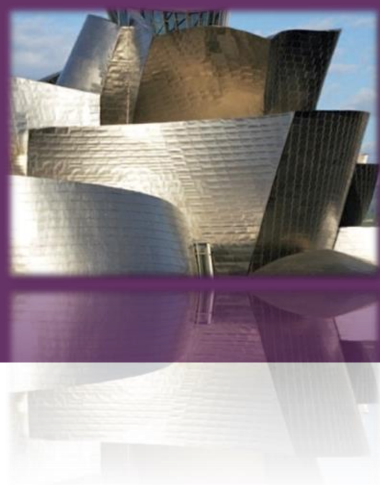


Kiến trúc máy tính

Chương 9

Bộ xử lý số học

+

Chương 9. Bộ xử lý số học

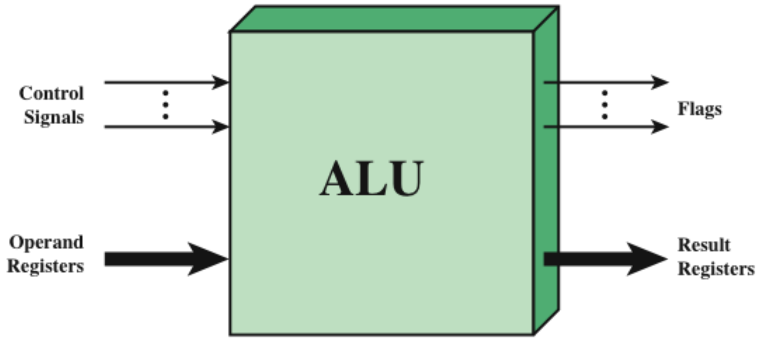
1. Đơn vị số học và logic
2. Biểu diễn số nguyên
3. Phép toán số học với số nguyên
4. Biểu diễn dấu chấm động
5. Phép toán với dấu chấm động

+

1. Đơn vị số học & logic (ALU)

- Phần của máy tính thực hiện phép toán số học và lôgíc trên dữ liệu
- Tất cả các bộ phận khác trong hệ thống máy tính (CU, thanh ghi, bộ nhớ, I/O) đưa dữ liệu tới ALU để xử lý, sau đó gửi lại kết quả
- Khối ALU được thực hiện sử dụng các linh kiện logic số có thể lưu trữ các số nhị phân và thực hiện các phép toán logic Boolean đơn giản

Đầu vào và đầu ra ALU



- Control Signals: các tín hiệu điều khiển được gửi đến từ CPU, điều khiển hoạt động của ALU
- Operand registers: các thanh ghi lưu trữ giá trị toán hạng của phép toán
- Result registers: các thanh ghi lưu trữ kết quả phép toán
- Flags: các cờ. Vd: cờ tràn để đánh dấu kết quả tính toán vượt quá kích thước thanh ghi đang lưu trữ

\+

2. Biểu diễn số nguyên

- Trong hệ nhị phân, số bất kì có thể được biểu diễn bằng:
  - Các con số 0 và 1, dấu âm (với số âm), dấu phẩy (với số có phần thập phân)

$$-1101.0101_2 = -13.3125_{10}$$

- Với mục đích lưu trữ và xử lý máy tính, không có các ký hiệu đặc biệt cho dấu trừ và dấu phẩy → chỉ sử dụng các bit 0,1 để biểu diễn
- Với số nguyên: có một số phương pháp biểu diễn số nguyên âm, các phương pháp này đều quy ước dùng bit quan trọng nhất (ngoài cùng bên trái) trong từ (word) để làm bit dấu: 0 → số dương, 1 → số âm
  - Biểu diễn dấu – độ lớn
  - Biểu diễn bù 2
  - *Các phương pháp biểu diễn này có thể được gọi là biểu diễn dấu phẩy tĩnh*

a. Biểu diễn dấu – độ lớn
- Sign-magnitude representation

- Đây là dạng biểu diễn đơn giản nhất
- Trong một từ n bit, sử dụng (n-1) bên phải biểu diễn độ lớn của số

$+18$	$= 00010010$	
-18	$= 10010010$	(sign magnitude)

- Nhược điểm:
  - Thực hiện phép cộng và phép trừ đòi hỏi phải xem xét cả dấu của các số và độ lớn của chúng
  - Có hai dạng biểu diễn của số 0: gây khó khăn khi thực hiện việc kiểm tra 0 trong một số phép toán

$+0_{10}$	$= 00000000$	
-0_{10}	$= 10000000$	(sign magnitude)

- Do những nhược điểm này, biểu diễn dấu – độ lớn hiếm khi được sử dụng trong việc mã hóa phần số nguyên trong ALU

+
b. Biểu diễn bù 2

- Sử dụng bit quan trọng nhất làm bit dấu
- Khác với biểu diễn dấu – độ lớn ở cách biểu diễn các số âm

Miền giá trị (n bit)	-2^{n-1} đến $2^{n-1}-1$
Biểu diễn số 0	Một
Biểu diễn số âm	Lấy bù của số dương tương ứng sau đó cộng thêm 1
Mở rộng chiều dài chuỗi bit	Thêm 1 bit vào bên trái, điền giá trị dấu vào
Luật tràn	Khi cộng hai số cùng dấu, nếu kết quả có dấu ngược lại → tràn
Luật trừ	Khi trừ A cho B, lấy bù 2 của B sau đó cộng với A

+

b. Biểu diễn bù 2

- Một số nguyên A, biểu diễn dưới dạng bù 2, n-bit:
- Nếu A là số dương

$$A = \sum_{i=0}^{n-2} 2^i a_i \qquad \text{for } A \geq 0$$

  - Bit $n-1$ là bit dấu có giá trị 0
- Nếu A là số âm (A<0)

$$A = -2^{n-1} a_{n-1} + \sum_{i=0}^{n-2} 2^i a_i$$

  - Bit dấu $n-1$ có giá trị 1

❖ ***Trong đó, a_i là giá trị bit tại vị trí i***

+

Hộp giá trị

−128	64	32	16	8	4	2	1

(a) An eight-position twos complement value box

−128	64	32	16	8	4	2	1
1	0	0	0	0	0	1	1

−128 +2 +1 = −125

(b) Convert binary 10000011 to decimal

−128	64	32	16	8	4	2	1
1	0	0	0	1	0	0	0

−120 = −128 +8

(c) Convert decimal −120 to binary

Bảng 10.2
Biểu diễn số nguyên 4-Bit

Biểu diễn thập phân	Biểu diễn dấu – độ lớn	Biểu diễn bù 2
+8	—	—
+7	0111	0111
+6	0110	0110
+5	0101	0101
+4	0100	0100
+3	0011	0011
+2	0010	0010
+1	0001	0001
+0	0000	0000
–0	1000	—
–1	1001	1111
–2	1010	1110
–3	1011	1101
–4	1100	1100
–5	1101	1011
–6	1110	1010
–7	1111	1001
–8	—	1000

+

Mở rộng phạm vi

- Trong một số trường hợp, ta muốn biểu diễn một số n-bit sang dạng biểu diễn m-bit ($m > n$): mở rộng phạm vi biểu diễn
- Trong biểu diễn dấu – độ lớn: di chuyển bit dấu tới vị trí mới ngoài cùng bên trái và điền (m-n) bit 0 vào sau bit dấu
- Biểu diễn số bù 2:
- Quy tắc: di chuyển bit dấu sang vị trí ngoài cùng bên trái và điền vào bằng bản sao bit dấu
  - Đối với số dương, điền 0 vào, và số âm thì điền vào số 1
  - Đây được gọi là phần mở rộng dấu

Mở rộng phạm vi
Ví dụ số bù 2

- Với số dương:
  - +35 = 0010 0011 (8 bit)
  - +35 = 0000 0000 0010 0011 (16 bit)
  - → Thêm 8 bit 0 vào bên trái
- Với số âm:
  - -79 = 1011 0001 (8 bit)
  - -79 = 1111 1111 1011 0001 (16 bit)
  - → Thêm 8 bit 1 vào bên trái
- Kết luận: mở rộng sang bên trái 8 bit bằng bit dấu

2. Phép toán trên số nguyên

a. Phép đảo (phép phủ định)

b. Phép cộng

c. Phép trừ - luật trừ

d. Phép nhân

e. Phép chia

+

a. Phép đảo (phép phủ định)

- Quy tắc:
  - Biểu diễn dấu – độ lớn: đảo bit dấu
  - Biểu diễn bù 2:
    - Đảo từng bit (kể cả bit dấu)
    - Cộng 1

$$\begin{array}{rcll} +18 & = & 00010010 & \text{(twos complement)} \\ \text{bitwise complement} & = & 11101101 & \\ & & +\qquad\quad 1 & \\ & & 11101110 = -18 & \end{array}$$

- Đảo của đảo một số là chính nó

$$\begin{array}{rcll} -18 & = & 11101110 & \text{(twos complement)} \\ \text{bitwise complement} & = & 00010001 & \\ & & \underline{+\qquad\quad 1} & \\ & & 00010010 = +18 & \end{array}$$

+

Số bù 2: xét hai trường hợp đặc biệt

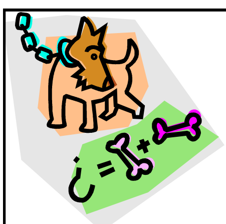
$$\begin{array}{rcll} 0 & = & 00000000 & \text{(twos complement)} \\ \text{bitwise complement} & = & 11111111 & \\ & & \underline{+\qquad\quad 1} & \\ & & 100000000 = 0 & \end{array}$$

$$\begin{array}{rcll} -128 & = & 10000000 & \text{(twos complement)} \\ \text{bitwise complement} & = & 01111111 & \\ & & \underline{+\qquad\quad 1} & \\ & & 10000000 = -128 & \end{array}$$

b. Phép cộng

- Phép cộng được thực hiện bình thường như cộng hai số nhị phân
- Trong một số trường hợp, xuất hiện thêm 1b (bit bôi đen) → bỏ qua các bit này

1001 = −7 +0101 = 5 1110 = −2 (a) (−7) + (+5)	1100 = −4 +0100 = 4 10000 = 0 (b) (−4) + (+4)
0011 = 3 +0100 = 4 0111 = 7 (c) (+3) + (+4)	1100 = −4 +1111 = −1 11011 = −5 (d) (−4) + (−1)
0101 = 5 +0100 = 4 1001 = Overflow (e) (+5) + (+4)	1001 = −7 +1010 = −6 10011 = Overflow (f) (−7) + (−6)



Nguyên tắc

Tràn

- **Tràn ô nhớ**: khi kết quả của một phép toán quá lớn vượt qua phạm vi biểu diễn của ô nhớ
- Số bù 2: **Tràn** ô nhớ xảy ra nếu hai số cùng dấu cộng với nhau mà kết quả thu được lại là một số có dấu ngược với dấu của hai số đó
- Khi phát hiện tràn, ALU cần phải báo hiệu việc này để CPU không sử dụng kết quả

c. Phép trừ

Nguyên tắc

Trừ

NGUYÊN TẮC TRỪ:

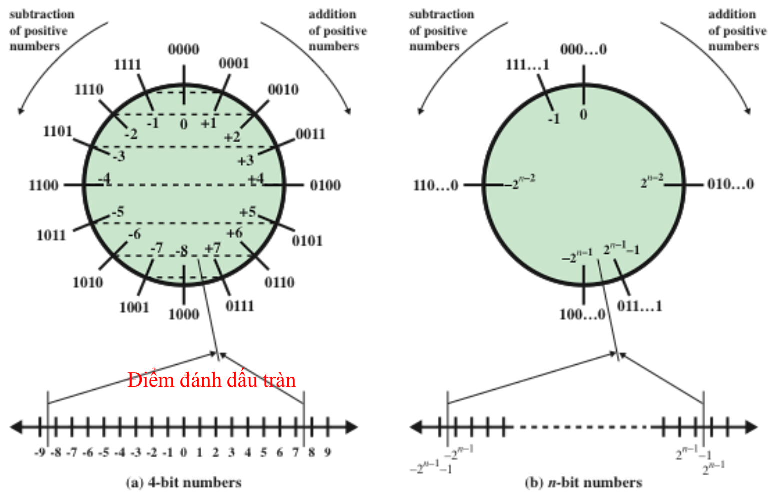
Lấy bù 2 (đảo) của số trừ và cộng với số bị trừ.

Phép trừ

0010 = 2 +1001 = −7 1011 = −5 (a) M = 2 = 0010 S = 7 = 0111 −S = 1001	0101 = 5 +1110 = −2 10011 = 3 (b) M = 5 = 0101 S = 2 = 0010 −S = 1110
1011 = −5 +1110 = −2 11001 = −7 (c) M = −5 = 1011 S = 2 = 0010 −S = 1110	0101 = 5 +0010 = 2 0111 = 7 (d) M = 5 = 0101 S = −2 = 1110 −S = 0010
0111 = 7 +0111 = 7 1110 = Overflow (e) M = 7 = 0111 S = −7 = 1001 −S = 0111	1010 = −6 +1100 = −4 10110 = Overflow (f) M = −6 = 1010 S = 4 = 0100 −S = 1100

Figure 10.4 Subtraction of Numbers in Twos Complement Representation (M − S)

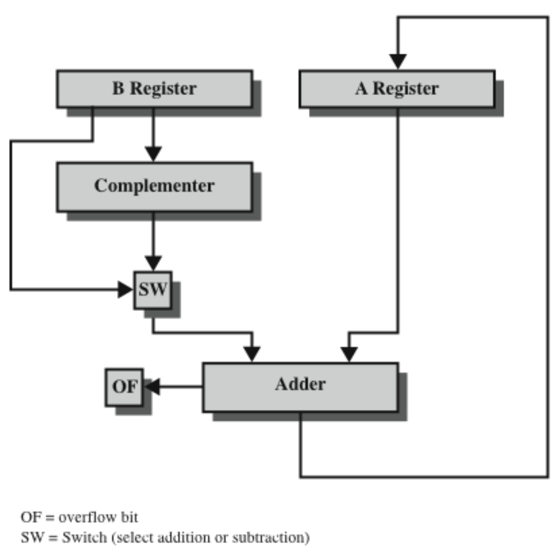
Mô tả hình học của số nguyên bù 2



Hình 10.5 Mô tả hình học của số nguyên bù 2

Phần cứng thực hiện cộng và trừ

- Thanh ghi A, B (A, B Register): lưu trữ hai số
- Với phép trừ: thanh ghi B lưu trữ số trừ
- Complementer: bộ lấy bù 2
- SW (switch): lựa chọn cộng hoặc trừ
- Bộ cộng (Adder): thực hiện phép toán và đưa ra cờ tràn (nếu có)
- Cờ tràn (OF-overflow bit):
  - 0: không tràn
  - 1: tràn
- Kết quả có thể được lưu trữ ở thanh ghi thứ 3 hoặc một trong 2 thanh ghi A, B



1. Biểu diễn số thập phân sau sang biểu diễn dấu-độ lớn và bù 2 nhị phân:

a) +512 **b) -29** **c) -91**

2. biểu diễn các giá trị bù 2 sau ra thập phân:

a) 1101011 **b) 0101101**

3. Giả sử dùng biểu diễn bù 2 8 bit. Thực hiện phép tính:

a) 6+13 **b) -6+13** **c) 6 – 13** **d) -6 – 13**

4. Thực hiện phép tính số học bù 2:

a. $\begin{array}{r} 111000 \\ -110011 \end{array}$ **b.** $\begin{array}{r} 11001100 \\ -\quad 101110 \end{array}$ **c.** $\begin{array}{r} 111100001111 \\ -110011110011 \end{array}$ **d.** $\begin{array}{r} 11000011 \\ -11101000 \end{array}$

d. Phép nhân

- So với phép cộng và phép trừ, phép nhân phức tạp hơn
- Nhiều thuật toán tính toán phép nhân khác nhau đã được sử dụng trong các máy tính khác nhau
- Trong phần này:
  i. Phép nhân giữa hai số nguyên không dấu
  ii. Phép nhân hai số biểu diễn bù 2

i. Phép nhân giữa hai số nguyên không dấu

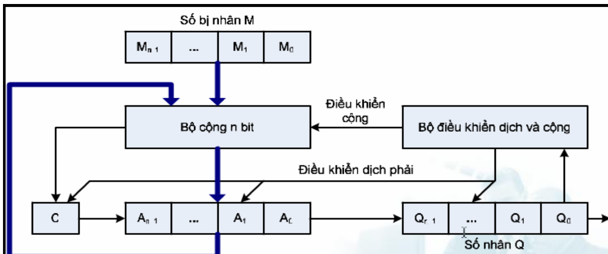
Các bước bằng tay:

- Tính các tích thành phần
- Nếu số nhân là bit 0, tích thành phần bằng 0. Nếu số nhân là bit 1, tích thành phần là *số bị nhân (multiplicand)*

1011	**Multiplicand (11)**
×1101	**Multiplier (13)**
1011	
0000	**Partial products**
1011	
1011	
10001111	**Product (143)**

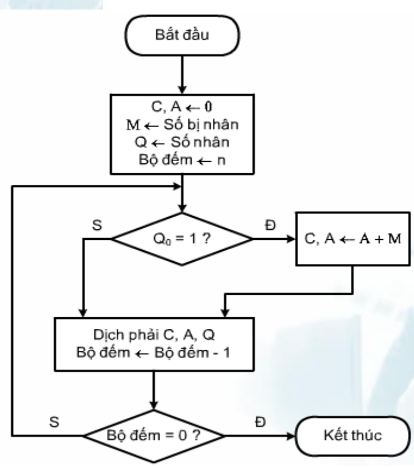
- Tính tổng các tích thành phần (mỗi tích dịch trái đơn vị so với tích trước đó)
- Tích của hai số *n*-bit là một số có kích thước *2n*-bit

Thuật toán phép nhân nhị phân không dấu theo hướng máy tính



Mô tả:

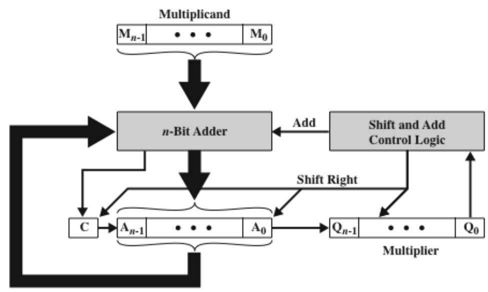
- Thanh ghi M: số bị nhân
- Thanh ghi Q: số nhân
- Thanh ghi 1bit C: ghi giá trị tràn của A+M (nếu có)
- ***Kết quả 2n-bit: thanh ghi A, Q***



+

Ví dụ phép nhân nhị phân không dấu

1011	Multiplicand (11)
×1101	Multiplier (13)
1011	Partial products
0000	
1011	
1011	
10001111	Product (143)



(a) Block Diagram

C	A	Q	M		
0	0000	1101	1011	Initial Values	
0	1011	1101	1011	Add	First Cycle
0	0101	1110	1011	Shift	
0	0010	1111	1011	Shift	Second Cycle
0	1101	1111	1011	Add	Third Cycle
0	0110	1111	1011	Shift	
1	0001	1111	1011	Add	Fourth Cycle
0	1000	1111	1011	Shift	

(b) Example from Figure 9.7 (product in A, Q)

ii. Phép nhân bù 2

1011	
×1101	
00001011	$1011 \times 1 \times 2^0$
00000000	$1011 \times 0 \times 2^1$
00101100	$1011 \times 1 \times 2^2$
01011000	$1011 \times 1 \times 2^3$
10001111	

- Số nguyên dương: nguyên tắc nhân giống số nguyên không dấu
- Số nguyên âm: do có sự xuất hiện của bit dấu nên nguyên tắc trên không còn đúng nữa

1001	(-7)	1001	(-7)
× 0011	(3)	× 1100	(-4)
11111001	$(-7) \times 2^0 = (-7)$	11100100	(-28)
11110010	$(-7) \times 2^1 = (-14)$	11001000	(-56)
11101011	(-21)	10101100	(-84) ???

Giải pháp 1

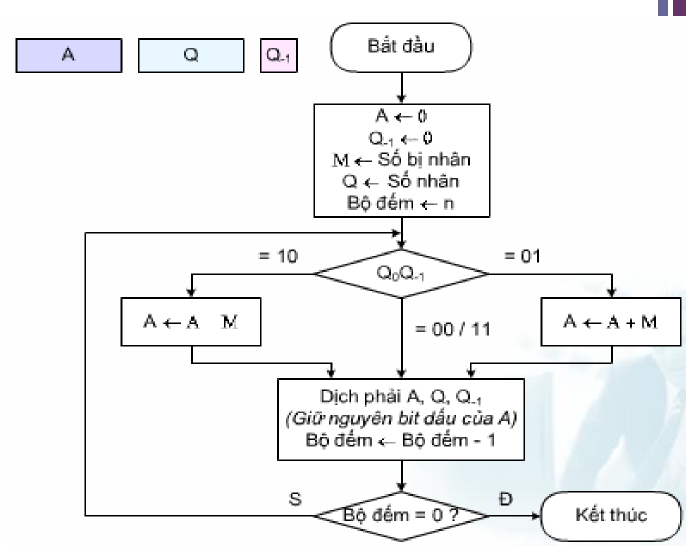
Sử dụng thuật toán nhân hai số không dấu:

- Bước 1: Chuyển đổi số nhân và số bị nhân thành số dương tương ứng.
- Bước 2: Nhân 2 số bằng thuật giải nhân số nguyên không dấu → được tích 2 số dương.
- Bước 3: Hiệu chỉnh dấu của tích:
  - Nếu 2 thừa số ban đầu cùng dấu thì tích nhận được ở bước 2 là kết quả cần tính.
  - Nếu 2 thừa số ban đầu khác dấu nhau thì kết quả là số bù 2 của tích nhận được ở bước 2.

Giải pháp 2: thuật toán Booth

- Tốc độ tính toán nhanh hơn do số lượng phép toán ít hơn
- Thuật toán chung cho cả số nguyên dương và âm

Thuật toán



+

Giải thích thuật toán

Ta có: $2^i + 2^{i-1} + \ldots + 2^j = 2^{i+1} - 2^j$ (với i>=j)

VD: M * 0<u>111</u>00<u>1</u>0 = M * $(2^7 - 2^4 + 2^2 - 2^1)$

- Quy tắc: duyệt từ trái sang phải:
  - Nếu gặp 10 thì trừ A đi M rồi dịch phải
  - Nếu gặp 01 thì cộng A với M rồi dịch phải
  - Nếu gặp 00 hay 11 thì chỉ dịch phải

Ví dụ thuật toán Booth

Ví dụ 1:

```
n = 4 bit, M = +7, Q = +3
M = 0111, Q = 0011, -M = 1001

  A     Q    Q-1
 0000  0011   0      ; khởi tạo
+1001
 1001  0011   0      ; A ← A - M
 1100  1001   1      ; dịch phải
 1110  0100   1      ; dịch phải
+0111
10101  0100   1      ; A ← A + M
 0010  1010   0      ; dịch phải
 0001  0101   0      ; dịch phải
```

Ví dụ 2:

```
n = 4 bit, M = +7, Q = -3
M = 0111, Q = 1101, -M = 1001

  A     Q    Q-1
 0000  1101   0      ; khởi tạo
+1001
 1001  1101   0      ; A ← A - M
 1100  1110   1      ; dịch phải
+0111
10011  1110   1      ; A ← A + M
 0001  1111   0      ; dịch phải
+1001
 1010  1111   0      ; A ← A - M
 1101  0111   1      ; dịch phải
 1110  1011   1      ; dịch phải
```

5/ x = 0101 và y = 1010 theo biểu diễn bù 2 (tức là x = 5,y = -6).

Hãy tính tích p = x * y bằng thuật toán Booth.

6/ Tính 29*23 bằng thuật toán Booth, trong đó mỗi số biểu diễn bằng 6 bit.

+

e. Phép chia

Phức tạp hơn phép nhân nhưng cũng sử dụng chung nguyên lý

Bằng tay

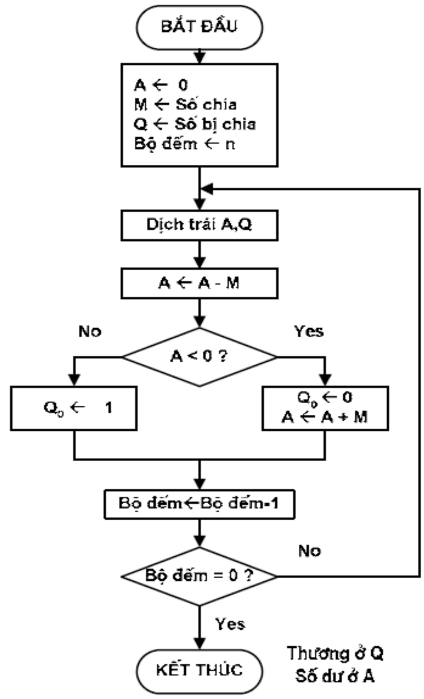
```
10010011 |1011       Số chia
 1011    |00001101   Thuong
  1110
  1011
    1111
    1011
     100             Số du
```

Thuật toán thực hiện phép chia hai số nguyên không dấu trong máy tính

- Số chia đặt trong thanh ghi M, số bị chia trong thanh ghi Q. A=0. Bộ đếm = n
- Tại mỗi bước:
  - A và Q dịch trái 1 đơn vị
  - Thực hiện $A - M$, nếu
    - $A - M \geq 0$ thì $Q_0 = 1$, $A = A - M$
    - $A - M < 0$ thì $Q_0 = 0$
  - Giảm bộ đếm và quay trở lại thực hiện vòng lặp đến khi Bộ đếm = 0

+

Lưu đồ thuật toán phép chia hai số nguyên không dấu



+

Chia số nguyên có dấu

- Bước 1: Chuyển đổi số chia và số bị chia thành số dương tương ứng
- Bước 2: Sử dụng thuật giải chia số nguyên không dấu để chia 2 số dương, kết quả nhận được là thương Q và phần dư R đều dương
- Bước 3: Hiệu chỉnh dấu kết quả theo quy tắc sau:

Số bị chia	Số chia	Thương	Số dư
+	+	giữ nguyên	giữ nguyên
+	-	đảo dấu	giữ nguyên
-	+	đảo dấu	đảo dấu
-	-	giữ nguyên	đảo dấu

+

Phép chia số bù 2

- Tương tự như phép nhân, do có bit dấu nên phải có thuật toán khác:
- Giả sử số chia V và số bị chia D dương và $|V| < |D|$
  - Nếu $|V| = |D|$: thương = 1, dư = 0.
  - Nếu $|V| > |D|$: thương = 0, dư = D
- Thuật toán như sau:
  - B1: Ghi số bù 2 của V vào thanh ghi M (M chứa số âm của V), ghi D vào thanh ghi A, Q, bộ đếm = n
  - B2: Dịch A,Q sang trái 1 đơn vị
  - B3: Tính A+M →A
  - B4: Kiểm tra:
    - Nếu A dương (bit dấu = 0), $Q_0 = 1$
    - Nếu A âm (bit dấu =1), $Q_0 = 0$, khôi phục A lại giá trị trước đó
  - B5: Giảm bộ đếm đi 1 đơn vị
  - Lặp lại các bước từ 2 đến 5 cho đến khi bộ đếm = 0
- Với các trường hợp V, D không dương, hiệu chỉnh kết quả dựa theo bảng ở trên

+

Ví dụ phép chia bù 2

A	Q	
0000	0111	Initial value
0000	1110	Shift
1101		Use twos complement of 0011 for subtraction
1101		Subtract
0000	1110	Restore, set Q_0 = 0
0001	1100	Shift
1101		
1110		Subtract
0001	1100	Restore, set Q_0 = 0
0011	1000	Shift
1101		
0000	1001	Subtract, set Q_0 = 1
0001	0010	Shift
1101		
1110		Subtract
0001	0010	Restore, set Q_0 = 0

Figure 10.17 Example of Restoring Twos Complement Division (7/3)

4. Biểu diễn dấu chấm động

a. Nguyên lý

- Quy ước: "dấu chấm" (point) được hiểu là kí hiệu ngăn cách giữa phần nguyên và phần lẻ của 1 số thực.
- Có 2 cách biểu diễn số thực trong máy tính:
  - Số dấu chấm tĩnh (fixed-point number):
    - Dấu chấm là cố định (số bit dành cho phần nguyên và phần lẻ là cố định)
    - Hạn chế: không thể biểu diễn số rất lớn hoặc số thập phân rất nhỏ. Phần thập phân trong thương của một phép chia hai số lớn có thể bị mất
    - Dùng trong các bộ vi xử lý hay vi điều khiển thế hệ cũ.
  - Số dấu chấm động (floating-point number):
    - Dấu chấm không cố định
    - Dùng trong các bộ vi xử lý hiện nay, có độ chính xác cao hơn.

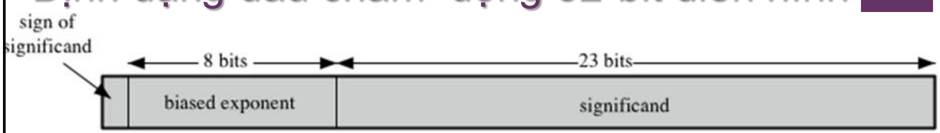
Biểu diễn số nhị phân dấu chấm động

- Một số nhị phân có thể được biểu diễn dưới dạng như sau:

$$\pm S \times B^{\pm E}$$

- Trong đó:
  - S: phần định trị
  - B: cơ số
  - E: số mũ
- Nếu B là một số định sẵn, ta chỉ cần lưu trữ S và B
- Vậy một số nhị phân có thể được lưu trữ trong máy tính với 3 trường sau:
  - Dấu
  - Giá trị S
  - Số mũ E

Định dạng dấu chấm động 32-bit điển hình



(a) Format

$$1.1010001 \times 2^{10100} = 0\ 10010011\ 10100010000000000000000 = 1.6328125 \times 2^{20}$$
$$-1.1010001 \times 2^{10100} = 1\ 10010011\ 10100010000000000000000 = -1.6328125 \times 2^{20}$$
$$1.1010001 \times 2^{-10100} = 0\ 01101011\ 10100010000000000000000 = 1.6328125 \times 2^{-20}$$
$$-1.1010001 \times 2^{-10100} = 1\ 01101011\ 10100010000000000000000 = -1.6328125 \times 2^{-20}$$

Ví dụ: các số trên được lưu trữ như sau:
B: ngầm định = 2
1 bit dấu: 0 nếu là số dương, 1 biểu diễn số âm
8 bit biased exponent: số mũ lệch
→ Số mũ thực tế = số mũ lệch - độ lệch (độ lệnh = $2^{k-1} - 1$, k: số bit phần mũ). Trong trường hợp này, độ lệch = 127
→ Số mũ thực tế nằm trong khoảng -127 đến +128
23 bit còn lại: phần định trị, được quy ước dạng 1.bbbbb...... Trong đó, số 1 đầu tiên là ngầm định

+ Chú ý: phần định trị - Significand

- Phần định trị có thể được biểu diễn thành nhiều dạng

Các cách viết sau đây là tương đương, trong đó phần định trị được biểu diễn dưới dạng nhị phân:
$0.110 * 2^5$
$110 * 2^2$
$0.0110 * 2^6$

- *Quy ước biểu diễn: đưa về dạng 1.bbbbbbb....*
- ***Khi lưu trữ không cần lưu trữ phần nguyên, chỉ cần lưu trữ phần thập phân. Dấu chấm ngầm định ngay sau 8 bit phần số mũ***

+
Ví dụ

Chuyển số 2.5436 ra số nhị phân biểu diễn dạng dấu chấm động 16b (4b phần số mũ và 11b phần thập phân)

- B1: chuyển 23.5436 ra số nhị phân: 10111.1001101 (xấp xỉ)
- B2: chuyển thành dạng cơ số - số mũ: $1.01111001101\ x\ 2^4$
- Biểu diễn dạng dấu chấm động:
  - Bit dấu: 0 (số dương)
  - Số mũ lệch = số mũ thực tế (4) + độ lệch (7)=$(11)_{10} = (1011)_2$

$$\mathbf{1.01111001101\ x\ 2^4 \leftrightarrow 0\ 1011\ 01111001101}$$

+
Miền giá trị

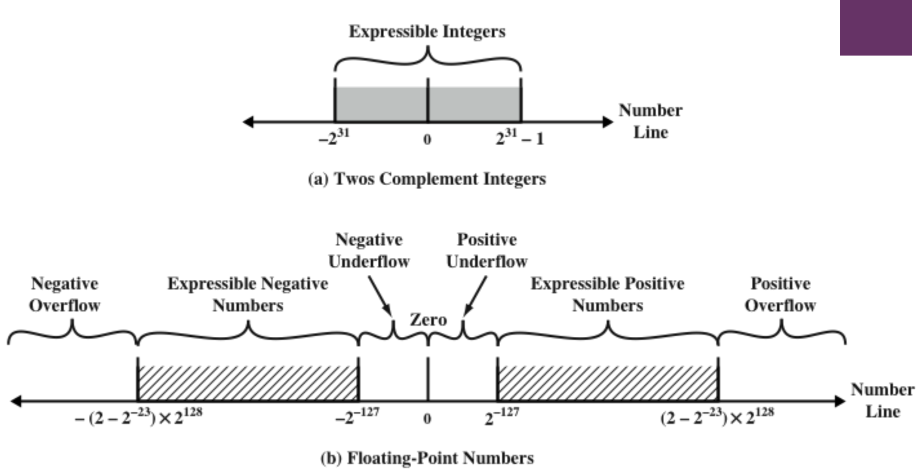


Figure 10.19 Expressible Numbers in Typical 32-Bit Formats

Mật độ số dấu phẩy động

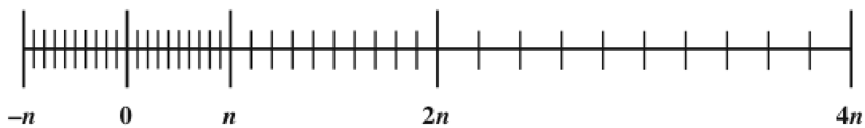


Figure 10.20 Density of Floating-Point Numbers

b. Chuẩn IEEE 754

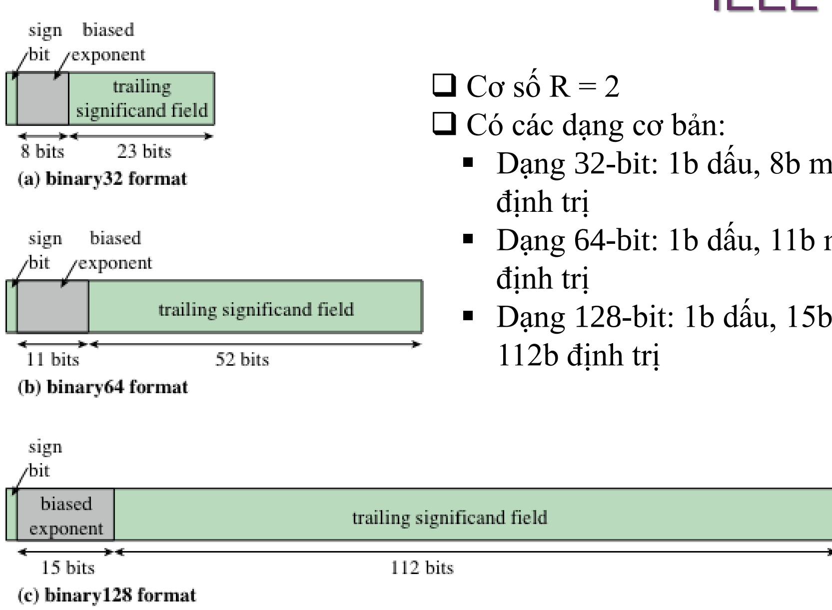
- Hiệp hội IEEE đã chuẩn hóa cho việc biểu diễn số dấu phẩy động nhị phân trong máy tính
- Mục đích:
  - Hỗ trợ tính di động của chương trình từ bộ xử lý này sang bộ xử lý khác
  - Khuyến khích phát triển các chương trình định hướng số học tinh vi hơn
- Chuẩn được công nhận rộng rãi và được sử dụng trên hầu hết các bộ VXL và bộ tính toán số học hiện đại
- IEEE 754-2008 quy định các định dạng biểu diễn dấu phẩy động nhị phân và thập phân
- Trong phần này chỉ đề cập đến dạng biểu diễn dấu phẩy động nhị phân

+

IEEE 754-2008

- IEEE 754-2008 định nghĩa 3 định dạng dấu chấm động sau:
- **Định dạng số học**
  - Được sử dụng để biểu diễn các toán hạng hoặc kết quả phép toán dưới dạng dấu chấm động.
- **Định dạng cơ bản:** quy định 5 dạng biểu diễn dấu chấm động:
  - Ba cho số nhị phân: chiều dài 32b, 64b và 128b
  - Hai cho thập phân: chiều dài 64b và 128b
- **Định dạng chuyển đổi**
  - Đưa ra dạng mã hoá nhị phân độ dài cố định cho phép trao đổi dữ liệu giữa các nền tảng khác nhau và có thể được sử dụng để lưu trữ.

Định dạng IEEE 754



- ❑ Cơ số R = 2
- ❑ Có các dạng cơ bản:
  - Dạng 32-bit: 1b dấu, 8b mũ, 23b định trị
  - Dạng 64-bit: 1b dấu, 11b mũ, 52b định trị
  - Dạng 128-bit: 1b dấu, 15b mũ, 112b định trị

Parameter	Format		
	Binary32	Binary64	Binary128
Storage width (bits)	32	64	128
Exponent width (bits)	8	11	15
Exponent bias	127	1023	16383
Maximum exponent	127	1023	16383
Minimum exponent	−126	−1022	−16382
Approx normal number range (base 10)	$10^{-38}, 10^{+38}$	$10^{-308}, 10^{+308}$	$10^{-4932}, 10^{+4932}$
Trailing significand width (bits)*	23	52	112
Number of exponents	254	2046	32766
Number of fractions	2^{23}	2^{52}	2^{112}
Number of values	1.98×2^{31}	1.99×2^{63}	1.99×2^{128}
Smallest positive normal number	2^{-126}	2^{-1022}	2^{-16362}
Largest positive normal number	$2^{128} - 2^{104}$	$2^{1024} - 2^{971}$	$2^{16384} - 2^{16271}$
Smallest subnormal magnitude	2^{-149}	2^{-1074}	2^{-16494}

Bảng 10.3 IEEE 754 Thông số định dạng

* not including implied bit and not including sign bit

+

Ví dụ 1

Biểu diễn số thực X = 9.6875 về dạng số dấu chấm động theo chuẩn IEEE 754 dạng 32 bit

Giải:

$X = 9.6875_{10} = 1001.1011_2 = 1.0011011 \times 2^3$ Ta có:

- S = 0 vì đây là số dương
- E = e – 127 nên e = 127 + 3 = $130_{10} = 1000\ 0010_2$
- m = 001101100...00 (23 bit)
  Vậy:
  X = 0100 0001 0001 1011 0000 0000 0000 0000

Ví dụ 2

Có một số thực X có dạng biểu diễn nhị phân theo chuẩn IEEE 754 dạng 32 bit như sau: 1100 0001 0101 0110 0000 0000 0000 0000. Xác định giá trị thập phân của số thực đó.

Giải:

- S = 1 → X là số âm
- e = 1000 0010 = 130
- m = 10101100...00
- Vậy $X = (-1) \times 1.10101100 \ldots 00 \; x \; 2^{130-127} = -1.101011 \times 2^3 = -1101.011 = -13.375$

Ví dụ 3

Xác định giá trị thập phân của số thực X có dạng biểu diễn theo chuẩn IEEE 754 dạng 32 bit như sau:

0011 1111 1000 0000 0000 0000 0000 0000

Một số quy ước đặc biệt

- Nếu tất cả các bit của e đều bằng 0, các bit của m đều bằng 0, thì X = ± 0
- Nếu tất cả các bit của e đều bằng 1, các bit của m đều bằng 0, thì X = ± ∞
- Nếu tất cả các bit của e đều bằng 1, m có ít nhất một bit bằng 1, thì X không phải là số (not a number - NaN)

Các định dạng bổ sung

Extended Precision Formats

- Cung cấp các bit bổ sung trong số mũ (phạm vi mở rộng) và trong phần định trị (độ chính xác mở rộng)
- Giảm khả năng kết quả cuối cùng bị tồi đi do sai số làm tròn quá mức
- Giảm bớt khả năng tràn trung gian làm hủy bỏ phép tính có kết quả cuối cùng có thể biểu diễn được dưới định dạng cơ bản
- Có một số lợi ích của định dạng cơ bản rộng hơn mà không phải chịu chi phí thời gian khi muốn độ chính xác cao hơn

Extendable Precision Format

- Độ chính xác và phạm vi được xác định dưới sự kiểm soát của người dùng
- Có thể được sử dụng để tính toán trung gian nhưng chuẩn sẽ không có ràng buộc hoặc định dạng hoặc chiều dài

+

5 Các phép toán với số dấu chấm động

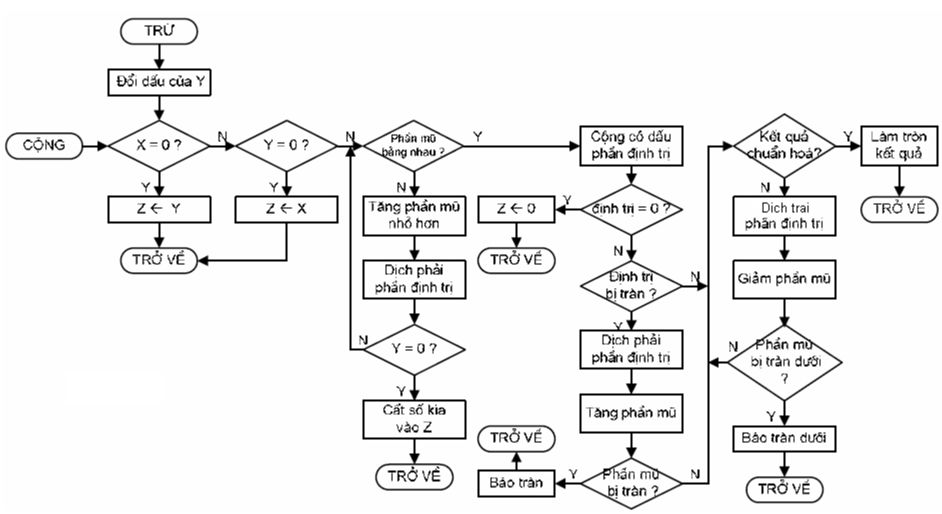
a. Phép toán cộng và trừ

Floating Point Numbers	Arithmetic Operations
$X = X_S \times B^{X_E}$ $Y = Y_S \times B^{Y_E}$	$\left.\begin{array}{l} X + Y = \left(X_s \times B^{X_E - Y_E} + Y_s\right) \times B^{Y_E} \\ X - Y = \left(X_s \times B^{X_E - Y_E} - Y_s\right) \times B^{Y_E} \end{array}\right\} X_E \le Y_E$ $X \times Y = \left(X_s \times Y_s\right) \times B^{X_E + Y_E}$ $\frac{X}{Y} = \left(\frac{X_s}{Y_s}\right) \times B^{X_E - Y_E}$

$X = 0.3 \times 10^2 = 30$
$Y = 0.2 \times 10^3 = 200$

$X + Y = (0.3 \times 10^{2-3} + 0.2) \times 10^3 = 0.23 \times 10^3 = 230$
$X - Y = (0.3 \times 10^{2-3} - 0.2) \times 10^3 = (-0.17) \times 10^3 = -170$
$X \times Y = (0.3 \times 0.2) \times 10^{2+3} = 0.06 \times 10^5 = 6000$
$X \div Y = (0.3 \div 0.2) \times 10^{2-3} = 1.5 \times 10^{-1} = 0.15$

Cộng và trừ dấu phẩy động



Bốn bước cơ bản của thuật toán cộng và trừ

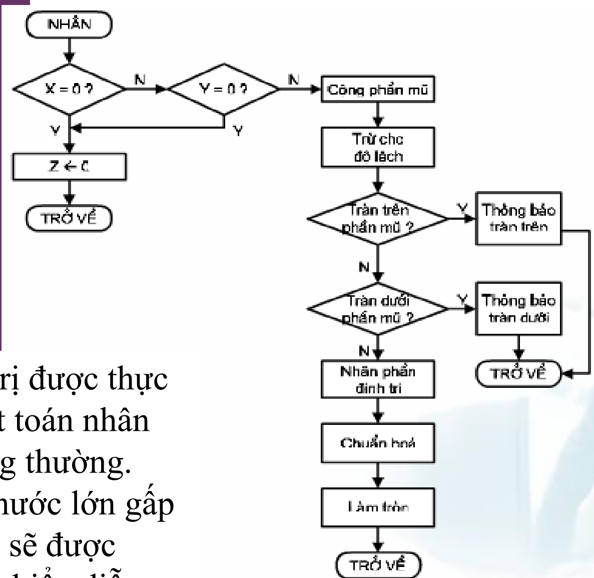
- Kiểm tra các số hạng có bằng 0 hay không
  - Nếu có thì gán kết quả dựa trên số còn lại.
- Hiệu chỉnh phần định trị
  - Sao cho 2 số có phần mũ giống nhau: tăng số mũ nhỏ và dịch phải phần định trị tương ứng (dịch phải để hạn chế sai số nếu có).
  - VD: $1.01 * 2^3 + 1.11 = 1.01 * 2^3 + 0.00111 * 2^3$
- Cộng hoặc trừ phần định trị
  - Nếu tràn thì dịch phải và tăng số mũ, nếu bị tràn số mũ thì báo lỗi tràn số.
- Chuẩn hóa kết quả
  - Dịch trái phần định trị để bit trái nhất (bit MSB) khác 0.
  - Tương ứng với việc giảm số mũ nên có thể dẫn đến hiện tượng tràn dưới số mũ.

Các khả năng tràn số

Phép toán cộng hoặc trừ có thể gây ra một số khả năng tràn như sau:

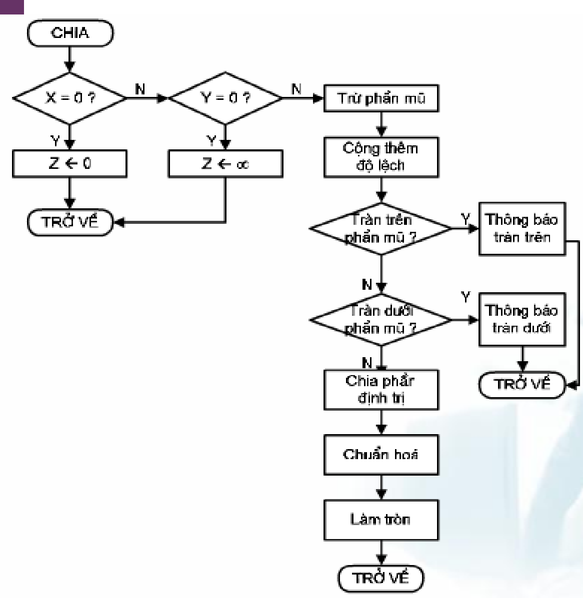
- Tràn trên số mũ (Exponent Overflow): mũ dương vượt ra khỏi giá trị cực đại của số mũ dương có thể.
- Tràn dưới số mũ (Exponent Underflow): mũ âm vượt ra khỏi giá trị cực đại của số mũ âm có thể.
- Tràn trên phần định trị (Mantissa Overflow): cộng hai phần định trị có cùng dấu, kết quả bị nhớ ra ngoài bit cao nhất.
- Tràn dưới phần định trị (Mantissa Underflow): Khi hiệu chỉnh phần định trị, các số bị mất ở bên phải phần định trị.

b. Phép nhân dấu chấm động



- Bộ nhân phần định trị được thực hiện giống như thuật toán nhân hai số nhị phân thông thường.
- Kết quả sẽ có kích thước lớn gấp đôi, tuy nhiên giá trị sẽ được chuẩn hóa theo dạng biểu diễn

c. Phép chia dấu chấm động



Nhân và chia dấu chấm động

Các bước cơ bản

1. Kiểm tra 0
2. Cộng/trừ số mũ (có thể xảy ra tràn → kiểm tra tràn mũ)
3. Nhân/chia các định trị
4. Chuẩn hóa
5. Làm tròn

Tổng kết

Chương 10

- ALU
- Biểu diễn số nguyên
  - Biểu diễn dấu-độ lớn
  - Biểu diễn bù 2
  - Mở rộng phạm vi
  - Biểu diễn dấu chấm tĩnh
- Biểu diễn dấu chấm động
  - Nguyên tắc
  - Chuẩn IEEE cho Biểu diễn dấu chấm động nhị phân

Số học máy tính

- Số học máy tính
  - phép đảo
  - Cộng và trừ
  - Nhân
  - Chia
- Số học dấu chấm động
  - Cộng và trừ
  - Nhân và chia
  - Độ chính xác
  - Chuẩn IEEE cho số học dấu chấm động nhị phân

Câu hỏi chương 10

1. Giải thích ngắn gọn về các biểu diễn: dấu-độ lớn, bù 2.
2. Cách xác định một số là âm hay dương trong các biểu diễn: dấu-độ lớn, bù 2.
3. Nguyên tắc mở rộng phạm vi biểu diễn số cho biểu diễn bù 2 là gì?
4. Cách đảo một số nguyên trong biểu diễn bù 2?
5. Phân biệt biểu diễn bù 2 của một số và bù 2 của một số?
6. Nếu coi 2 số bù 2 như là số nguyên không dấu khi thực hiện cộng, kết quả hiểu theo số bù 2 là chính xác. Điều này không đúng với phép nhân. Tại sao?
7. Bốn thành phần của một số trong biểu diễn dấu chấm động là gì?
8. Vì sao sử dụng biểu diễn bias cho số mũ của một số dấu chấm động?
9. Phân biệt tràn số mũ, và tràn định trị?
10. Các yếu tố cơ bản của phép cộng và trừ dấu chấm động là gì?