

Khảo sát số chấm động của các ngôn ngữ lập trình trên nền kiến trúc x86

Mục đích

- Tìm hiểu số chấm động trong các ngôn ngữ lập trình trên nền kiến trúc x86
- Hiểu rõ hơn về cách tổ chức số chấm động

Tóm tắt lý thuyết

Hầu hết các ngôn ngữ lập trình trên nền kiến trúc x86 như Assembly, C/C++, Java, VB, C#,... đều sử dụng chuẩn số chấm động IEEE 754 để biểu diễn số chấm động. Trong đó, chuẩn số chấm động 32-bit (single – chính xác đơn) và chuẩn 64-bit (double – chính xác kép) được sử dụng phổ biến nhất. Ví dụ trong ngôn ngữ C, kiểu float sử dụng số chấm động 32-bit, kiểu double sử dụng số chấm động 64-bit.

Bài tập

Bài 1. Viết chương trình nhập vào số chấm động. Hãy xuất ra biểu diễn nhị phân từng thành phần (dấu, phần mũ, phần trị) của số chấm động vừa nhập

Ví dụ:

Nhập vào số chấm động (32-bit): 6
 Biểu diễn nhị phân tương ứng: 0 10000001 100000000000000000000000
 Nhập vào số chấm động (32-bit): -12.625
 Biểu diễn nhị phân tương ứng: 1 10000010 100101000000000000000000
 Nhập vào số chấm động (32-bit): 0.1015625
 Biểu diễn nhị phân tương ứng: 0 01111011 101000000000000000000000
 Nhập vào số chấm động (32-bit): 0.1
 Biểu diễn nhị phân tương ứng: 0 01111011 10011001100110011001101
 Nhập vào số chấm động (32-bit): 0
 Biểu diễn nhị phân tương ứng: 0 00000000 000000000000000000000000

Hướng dẫn:

- Viết hàm **dumpFloat(float *p)** trên ngôn ngữ C++ cho phép xem các bit của một biến kiểu **float**
 - Ví dụ trong chương trình có khai báo biến **float x** thì khi gọi **dumpFloat(&x)** sẽ in ra màn hình biểu diễn nhị phân của giá trị đang lưu trong x, trong đó chỉ rõ phần nào là exponent, phần nào là significand
 - Lưu ý: nên dùng các phép toán trên bit để lấy nội dung các bit và in ra chứ không thực hiện việc chuyển đổi thủ công

Bài 2. Viết chương trình nhập vào biểu diễn nhị phân của số chấm động. Hãy xuất ra biểu diễn thập phân tương ứng

Ví dụ:

Dãy nhị phân: 0 10001000 011011000010000000000000
 Số chấm động (single) tương ứng: 728.25
 dãy nhị phân: 1 01000110 011010110000000000000000
 Số chấm động (single) tương ứng: $-9.83913471531 \times 10^{-18}$
 dãy nhị phân: 0 01111011 10011001100110011001101
 Số chấm động (single) tương ứng: 0.1
 dãy nhị phân: 0 11111111 000000000000000000000000
 Số chấm động (single) tương ứng: $+\infty$
 dãy nhị phân: 0 11111111 100000000000000000000000
 Số chấm động (single) tương ứng: NaN

Hướng dẫn:

- Viết hàm **forceFloat(float *p, char *s)** trên ngôn ngữ C++ cho phép ghi các bit vào một vùng nhớ kiểu **float**
 - o Ví dụ trong chương trình có khai báo biến **float x** thì khi gọi **forceFloat(&x, "10011")**, 5 bit cao nhất (LSB) của vùng bộ nhớ 32 bit chiếm bởi x sẽ bị ghi giá trị lần lượt là 1,0,0,1,1 còn các bit còn lại sẽ được gán giá trị 0. Giả định chuỗi s chỉ chứa các ký tự 0 hoặc 1 và có độ dài không quá 32

Bài 3: Dùng 2 hàm đã viết để khảo sát các câu hỏi:

- **1.3E+20** có biểu diễn nhị phân ra sao
- Số float nhỏ nhất lớn hơn 0 là số nào? Biểu diễn nhị phân của nó?
- Những trường hợp nào tạo ra các số đặc biệt (kiểu float) (viết chương trình thử nghiệm và giải thích kết quả):
 - o Số vô cùng (inf)
 - o Số báo lỗi NaN
 - o Ví dụ: $X - (+\infty)$, $(+\infty) - (+\infty)$, $X/0$, $0/0$, ∞/∞ , $\text{sqrt}(X)$ với $X < 0, \dots$ (Tham khảo thêm một số trường hợp trong slide 13 bài giảng Số chấm động)

Bài 4: Khảo sát các trường hợp sau đây (viết chương trình thử nghiệm và giải thích kết quả):

1. Chuyển đổi float -> int -> float. Kết quả như ban đầu ?
2. Chuyển đổi int -> float -> int. Kết quả như ban đầu ?
3. Phép cộng số chấm động có tính kết hợp ?


$$(x+y)+z = x+(y+z)$$

Với i là biến kiểu int, f là biến kiểu float

4. `i = (int) (3.14159 * f);`
5. `f = f + (float) i;`
6. `if (i == (int)((float) i)) { printf("true"); }`
7. `if (i == (int)((double) i)) { printf("true"); }`
8. `if (f == (float)((int) f)) { printf("true"); }`
9. `if (f == (double)((int) f)) { printf("true"); }`

Hướng dẫn thuật toán

Cách 1: Thực hiện theo các thuật toán

 **Thuật toán: chuyển từ giá trị số chấm động sang biểu diễn nhị phân.**

Ví dụ: chuyển -12.625 sang biểu diễn nhị phân

1. Chuyển phần trị sang dạng nhị phân.
 - a. Phần nguyên: theo thuật toán chuyển từ số thập phân sang nhị phân

$$12_{10} = 1100_2$$
 - b. Phần thập phân: Lặp lại việc nhân phần thập phân với 2 cho tới khi phần thập phân bằng 0 (hoặc đủ số bit phần trị). Tại mỗi bước nhân, sẽ phát sinh 1 bit tùy thuộc vào phần nguyên của kết quả phép nhân

$$0.625 \times 2 = 1.25 \rightarrow 1$$

$$0.25 \times 2 = 0.5 \rightarrow 0$$

$$0.5 \times 2 = 1.0 \rightarrow 1$$
 Như vậy, biểu diễn nhị phân của phần trị:

$$1100.101$$
2. Chuẩn hóa

$$1100.101 = 1.100101 \times 2^3$$
3. Điền các bit vào các trường theo chuẩn:
 - Dấu: số âm nên bit dấu
 - $S = 1$
 - Phần mũ: cộng phần mũ của 2 (tìm được sau bước chuẩn hóa) với phần bias ($2^{k-1}-1$)

$$E = 3_{10} + 127_{10} = 130_{10} = 10000010_2 \text{ (với bias} = 2^{8-1} - 1 = 127)$$

- Phần trị: thêm bit 0 vào bên phải cho đủ số bit phần trị

$$M = 100101000000000000000000$$

Như vậy, biểu diễn nhị phân của số chấm động -12.625 là

$$1 \ 10000010 \ 100101000000000000000000$$

Thuật toán: chuyển từ biểu diễn nhị phân của số chấm động sang biểu diễn thập phân

Ví dụ: chuyển dãy bit 01000100001101100001000000000000 sang biểu diễn thập phân

1. Chia thành 3 thành phần

$$S = 0$$

$$E = 10001000$$

$$M = 011011000010000000000000$$

2. Tính phần mũ: đổi dãy nhị phân thành số thập phân rồi trừ cho bias ($2^{k-1} - 1$)

$$E = 10001000_2 = 136_{10} = 136 - 127 = 9$$

3. Tính phần trị: phục hồi bit 1 ở phần nguyên và loại bỏ các bit 0 thừa ở cuối

$$M = 1.01101100001$$

4. Kết hợp phần trị với phần mũ:

$$1.01101100001 \times 2^9 = 1011011000.01$$

Nếu trị tuyệt đối của phần mũ quá lớn, không thể kết hợp với phần trị như trên, thì có thể thực hiện tính giá trị của phần mũ rồi nhân với phần trị sẽ cho kết quả gần đúng

5. Chuyển dãy nhị phân thành giá trị số chấm động tương ứng

Lũy thừa	2^9	2^8	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0	2^{-1}	2^{-2}
Giá trị tương ứng	512	256	128	64	32	16	8	4	2	1	0.5	0.25
Bits	1	0	1	1	0	1	1	0	0	0	0	1
Giá trị tính được	512		+ 128	+ 64		+ 16	+ 8					+ 0.25
	$= 728.25$											

6. Thêm dấu:

$$S = 0 \text{ nên là số dương}$$

Như vậy, giá trị thập phân tương ứng với dãy bit 01000100001101100001000000000000 là 728.25

Cách 2:

Dựa vào bản chất lưu trữ nhị phân, có thể dễ dàng xuất dạng biểu diễn nhị phân của số chấm động hay chuyển dãy bit biểu diễn thành các giá trị lưu trữ tương ứng dựa vào các thao tác luận lý AND, OR, SHIFT LEFT, SHIFT RIGHT,...

Chú ý:

Không thể thực hiện các thao tác luận lý trực tiếp trên số thực

float x;

x >> 1; ← báo lỗi

Nên, cần lấy con trỏ tới vùng nhớ chứa số thực. Nhưng...

long *p = &x; ← báo lỗi

Do đó, cần thực hiện ép kiểu như sau:

long *p = (long *)&x;

Và thực hiện các thao tác luận lý:

*p >> 1;

Mở rộng

Mô phỏng một số phép toán trên số chấm động: cộng, trừ, nhân, chia