, xuất | 100% |
| Chuyển đổi giữa hệ cơ số a sang hệ cơ số b (với a, b thuộc {2, 10}) | 100% |
| Toán tử cộng, trừ, nhân, chia | 100% |
| **Giao diện** | Thiết kế giao diện Calculator ở 2 chế độ QInt và QFloat | 100% |
| **Tổng thể đồ án** |  | **100%** |

# TÀI LIỆU THAM KHẢO

[1] Slides bài giảng Kiến trúc máy tính và hợp ngữ, Thầy ThS Phạm Tuấn Sơn, trường Đại học Khoa học Tự nhiên.

[2] Quadruple-precision floating-point format: <https://en.wikipedia.org/wiki/Quadruple-precision_floating-point_format>

[3] Floating-Point Arithmetic, A. Yavuz Oruç Professor, UMD, College Park: <https://user.eng.umd.edu/~yavuz/teaching/courses/enee446/lecture%20notes/lecture4467N8N9.ppt.pdf>

[4] Computer Organization and Architecture Designing for performance (8th edition), William Stallings.