**KHOA CÔNG NGHỆ THÔNG TIN**

**ĐẠI HỌC KHOA HỌC TỰ NHIÊN THÀNH PHỐ HỒ CHÍ MINH**

Thực hiện: Nhóm 15

20C12007 – Trần Đình Lâm

20C11035 – Trương Thế Kiệt

20C11040 – Đặng Nhật Minh

ĐỒ ÁN MÔN HỌC PHƯƠNG PHÁP TOÁN TRONG TIN HỌC

NĂM HỌC 2020-2021

**ĐỒ ÁN 4: NGHIÊN CỨU**

**CẤU TRÚC SỐ FLOAT/DOUBLE**

**ĐẠI HỌC QUỐC GIA TPHCM**

**BẢNG THÔNG TIN CHI TIẾT NHÓM**

|  |  |  |  |
| --- | --- | --- | --- |
| **Mã nhóm:** | **15** | | |
| **Tên nhóm:** | **K2014** | | |
| **Số lượng:** | **3** | | |
| **MSSV** | **Họ tên** | **Email** | **Điện thoại** |
| 20C12007 | Trần Đình Lâm | tdlam123@gmail.com | 0383522356 |
| 20C11035 | Trương Thế Kiệt | truongthekiet709@gmail.com |  |
| 20C11040 | Đặng Nhật Minh | minhdangnhat685@gmail.com |  |

|  |  |  |  |
| --- | --- | --- | --- |
| **BẢNG PHÂN CÔNG & ĐÁNH GIÁ HOÀN THÀNH CÔNG VIỆC** | | | |
| **Người thực hiện** | **Công việc thực hiện** | **Mức độ hoàn thành** | **Đánh giá của nhóm** |
| 20C12007  Trần Đình Lâm | Tìm hiểu nguyên nhân lỗ hổng nguyên | 70% | 7/10 |
| 20C11035  Trương Thế Kiệt | Tìm hiểu cấu trúc số floating point | 70% | 7/10 |
| 20C11040  Đặng Nhật Minh | Khảo sát lỗ hổng nguyên | 70% | 7/10 |

MỤC LỤC

[I. Cấu trúc 2](#_Toc67800182)

[1. Cấu trúc của số độ chính xác đơn 2](#_Toc67800183)

[2. Cấu trúc của số độ chính xác kép 3](#_Toc67800184)

[3. Số không nguyên trong dấu chấm động 4](#_Toc67800185)

[II. Lỗ hổng nguyên 6](#_Toc67800186)

[1. Tính liên tục của số nguyên 6](#_Toc67800187)

[2. Biểu diễn số nguyên bằng floating point 6](#_Toc67800188)

[3. Hiện tượng lỗ hổng nguyên 6](#_Toc67800189)

[III. Khảo sát lỗ hổng nguyên bằng thực nghiệm 9](#_Toc67800190)

[1. Cách thực hiện khảo sát sự thay đổi độ nhảy cóc qua các mốc số nguyên 9](#_Toc67800191)

[2. Kết quả và nhận xét 9](#_Toc67800192)

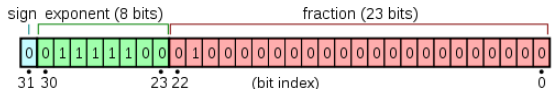
[3. Đồ thị thể hiện sự thay đổi độ nhảy cóc qua các mức số nguyên 12](#_Toc67800193)

[TÀI LIỆU THAM KHẢO 14](#_Toc67800194)

# Cấu trúc

* Để biểu diễn một số không nguyên chúng ta có nhiều cách để thể hiện như:
* Fixed-point
* Logarithmic number systems (LNSs)
* Interval arithmetic
* Đến năm 1985, tính toán dấu phẩy động được thành lập bởi Institute of Electrical and Electronics Engineers (IEEE), tiêu chuẩn này giải quyết được nhiều vấn đề bất cập trước đó.
* Trong tiêu chuẩn IEEE 754, các số được chia làm hai nhóm: đ**ộ chính xác đơn(single precision)** và **độ chính xác kép (double precision).**

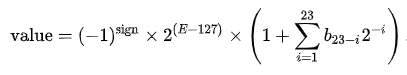
## Cấu trúc của số độ chính xác đơn



* Cấu trúc của số không nguyên với độ chính xác đơn được xác định bởi dãy 32 bit. Trong đó từ trái qua bit đầu tiên sẽ được thể hiện là bit dấu với giá trị 1 là số âm – 0 là số dương, kế tiếp sau đó là 8 bit cho phần xác định số mũ, 23 bit tiếp theo được sử dụng cho phần định trị.
* Giá trị của phần mũ(exponent) để biểu diễn được phần mũ âm và mũ dương người ta qui định giá trị thực tế của phần mũ sẽ được tính với công thức như sau:

Giá trị số mũ thực tế = E - 127

* Giá trị thể hiện của một số không nguyên theo tiêu chuẩn IEE 754 với độ chính xác đơn từ một chuỗi 32 bit sẽ được tính như sau:



Trong đó:

sign: giá trị của phần mũ (1 hoặc 0)

E: giá trị phần mũ

b23-i: là giá trị ở các vị trí trong phần định trị

* Ví dụ:

Cho dãy 32 bit: 1 1000 0001 011 0000 0000 0000 0000 0000

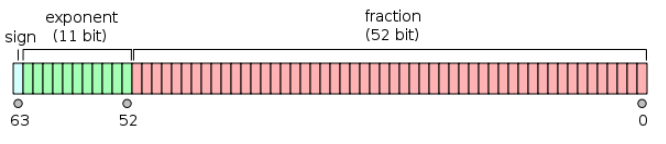
* + - Sign: 1
    - Exponent: 10000001
    - Fraction(mantissa): 011 0000 0000 0000 0000 0000

value = (-1)1 x 2(129-127) x (1+ 1\*2-2 + 1\*2-3)

= (-1) x 4 x (1.375) = 5.5D

* Dựa vào cấu trúc ở trên, ta thấy với độ chính xác đơn ta có 23 bit để thể hiện giá trị của một sô nguyên tức số lượng phần tử nguyên có thể chứa được là **223– 1**  số. nhưng theo cup cấp từ trang của Microsoft thì khoảng giá trị lớn nhất mà số float thể hiện được là 3.402823466.E+38 = 3402823466. E+29 lớn hơn rất nhiều so với số lượng số nguyên có thể chưa được. Từ đó có một lỗi mà mỗi kiểu thể hiện theo dấu chấm động điều mắc phải là tính tràn nhảy cóc giữa các số liên típ nhau. Tức giá trị nguyên sẽ không được lưu tất cả với các số liên tiếp nhau chênh nhau 1 đơn vị. Mà nó sẽ có thể chênh nhau vài đơn vị đến vài trăm vài nghìn đơn vị. Chinh tiết sẽ được thể hiện rõ ở những phần sau.
* Dựa vào phần tích trên thì giá trị các số từ 223 – 1 sẽ được tính toán và thể hiện đúng giá trị của chuỗi số.

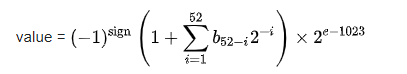
## Cấu trúc của số độ chính xác kép



* Tương tự với độ chính xác đơn, ở độ chính xác kép chúng ta có số lượng tăng lên gấp đôi với tổng số bit trong chuỗi là 64.
* Trong đó bit đầu tiên từ trái sẽ là bit dấu với 1 là số âm, 0 là số dương, 11 bít liền kề kế tiếp sẽ là số bít thể hiện giá trị phần mũ, 53 bit liền sau đó số bit cho phần định trị.
* Giá trị cho phần mũ thật sự được tính theo công thức:

Giá trị số mũ thực tế = E - 1023

* Giá trị thể hiện của một số không nguyên theo tiêu chuẩn IEE 754 với độ chính xác kép từ một chuỗi 64 bit sẽ được tính như sau:



Trong đó:

sign: giá trị của phần mũ (1 hoặc 0)

E: giá trị phần mũ

b23-i: là giá trị ở các vị trí trong phần định trị

* Ví dụ:

Cho dãy 64 bit: 0 10000000001 0100000000000000000000000000000000000000

000000000000

* + - Sign: 0
    - Exponent: 10000000001
    - Fraction: 0100000000000000000000000000000000000000000000000000

value = (-1)0 x 2(1025-1023) x (1+ 1\*2-2)

= (1) x 4 x (1.25) = 5D

* Cũng tương tự như số với độ chính xác đơn, với cách biểu diễn với độ chính xác kép giá trị trong miền an toàn để có thể thể hiện được giá trị chính xác với cận trên là

**253– 1.**

## Số không nguyên trong dấu chấm động

Các bước sau thể hiện cách chuyển đổi số không nguyên từ một số hệ thập phân sang kiểu thể hiện dấu chấm động với độ chính xác đơn

* Giả định số không nguyên với định dạng : xxx.yyy
* Với xxx là phần nguyên, yyy là phần thập phân
* **Bước 1**: Xác định phần nguyên và phần thập phân
* **Bước 2**: Chuyển đổi phần nguyên và phần thập phân sang chuỗi nhị phân
* **Bước 3**: Nối hai phần nguyên và phần nhị phân đã chuyển đổi qua chuỗi nhị phân lại với nhau với ngăn cách bằng dấu chấm.
* **Bước 4** : Xác định bit dấu (âm là 1,dương là 0)
* **Bước 5** : chuyển chuổi nhị phân ở bước 3 về định dạng **1.xxxx \* 2q**
* **Bước 6** : Xác định phần số trị exponent.
* **Bước 7** : Chuyển đổi exponent sang chuỗi nhị phân.
* **Bước 8** : đặt 3 thành phần sign, exponent, fraction vào đúng vị trí.

Ví dụ:

Cho số ở hệ thập phần là 85.125

**Bước 1:**

Phần nguyên : 85

Phần thập phân : 0.125

**Bước 2:**

Phần nguyên: 1010101

Phần thập phân: 001

**Bước 3:** 1010101.001

**Bước 4:** Do là số dương nên sign : 0

**Bước 5:** 1.010101 \* 26

**Bước 6:** exponent = 127 + 6 (vì 6 = expoment - 127)

**Bước 7:** exponent = 10000101

**Bước 8:** sign = 0, exponent = 10000101, fraction= 010101

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 1 | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

Biểu diễn nhị phần theo độ chính xác đơn của 85.125

# Lỗ hổng nguyên

## Tính liên tục của số nguyên

Trong toán học, tập số nguyên Z bao gồm các số nguyên dương (1, 2, 3,…), các số nguyên âm (−1, −2, −3,...) và số 0. Số nguyên có tính liên tục và khoảng cách giữa các số nguyên liên tiếp luôn luôn là 1 đơn vị.

## Biểu diễn số nguyên bằng floating point

Như đã trình bày ở phần trước, nếu dựa theo chuẩn IEEE-754 [1], nhờ phần định trị fraction thì ta có số thể biểu diễn chính xác được số nguyên tối đa trong phạm vi bằng floating point 32 bit có độ chính xác đơn, và bằng floating point 64 bit có độ chính xác kép. Trong các phạm vị số nguyên tương ứng này, floating point vẫn có khả năng biểu diễn và đảm bảo tính liên tục của số nguyên. Tuy nhiên khi vượt ngoài 2 ngưỡng nêu trên thì tính liên tục của số nguyên không còn được đảm bảo, mà xuất hiện hiện tượng "nhảy cóc" bước số ở một vài giá trị cụ thể. Bước nhảy cóc này được gọi là lỗ hổng nguyên.

## Hiện tượng lỗ hổng nguyên

Ở đây để đơn giản hóa vấn đề, ta chỉ khảo sát trên floating point 32 bit, kết quả tương ứng có thể suy ra cho floating point 64 bit.

Ví dụ cụ thể biểu diễn số nguyên , ta có:

Suy ra floating point 32 bit có được bao gồm:

* Sign (1 bit): 0
* Exponent (8 bit):
* Fraction (23 bit):

Đến đây phần định trị Fraction đã đạt ngưỡng giá trị tối đa mà 23 bit có thể biểu diễn. Điều này dẫn đến, vấn đề về độ chính xác phát sinh khi số lượng bit cần biểu diễn sau dấu chấm vượt quá số lượng bit fraction mà floating point có thể lưu. Ở ví dụ vừa trình bày, ta khảo sát thêm 2 giá trị tiếp theo, đối sánh với kết quả tham khảo [2] để có thể thấy được hiện tượng này như sau:

|  |
| --- |
| (24 bit 0)  Suy ra floating point 32 bit có được bao gồm:   * Sign (1 bit): 0 * Exponent (8 bit): * Fraction (23 bit): (23 bit 0, thiếu 1 bit 0)   Vẫn chưa bị mất mát giá trị, vì phần bit mất đi chỉ là 0 |
| (23 bit 0, 1 bit 1)  Suy ra floating point 32 bit có được bao gồm:   * Sign (1 bit): 0 * Exponent (8 bit): * Fraction (23 bit): (23 bit 0, thiếu 1 bit 1)   Bị mất 1 đơn vị, vì phần bit mất đi mang giá trị 1 |

Theo đó, mỗi khi bit cuối có giá trị 1, thì tổng giá trị của số nguyên bị mất đi 1 đơn vị.

Tổng quát hóa ở , có 1 bit cuối bị mất, ta có các giá trị bị nhảy cóc lần lượt là ;;;...;.

Ở , 2 bit cuối bị mất, các giá trị bị nhảy cóc : ;;;...;.

Ở , bit cuối bị mất, các giá trị bị nhảy cóc:

|  |
| --- |
|  |

Từ ta ánh xạ sang cấu trúc floating point 64 bit, với fraction tương ứng là thay vì 23. Ta thấy các khoảng cách các giá trị được biểu thị chính xác có biên độ nhảy cóc (gap size) tăng đáng kể so với biên độ thực (luôn là 1) khi đủ lớn, cụ thể các giá trị chính xác [3] là:

|  |
| --- |
|  |

Theo ta rút ra nhận xét: là biên độ nhảy cóc giữa các giá trị chính xác có thể lưu trữ. Khi (số mũ) tăng 1, thìtăng gấp đôi. Tính chất này sẽ được khảo sát thực nghiệm trong một ngôn ngữ lập trình cụ thể ở phần sau.

# Khảo sát lỗ hổng nguyên bằng thực nghiệm

## Cách thực hiện khảo sát sự thay đổi độ nhảy cóc qua các mốc số nguyên

Xét số float, với việc phân tích cấu trúc ở phần trên và cơ chế lưu trữ khi số bit cần để biểu diễn số nguyên vượt quá số bit của phần định trị (23 + 1), chúng ta dự đoán được các mốc thay đổi nhảy cóc sẽ là 2^n với n >= 24. Ta thực hiện việc khảo sát với số float bằng ngôn ngữ Java như sau:

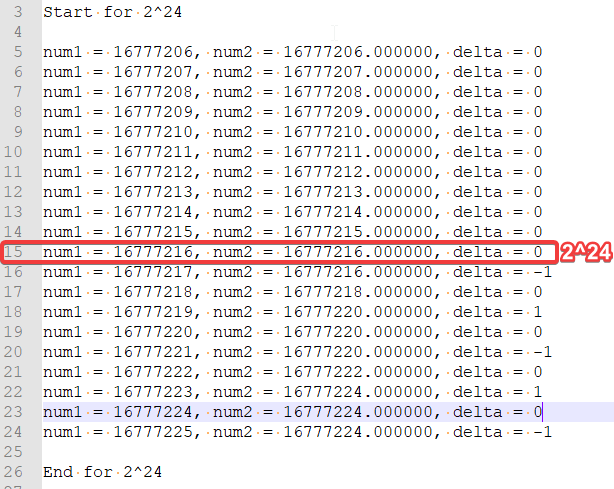
* Cho các số nguyên trong khoảng [2^n – 10, 2^n + 10] (với 24 <= n <= 30)
* Ta biểu diễn các số nguyên đó bằng 2 kiểu dữ liệu: kiểu int và kiểu float để từ đó nhận xét độ chênh lệch giữa 2 cách biểu diễn.

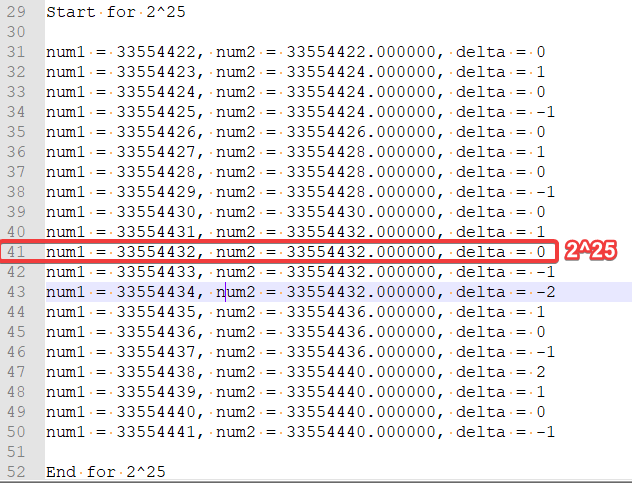
## Kết quả và nhận xét

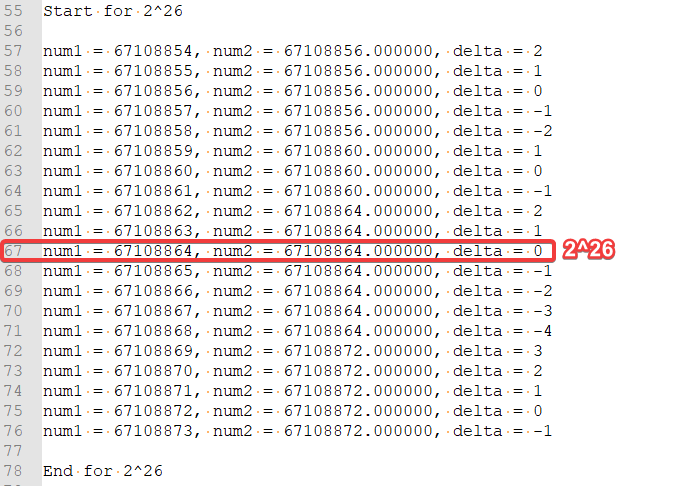
Code snippet và kết quả output được đính kèm trong bài nộp tương ứng ở 2 file: int\_to\_float\_java\_output.txt và int\_to\_float\_java\_code\_snippet.txt

**Chú thích**:

* num1 là số nguyên biểu diễn bằng kiểu int
* num2 là số nguyên biểu diễn bằng kiểu float
* delta = num2 – num1







**Từ bảng giá trị thực nghiệm, ta có nhận xét:**

* Với các số < 2^24, kiểu dữ liệu float có thể biểu diễn chính xác giá trị số nguyên.
* Từ 2^24 trở đi, giả sử đang khảo sát các số nguyên lân cận 2^n: nhảy cóc xuất hiện với giá trị tối đa là 2^(n-23) – 1, khoảng cách giữa các bước nhảy là 2^(n-23), nên các giá trị số nguyên i với [i mod 2^(n - 23) != 0] sẽ bị biểu diễn sai
* Mốc của sự thay đổi độ nhảy cóc là số nguyên 2^n, qua số nguyên này bước nhảy sẽ biến đổi từ 2^(n-1-23) sang 2^(n-23)
* Sự biến thiên độ lệch sẽ có một số bất thường không theo chu kỳ (như với 2^25 độ lệch sẽ biến đổi -1 -2 1 0 -1 2 1 0 -1 …) do cơ chế làm tròn cho số floating point của Java sẽ làm tròn số nguyên đầu vào tới số nguyên mà float có thể biểu diễn được gần nhất

## Đồ thị thể hiện sự thay đổi độ nhảy cóc qua các mức số nguyên

# TÀI LIỆU THAM KHẢO

|  |  |
| --- | --- |
| [1] | t. f. e. Wikipedia, "IEEE 754," Wikipedia, [Online]. Available: https://en.wikipedia.org/wiki/IEEE\_754. |
| [2] | "IEEE-754 Floating Point Converter," h-schmidt.net, [Online]. Available: https://www.h-schmidt.net/FloatConverter/IEEE754.html. |
| [3] | R. Regan, "The Spacing of Binary Floating-Point Numbers," Exploring Binary, 15 3 2015. [Online]. Available: https://www.exploringbinary.com/the-spacing-of-binary-floating-point-numbers/. |