



BIỂU DIỄN SỐ CHẠM ĐỘNG

Môn học: Kiến trúc máy tính & Hợp ngữ

Đặt vấn đề

- Biểu diễn số 123.375_{10} sang hệ nhị phân?
 - Ý tưởng đơn giản: Biểu diễn phần nguyên và phần thập phân riêng lẻ
 - Với phần nguyên: Dùng 8 bit ($[0_{10}, 255_{10}]$)
 $123_{10} = 64 + 32 + 16 + 8 + 2 + 1 = 0111\ 1011_2$
 - Với phần thập phân: Tương tự dùng 8 bit
 $0.375 = 0.25 + 0.125 = 2^{-2} + 2^{-3} = 0110\ 0000_2$
- $123.375_{10} = 0111\ 1011.0110\ 0000_2$
- Tổng quát công thức khai triển của số thập phân hệ nhị phân:

$$\underbrace{x_{n-1}x_{n-2}\dots x_0}_{\text{Phần nguyên}}.\underbrace{x_{-1}x_{-2}\dots x_{-m}}_{\text{Phần thập phân}} = \underbrace{x_{n-1}.2^{n-1} + x_{n-2}.2^{n-2} \dots + x_0.2^0}_{\text{Phần nguyên}} + \underbrace{x_{-1}.2^{-1} + x_{-2}.2^{-2} + \dots + x_{-m}.2^{-m}}_{\text{Phần thập phân}}$$

Đặt vấn đề

- Tuy nhiên...với 8 bit:

- Phần nguyên lớn nhất có thể biểu diễn: 255
- Phần thập phân nhỏ nhất có thể biểu diễn: $2^{-8} \sim 10^{-3} = 0.001$

→ Biểu diễn số nhỏ như 0.0001 (10^{-4}) hay 0.000001 (10^{-5})?

→ Một giải pháp: Tăng số bit phần thập phân

- Với 16 bit cho phần thập phân: $\min = 2^{-16} \sim 10^{-5}$
- Có vẻ không hiệu quả...Cách tốt hơn ?

→ Floating Point Number (Số thực dấu chấm động)



Floating Point Number ?

- Giả sử ta có số (ở dạng nhị phân)

$$X = 0.\underbrace{00000000000000}_{14 \text{ số } 0}11_2 = (2^{-15} + 2^{-16})_{10}$$

→ $X = 0.11_2 * (2^{-14})_{10} (= (2^{-1} + 2^{-2}).2^{-14} = 2^{-15} + 2^{-16})$

→ Thay vì dùng 16 bit để lưu trữ phần thập phân, ta có thể chỉ cần 6 bit:

$$X = 0.11 \ 1110$$

→ Cách làm: Di chuyển vị trí dấu chấm sang phải 14 vị trí, dùng 4 bit để lưu trữ số 14 này

→ Đây là ý tưởng **cơ bản** của số thực dấu chấm động (floating point number)

Chuẩn hóa số thập phân

- Trước khi các số được biểu diễn dưới dạng số chấm động, chúng cần được chuẩn hóa về dạng:
 $\pm 1.F * 2^E$
 - *F*: Phần thập phân không dấu (định trị - Significant)
 - *E*: Phần số mũ (Exponent)
- Ví dụ:
 - $+0.09375_{10} = 0.00011_2 = +1.1 * 2^{-4}$
 - $-5.25_{10} = 101.01_2 = -1.0101 * 2^2$

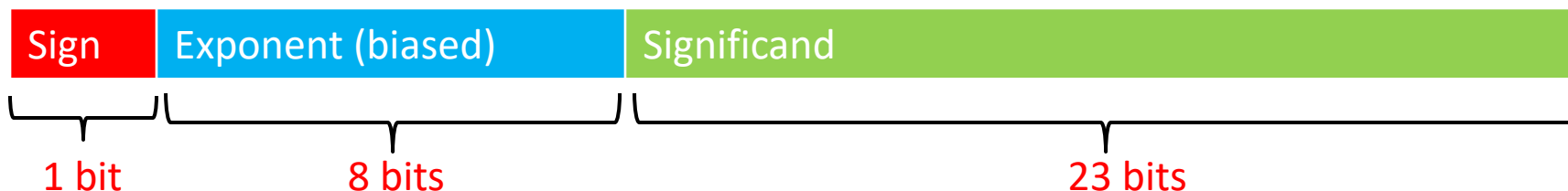
Biểu diễn số chấm động

- Có nhiều chuẩn nhưng hiện nay chuẩn IEEE 754 được dùng nhiều nhất để lưu trữ số thập phân theo dấu chấm động trong máy tính, gồm 2 dạng:
(slide sau)

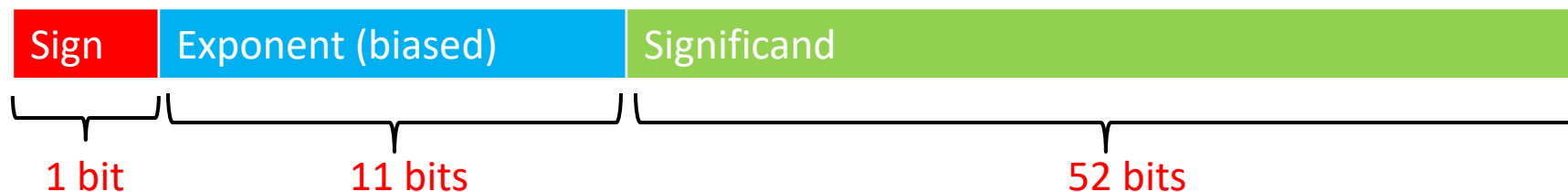


Biểu diễn số chấm động

- Số chấm động chính xác đơn (32 bits):



- Số chấm động chính xác kép (64 bits):



- Sign:** Bit dấu (1: Số âm, 0: Số dương)
- Exponent:** Số mũ (Biểu diễn dưới dạng số quá K (Biased)) với
 - Chính xác đơn: $K = 127$ ($2^{n-1} - 1 = 2^{8-1} - 1$) với n là số bit lưu trữ Exponent
 - Chính xác kép: $K = 1023$ ($2^{n-1} - 1 = 2^{11-1} - 1$)
- Significand (Fraction):** Phần định trị (phần lẻ sau dấu chấm)

Ví dụ

Biểu diễn số thực sau theo dạng số chấm động chính xác đơn (32 bit): **$X = -5.25$**

- **Bước 1:** Đổi X sang hệ nhị phân

$$X = -5.25_{10} = -101.01_2$$

- **Bước 2:** Chuẩn hóa theo dạng **$\pm 1.F * 2^E$**

$$X = -5.25 = -101.01 = -1.0101 * 2^2$$

- **Bước 3:** Biểu diễn Floating Point

- Số âm: bit dấu Sign = 1

- Số mũ $E = 2 \rightarrow$ Phần mũ exponent với số thừa $K=127$ được biểu diễn:

$$\rightarrow \text{Exponent} = E + 127 = 2 + 127 = 129_{10} = 1000\ 0001_2$$

- Phần định trị = 0101 0000 0000 0000 0000 000 (Thêm 19 số 0 cho đủ 23 bit)

\rightarrow Kết quả nhận được: 1 1000 0001 0101 0000 0000 0000 0000 000

Thảo luận về exponent

- Vì sao phần số mũ exponent không giữ nguyên lại phải lưu trữ dưới dạng số quá K (Dạng biased)?
 - Giả sử trong số chấm động chính xác đơn (32 bits), ta dùng 8 bits để lưu giá trị exponent (biểu diễn dưới dạng số quá K), vậy miền giá trị của nó là [0, 255]
- Với $K = 127$, số mũ gốc ban đầu có miền giá trị [-127, 128]
- Miền giá trị này **khá vô lý**, vậy tại sao chúng ta không chọn số $K = 128$ để miền giá trị gốc là [-128, 127] như bình thường?



Câu hỏi 1 - Đáp án

- Sở dĩ Exponent được lưu trữ dưới dạng Biased vì ta muốn chuyển từ miền giá trị **số có dấu** sang **số không dấu** (vì trong biased, số k được chọn để sau khi cộng số bất kỳ trong miền giá trị gốc, kết quả là số luôn dương)
→ Dễ dàng so sánh, tính toán



Câu hỏi 2 - Đáp án

- Số K được chọn là 127 mà không phải là 128 vì tại bước 2 trước khi biểu diễn thành số chấm động, chúng ta cần phải chuẩn hóa thành dạng $\pm 1.F * 2^E$
 - Tức là chúng ta sẽ **luôn luôn để dành 1 bit** (số 1) phía trước dấu chấm chứ không đẩy sang trái hết
- Với 8 bit, số mũ gốc ban đầu không thể đạt mức nhỏ nhất là -128 mà chỉ là -127
- Do vậy ta chỉ cần chọn $K = 127$ là được



Vậy thì...

- Khi muốn biểu diễn số 0 thì ta không thể tìm ra bit trái nhất có giá trị = 1 để đẩy dấu chấm động, vậy làm sao chuẩn hóa về dạng $\pm 1.F * 2^E$?
 - Với số dạng $\pm 0.F * 2^{-127}$ thì chuẩn hóa được nữa không?
 - Với $K = 127$, exponent lớn nhất sẽ là 255
- Số mũ gốc ban đầu lớn nhất là $255 - 127 = +128$
- **Vô lý** vì với 8 bit có dấu ta không thể biểu diễn được số +128 ?



Trả lời

- Vì đó là những **số thực đặc biệt**, ta không thể biểu diễn bằng dấu chấm động.



Số thực đặc biệt

- Số 0 (zero)
 - Exponent = 0, Significand = 0
- Số không thể chuẩn hóa (denormalized)
 - Exponent = 0, Significand \neq 0
- Số vô cùng (infinity)
 - Exponent = 111...1 (toàn bit 1), Significand = 0
- Số báo lỗi (NaN – Not a Number)
 - Exponent = 111...1 (toàn bit 1), Significand \neq 0



Normalized number

- Largest positive normalized number: $+1.[23 \text{ số } 1] * 2^{127}$

| S | Exp | Significand (Fraction) |
|---|-----------|------------------------------|
| - | ----- | ----- |
| 0 | 1111 1110 | 1111 1111 1111 1111 1111 111 |

- Smallest positive normalized number: $+1.[23 \text{ số } 0] * 2^{-126}$

| S | Exp | Significand (Fraction) |
|---|-----------|------------------------------|
| - | ----- | ----- |
| 0 | 0000 0001 | 0000 0000 0000 0000 0000 000 |

- Tương tự cho số negative (số âm)



Denormalized number

- Largest positive denormalized number: $+0.[23 \text{ số } 1] * 2^{-127}$

| S | Exp | Significand (Fraction) |
|---|-----------|------------------------------|
| - | ----- | ----- |
| 0 | 0000 0000 | 1111 1111 1111 1111 1111 111 |

Tuy nhiên IEEE 754 quy định là $+0.[23 \text{ số } 1] * 2^{-126}$ vì muốn tiến gần hơn với "Smallest positive normalized number = $+1.[23 \text{ số } 0] * 2^{-126}$ "

- Smallest positive denormalized number: $+0.[22 \text{ số } 0]1 * 2^{-127}$

| S | Exp | Significand (Fraction) |
|---|-----------|------------------------------|
| - | ----- | ----- |
| 0 | 0000 0000 | 0000 0000 0000 0000 0000 001 |

Tuy nhiên IEEE 754 quy định là $+0.[22 \text{ số } 0]1 * 2^{-126}$

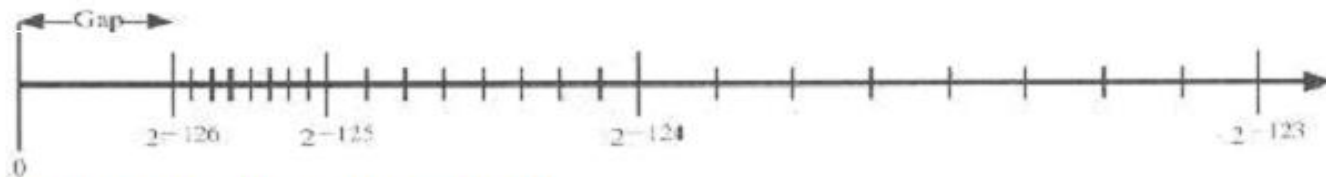
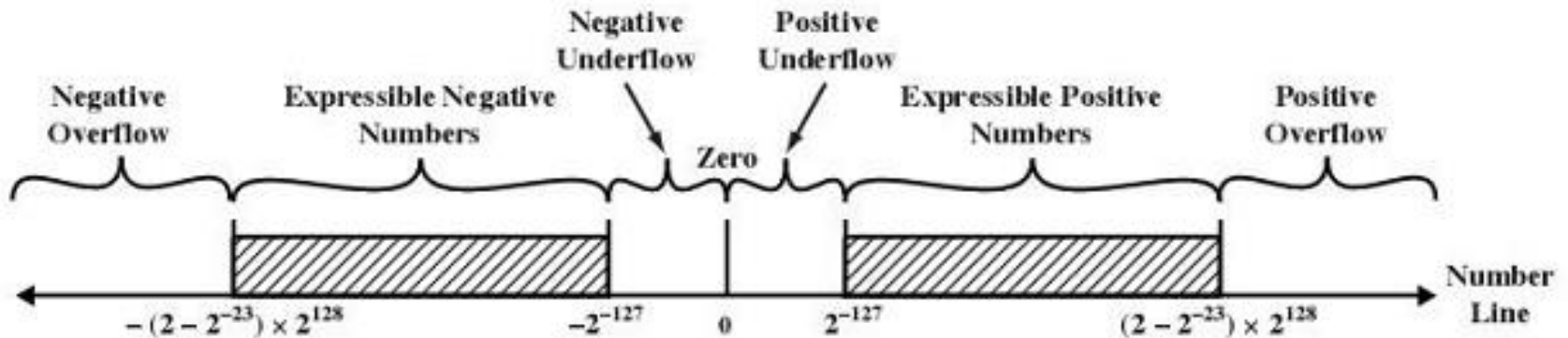
- Tương tự cho số negative (số âm)



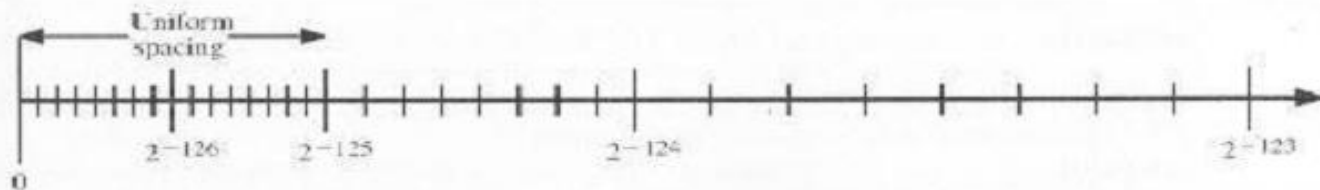
Ví dụ: $n = 4, m = 3, \text{bias} = 7$

| | s | exp | frac | E | Value |
|----------------------|-------|------|------|-----|---|
| Denormalized numbers | 0 | 0000 | 000 | -6 | 0 |
| | 0 | 0000 | 001 | -6 | $1/8 * 1/64 = 1/512$ ← closest to zero |
| | 0 | 0000 | 010 | -6 | $2/8 * 1/64 = 2/512$ |
| | ... | | | | |
| | 0 | 0000 | 110 | -6 | $6/8 * 1/64 = 6/512$ |
| | 0 | 0000 | 111 | -6 | $7/8 * 1/64 = 7/512$ ← largest denorm |
| | | | | | |
| Normalized numbers | 0 | 0001 | 000 | -6 | $8/8 * 1/64 = 8/512$ ← smallest norm |
| | 0 | 0001 | 001 | -6 | $9/8 * 1/64 = 9/512$ |
| | ... | | | | |
| | 0 | 0110 | 110 | -1 | $14/8 * 1/2 = 14/16$ |
| | 0 | 0110 | 111 | -1 | $15/8 * 1/2 = 15/16$ ← closest to 1 below |
| | 0 | 0111 | 000 | 0 | $8/8 * 1 = 1$ |
| | 0 | 0111 | 001 | 0 | $9/8 * 1 = 9/8$ ← closest to 1 above |
| | 0 | 0111 | 010 | 0 | $10/8 * 1 = 10/8$ |
| | ... | | | | |
| | 0 | 1110 | 110 | 7 | $14/8 * 128 = 224$ |
| | 0 | 1110 | 111 | 7 | $15/8 * 128 = 240$ ← largest norm |
| | | | | | |
| | 0 | 1111 | 000 | n/a | inf |

Phân bố các số thực (32 bits)



Without denormalized numbers



With denormalized numbers

Chuẩn IEEE 754

| Parameter | Format | | | |
|---------------------------|----------------------|-----------------|------------------------|-----------------|
| | Single | Single Extended | Double | Double Extended |
| Word width (bits) | 32 | ≥ 43 | 64 | ≥ 79 |
| Exponent width (bits) | 8 | ≥ 11 | 11 | ≥ 15 |
| Exponent bias | 127 | unspecified | 1023 | unspecified |
| Maximum exponent | 127 | ≥ 1023 | 1023 | ≥ 16383 |
| Minimum exponent | -126 | ≤ -1022 | -1022 | ≤ -16382 |
| Number range (base 10) | $10^{-38}, 10^{+38}$ | unspecified | $10^{-308}, 10^{+308}$ | unspecified |
| Significand width (bits)* | 23 | ≥ 31 | 52 | ≥ 63 |
| Number of exponents | 254 | unspecified | 2046 | unspecified |
| Number of fractions | 2^{23} | unspecified | 2^{52} | unspecified |
| Number of values | 1.98×2^{31} | unspecified | 1.99×2^{63} | unspecified |

* not including implied bit



Bài tập 1

- Biểu diễn số thực sau theo dạng số chấm động chính xác đơn (32 bit): **$X = -12.625$**

- **Bước 1:** Đổi X sang hệ nhị phân

$$X = -12.625_{10} = -1100.101_2$$

- **Bước 2:** Chuẩn hóa theo dạng **$\pm 1.F * 2^E$**

$$X = -12.625_{10} = -1100.101_2 = -1.100101 * 2^3$$

- **Bước 3:** Biểu diễn Floating Point

- Số dương: bit dấu Sign = 1

- Số mũ E = 3 → Phần mũ exponent với số thừa K=127 được biểu diễn:

$$\rightarrow \text{Exponent} = E + 127 = 3 + 127 = 130_{10} = 1000\ 0010_2$$

- Phần định trị = 1001 0100 0000 0000 0000 000 (Thêm 17 số 0 cho đủ 23 bit)

→ Kết quả nhận được: 1 1000 0010 1001 0100 0000 0000 0000 000



Bài tập 2

- Biểu diễn số thực sau theo dạng số chấm động chính xác đơn (32 bit): **X = -3050**

- **Bước 1:** Đổi X sang hệ nhị phân

$$X = -3050_{10} = -1011\ 1110\ 1010_2$$

- **Bước 2:** Chuẩn hóa theo dạng **$\pm 1.F * 2^E$**

$$X = -3050_{10} = -1011\ 1110\ 1010_2 = -1.01111101010 * 2^{11}$$

- **Bước 3:** Biểu diễn Floating Point

- Số âm: bit dấu Sign = 1
- Số mũ E = 11 → Phần mũ exponent với số thừa K=127 được biểu diễn:
→ Exponent = E + 127 = 11 + 127 = 138₁₀ = 1000 1010₂
- Phần định trị = 0111 1101 0100 0000 0000 000 (Thêm 12 số 0 cho đủ 23 bit)

→ Kết quả nhận được: 1 1000 1010 0111 1101 0100 0000 0000 000



Bài tập 3

- Biểu diễn số thực sau theo dạng số chấm động chính xác đơn (32 bit): $X = +1.1 * 2^{-128}$
 - **Lưu ý:**
 - Số X: positive number
 - $X < \text{Smallest positive normalized number: } +1.[23 \text{ số } 0] * 2^{-126}$
→ số X là số không thể chuẩn hóa (denormalized number)
→ **Chuyển X về dạng: $X = +0.011 * 2^{-126}$**
 - **Bước 3:** Biểu diễn Floating Point
 - Số dương: bit dấu Sign = 0
 - Vì đây là số không thể chuẩn hóa → Phần mũ exponent được biểu diễn: 0000 0000₂
 - Phần định trị = 0110 0000 0000 0000 0000 000
- Kết quả nhận được: 0 0000 0000 0110 0000 0000 0000 0000 000



Homework

- Sách W.Stalling – Computer Arithmetic, đọc chương 9
- Đọc file 04_FloatingPoint.doc
- Trả lời các câu hỏi:
 - Overflow, underflow?
 - Cộng trừ nhân chia trên số thực?
 - Quy tắc làm tròn?
 - NaN: nguyên tắc phát sinh?
 - Quiet NaN và Signaling NaN?

