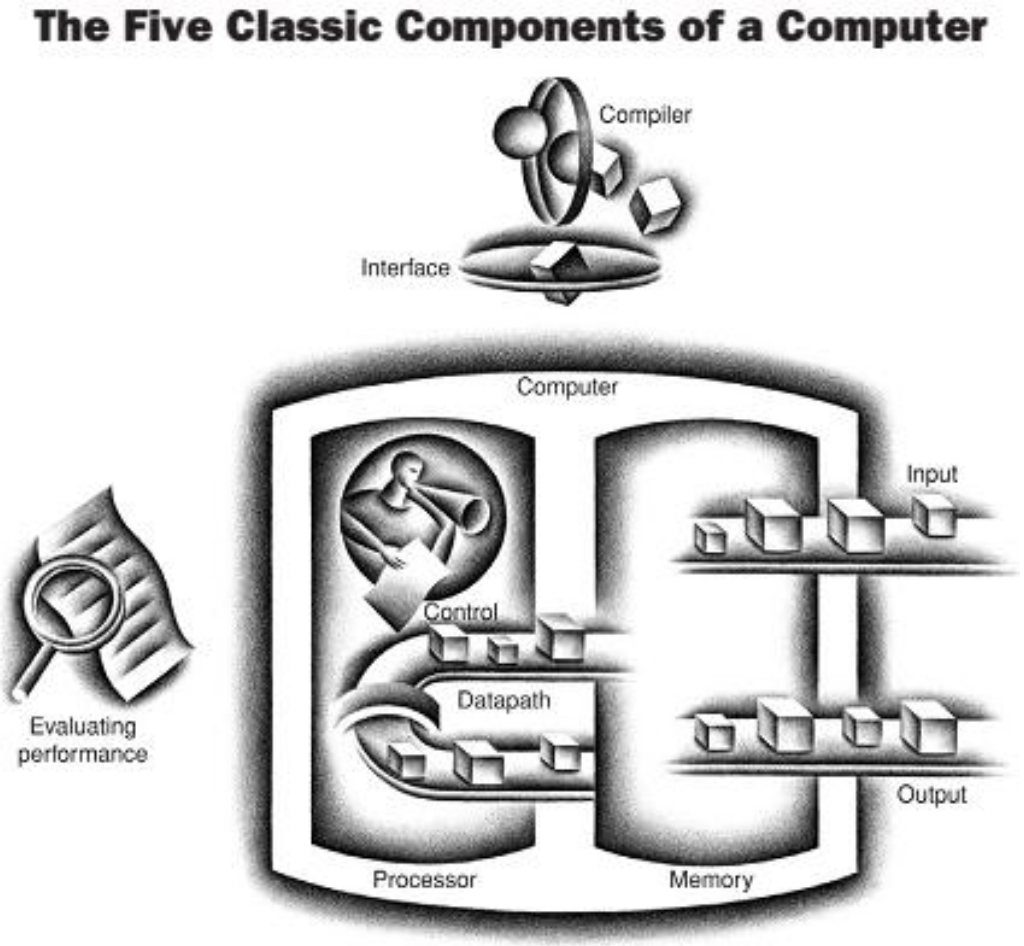
Chương 3

PHÉP TOÁN SỐ HỌC TRÊN MÁY TÍNH

PHÉP TOÁN SỐ HỌC TRÊN MÁY TÍNH

1. Giới thiệu
2. Phép cộng & Phép trừ
3. Phép Nhân
4. Phép chia
5. Số chấm động

PHÉP TOÁN SỐ HỌC TRÊN MÁY TÍNH



PHÉP TOÁN SỐ HỌC TRÊN MÁY TÍNH

1. **Giới thiệu**
2. **Phép cộng & Phép trừ**
3. **Phép Nhân**
4. **Phép chia**
5. **Số chấm động**

Giới thiệu

Các nội dung lưu trữ trong máy tính điều được biểu diễn ở dạng bit (giá trị của nó biểu diễn dưới dạng nhị phân, là 1 chuỗi các ký tự 0, 1). Trong chương 2, các số nguyên khi lưu trữ trong máy tính điều là các chuỗi nhị phân, hay các lệnh thực thi cũng lưu dưới dạng nhị phân. Vậy các dạng số khác thì biểu diễn như thế nào ?

Ví dụ:

- phần lẻ của số thực được biểu diễn, lưu trữ như thế nào?
- Điều gì sẽ xảy ra nếu kết quả của 1 phép toán sinh ra 1 số lớn hơn khả năng biểu diễn, hay lưu trữ ?
- Và một câu hỏi đặt ra là phép nhân và phép chia được phần cứng của máy tính thực hiện như thế nào?

PHÉP TOÁN SỐ HỌC TRÊN MÁY TÍNH

1. Giới thiệu
2. **Phép cộng & Phép trừ**
3. Phép Nhân
4. Phép chia
5. Số chấm động

Phép Cộng & Phép Trừ

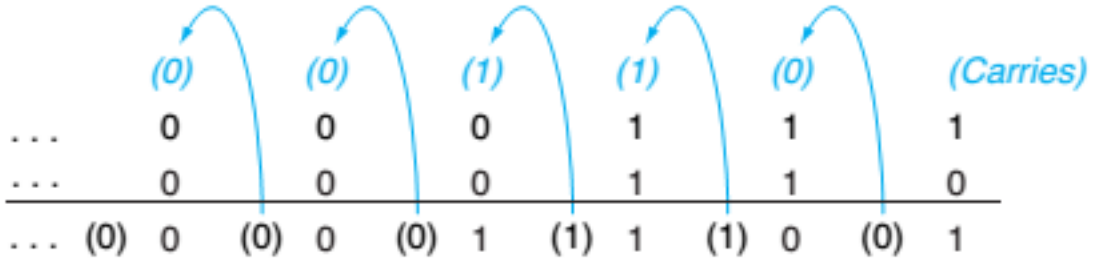
Phép cộng:

Ví dụ: $6_{10} + 7_{10}$ và $6_{10} - 7_{10}$

$$\begin{array}{ll} & 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0111_{two} = 7_{ten} \\ + & 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0110_{two} = 6_{ten} \\ \hline = & 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 1101_{two} = 13_{ten} \end{array}$$

Các bước thực hiện phép cộng trong số nhị phân: $a_n a_{n-1} \ldots a_1 a_0 + b_n b_{n-1} \ldots b_1 b_0$

1. Thực hiện phép cộng từ phải sang trái (hàng thứ 0 cho đến hàng n).
2. Số nhớ ở hàng cộng thứ i sẽ được cộng vào cho hàng cộng thứ i + 1.



Phép Cộng & Phép Trừ

Phép trừ: thực hiện phép trừ cho 2 số $a_n a_{n-1} \ldots a_1 a_0 - b_n b_{n-1} \ldots b_1 b_0$

1. Thực hiện phép trừ từ phải sang trái (hàng thứ 0 cho đến hàng n).
2. Số mượn ở hàng thứ i sẽ được cộng vào cho số trừ ở hàng từ i + 1.

Ví dụ: thực hiện phép toán: 7 – 6

Cách 1: Thực hiện phép trừ bình thường.

Cách 2: Chuyển số trừ sang dạng bù 2. Sau đó cộng với số bị trừ.

Subtracting 6_{ten} from 7_{ten} can be done directly:

$$\begin{array}{ll} & 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0111_{two} = 7_{ten} \\ - & 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0110_{two} = 6_{ten} \\ \hline = & 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0001_{two} = 1_{ten} \end{array}$$

or via addition using the two's complement representation of –6:

$$\begin{array}{ll} & 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0111_{two} = 7_{ten} \\ + & 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1010_{two} = -6_{ten} \\ \hline = & 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0001_{two} = 1_{ten} \end{array}$$

Phép Cộng & Phép Trừ

Số tràn

Các trường hợp tràn xảy ra khi thực hiện phép toán đối với số có dấu:

Tác vụ	Toán hạng A	Toán hạng B	Kết quả dẫn đến tràn
A + B	≥ 0	≥ 0	< 0
A + B	< 0	< 0	≥ 0
A – B	≥ 0	< 0	< 0
A – B	< 0	≥ 0	≥ 0

Một số điều lưu ý:

- Làm thế nào phát hiện việc tràn số trong biểu diễn số bù 2?
- Làm sao biết được khi nào thì tràn số không âm?

Phép Cộng & Phép Trừ

Xử lý tràn

- Các nhà thiết kế phần cứng làm sao nhận diện được các kết quả tràn số học, và cung cấp cơ chế bỏ qua tràn số học.
- Giải pháp xử lý trong kiến trúc MIPS cho 2 loại lệnh số học:
  - Lệnh cộng (add), cộng số tức thời (addi), trừ (sub) gây ra ngoại lệ tràn(ngắt quãng – interrupt).
  - Cộng số nguyên dương (**addu**), cộng số nguyên dương tức thời (**addiu**), và trừ số nguyên dương (**subu**) không gây ra ngoại lệ tràn.

Chú ý: kiến trúc MIPS xử lý tràn như là một ngoại lệ (exception), còn được gọi là ngắt quãng (interrupt). Một ngoại lệ hay một ngắt quãng là một phương thức được gọi mỗi khi sự kiện xảy ra.

PHÉP TOÁN SỐ HỌC TRÊN MÁY TÍNH

1. Giới thiệu
2. Phép cộng & Phép trừ
3. **Phép Nhân**
4. Phép chia
5. Số chấm động

Phép nhân

Ví dụ

Multiplicand		1000_{ten}
Multiplier	x	1001_{ten}
		1000
		0000
		0000
		1000
Product		1001000_{ten}

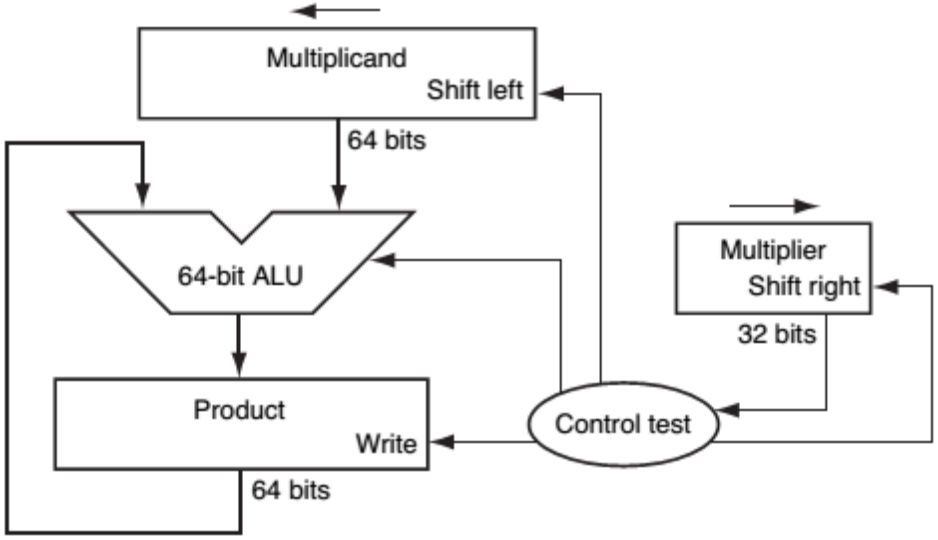
Multiplicand: số bị nhân
Multiplier: số nhân
Product: tích

Ví dụ trên ta chi lấy các con số ở dạng thập phân nhưng các chữ số điều là 0 và 1. Phép toán nhân trên số nhị phân cũng bắt buộc luôn dùng 0 và 1, và luôn luôn có 2 trường hợp:

1. Chép số bị nhân xuống vị trí thích hợp (***1*** ×multiplicand) nếu chữ số ở số nhân là 1.
2. Đặt số *0 (**0** ×multiplicand)* vào vị trí thích hợp nếu chữ số ở số nhân là 0.

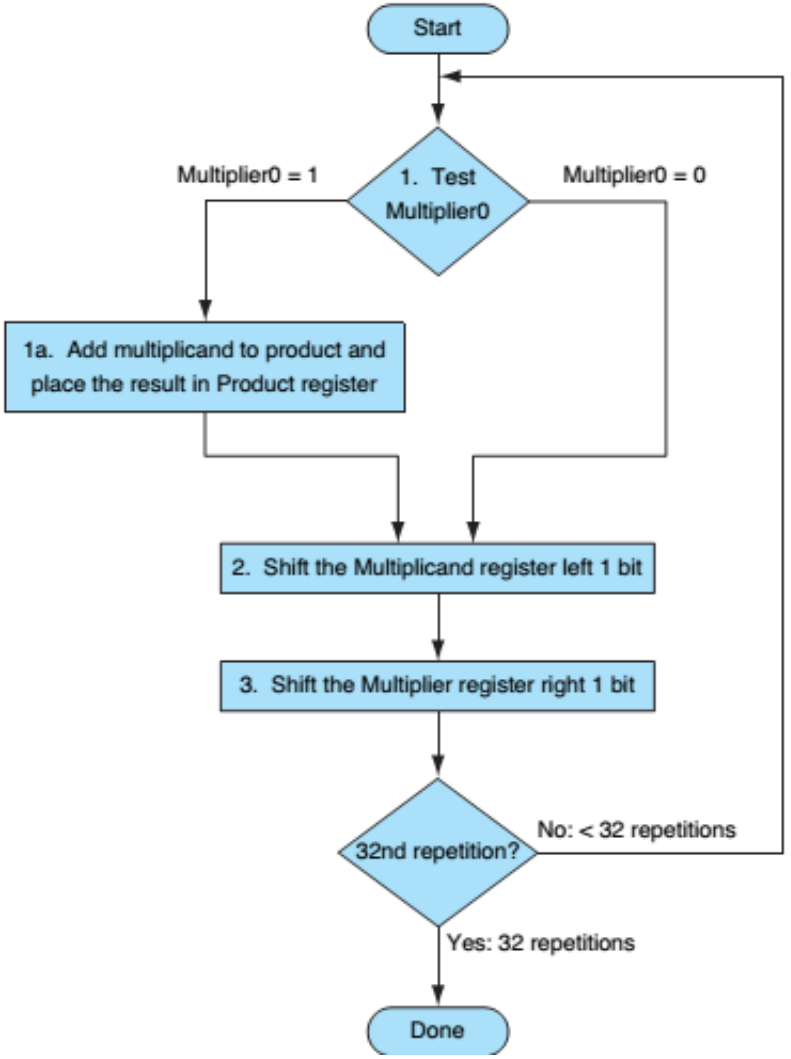
Phép Nhân

Giải thuật thực hiện phép nhân từng bước ở phần cứng



Hình 1: Sơ đồ các khối thực hiện phép nhân

Chú ý: khi thực hiện phép nhân cho giải thuật theo sơ đồ ta thấy có 3 bước, 3 bước này được lặp lại 32 lần. Mỗi bước được thực hiện bởi 1 chu kỳ xung clock CPU. Do đó giải thuật này yêu cầu 100 chu kỳ xung clock cho phép toán nhân 2 số 32 bit.



Hình 2: Sơ đồ giải thuật thực hiện phép nhân

Phép Nhân

Giải thuật thực hiện phép nhân từng bước ở phần cứng

Ví dụ: thực hiện phép toán nhân cho 2 số 4 bit sau: 2_{10} x 3_{10} .

Chuyển 2 số này sang dạng nhị phân: 0010_2 x 0011_2

Đáp án: bảng thực hiện từng bước giải thuật phép nhân 2 số

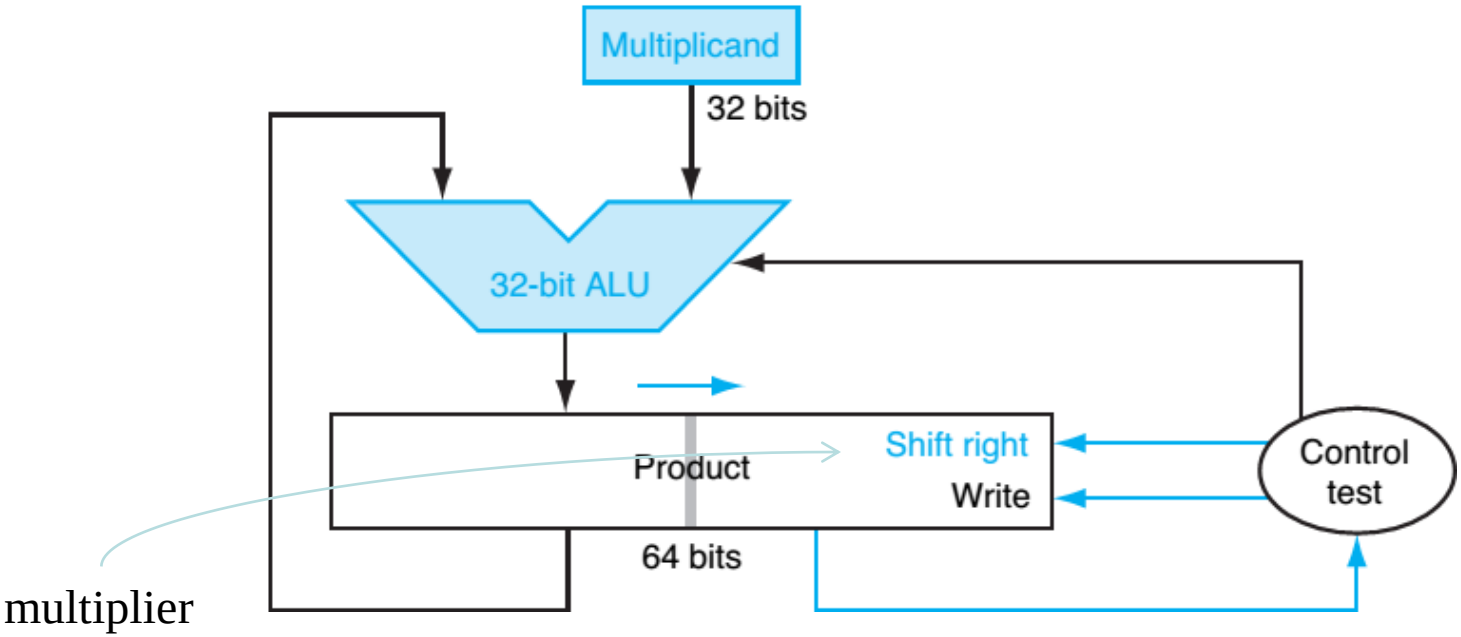
Phép Nhân

❑ **Đáp án:** bảng thực hiện từng bước giải thuật phép nhân 2 số

Iteration	Step	Multiplier	Multiplicand	Product
0	Initial values	0011	0000 0010	0000 0000
1	1a: 1 ⟹ Prod = Prod + Mcand	0011	0000 0010	0000 0010
	2: Shift left Multiplicand	0011	0000 0100	0000 0010
	3: Shift right Multiplier	0001	0000 0100	0000 0010
2	1a: 1 ⟹ Prod = Prod + Mcand	0001	0000 0100	0000 0110
	2: Shift left Multiplicand	0001	0000 1000	0000 0110
	3: Shift right Multiplier	0000	0000 1000	0000 0110
3	1: 0 ⟹ No operation	0000	0000 1000	0000 0110
	2: Shift left Multiplicand	0000	0001 0000	0000 0110
	3: Shift right Multiplier	0000	0001 0000	0000 0110
4	1: 0 ⟹ No operation	0000	0001 0000	0000 0110
	2: Shift left Multiplicand	0000	0010 0000	0000 0110
	3: Shift right Multiplier	0000	0010 0000	0000 0110

Phép Nhân

Giải thuật thực hiện phép nhân từng bước ở phần cứng



Hình 3 Sơ đồ của phép nhân cải tiến

- So với giải thuật trước đó thì thanh ghi số bị nhân, bộ ALU, thanh ghi số nhân tất cả điều 32 bits, chỉ có thanh ghi tích là khác – 64 bits;
- Chỉ sử dụng 1 chu kỳ xung clock để tính ra tích

Phép Nhân

Phép nhân có dấu

❖ Cách đơn giản để thực hiện phép toán nhân có dấu, ta chia làm 2 phần (1 bit dấu, 31 bit trị):

- ➢ Bit trọng số cao nhất là bit dấu, thực hiện phép xor trên 2 bit này để xác định dấu của tích.
- ➢ 31 bit còn lại xem như là phần trị và thực hiện việc tính toán bình thường.

❖ Giải thuật tính trong phép nhân thực hiện trong 31 lần lặp (bit biểu hiện dấu không tính trong phần này)

Phép Nhân

Phép nhân theo cách hiện thực tính nhanh

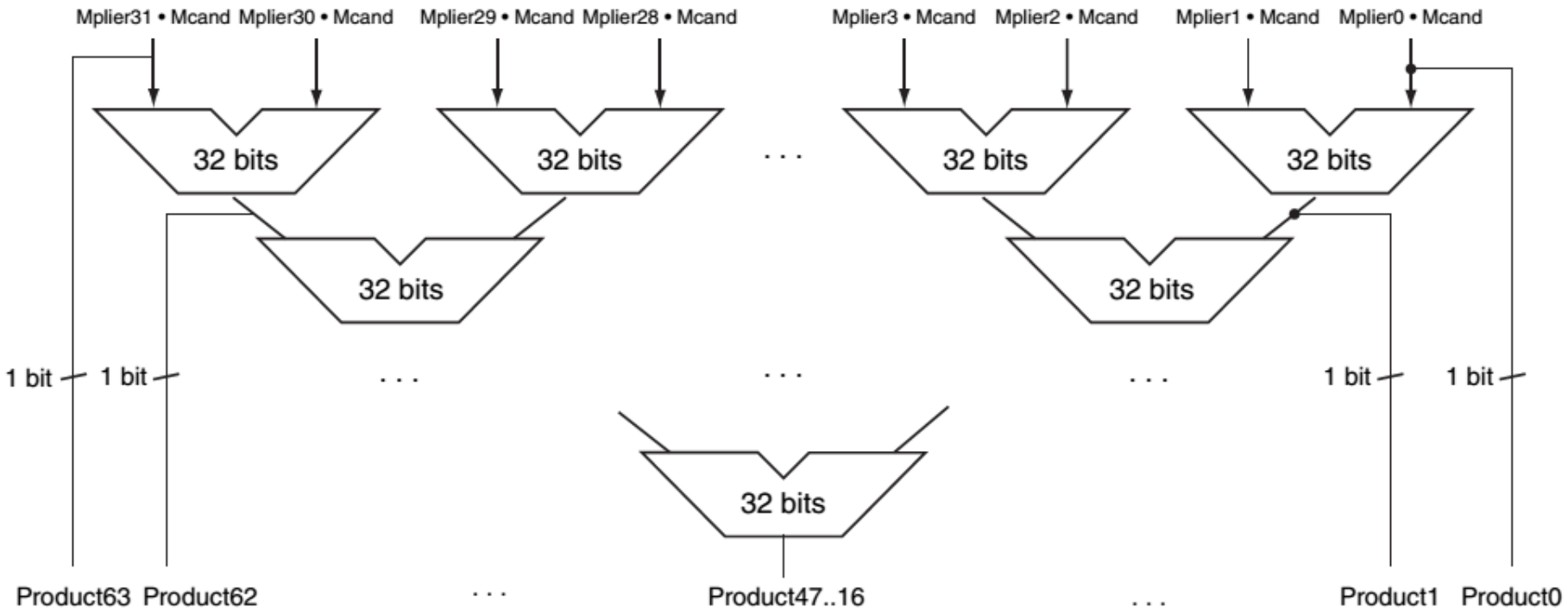


Fig.4 Sơ đồ hiện thực phép tính nhanh ở mức phần cứng

PHÉP TOÁN SỐ HỌC TRÊN MÁY TÍNH

1. Giới thiệu
2. Phép cộng & Phép trừ
3. Phép Nhân
4. **Phép chia**
5. Số chấm động

Phép Chia

- Ngược lại của phép nhân là phép chia.
- Trường hợp ngoại lệ – chia 0.

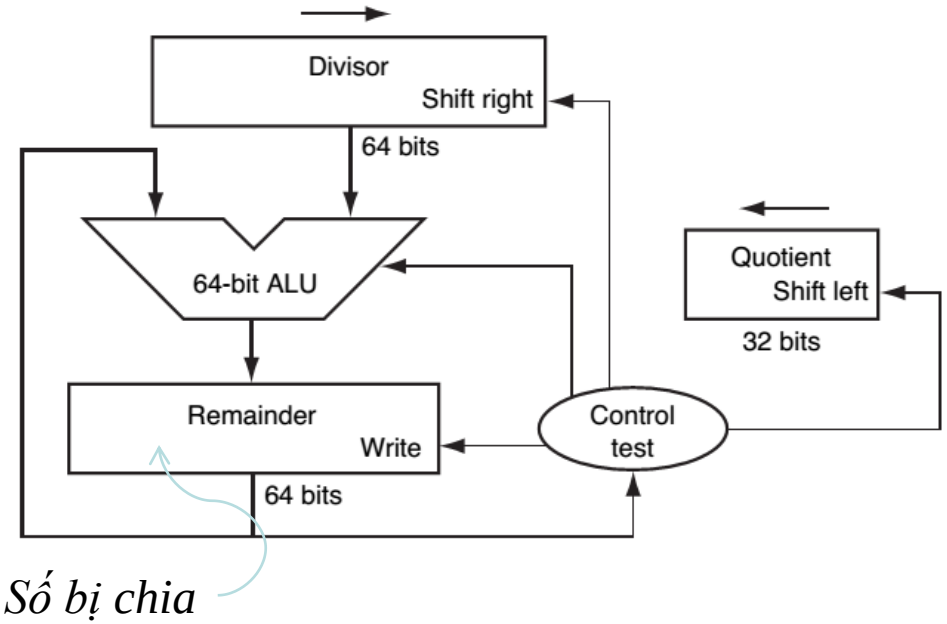
Ví dụ:

$$
\begin{array}{rll}
 & 1001_{ten} & \text{Quotient} \\
\text{Divisor } 1000_{ten} & \overline{)\,1001010_{ten}} & \text{Dividend} \\
 & \underline{-1000} & \\
 & 10 & \\
 & 101 & \\
 & 1010 & \\
 & \underline{-1000} & \\
 & 10_{ten} & \text{Remainder}
\end{array}
$$

Divisor: số chia
Quotient: số thương
Dividend: số phải chia
Remainder: số dư

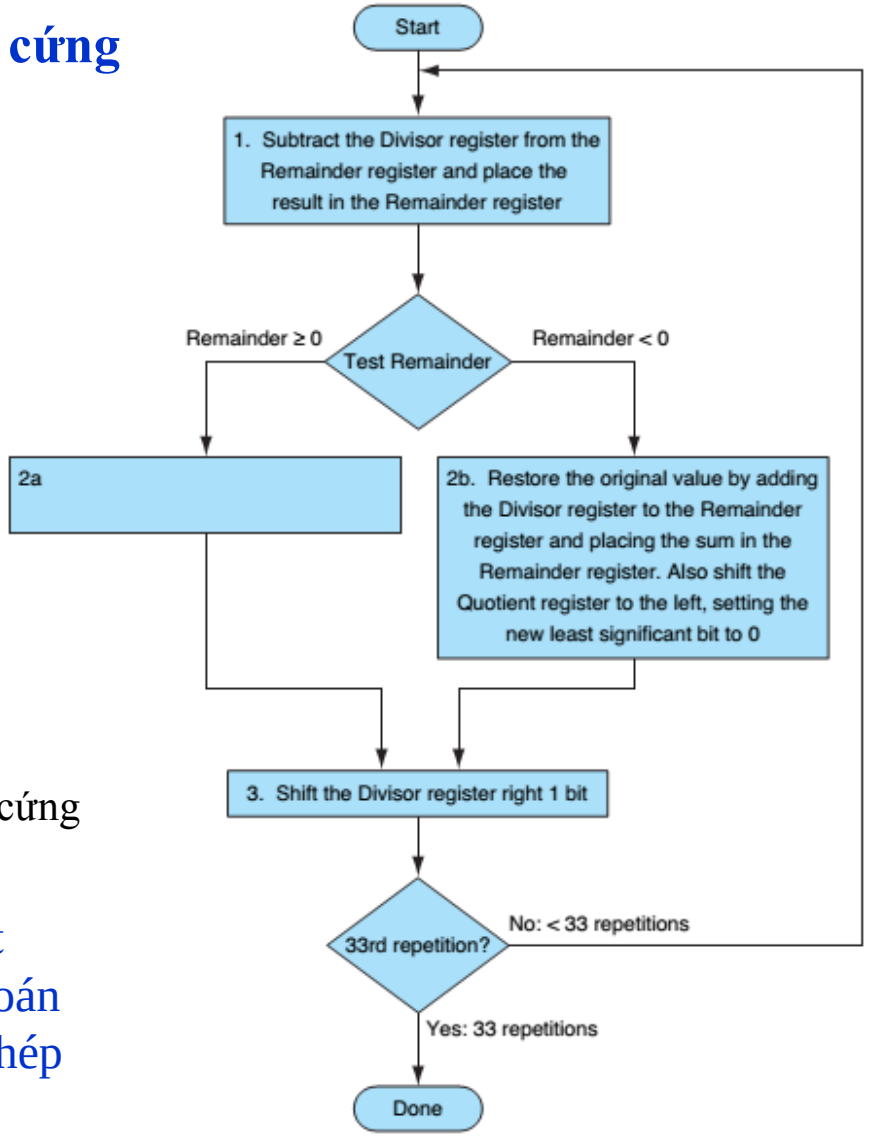
Phép Chia

Giải thuật thực hiện phép chia trên phần cứng



Hình 5: Sơ đồ các khối hiện thực phép chia ở mức phần cứng

Chú ý: Hai số chia và bị chia là số dương, do đó kết quả thương và số dư là không âm. Thực hiện phép toán trên số dương do đó thương và các toán hạng của phép chia có giá trị là 32 bit, bỏ qua các số có dấu.



Hình 6 giải thuật của phép chia

Phép Chia

Giải thuật thực hiện phép chia trên phần cứng

Ví dụ: thực hiệp phép chia cho 2 số 4 bit sau:

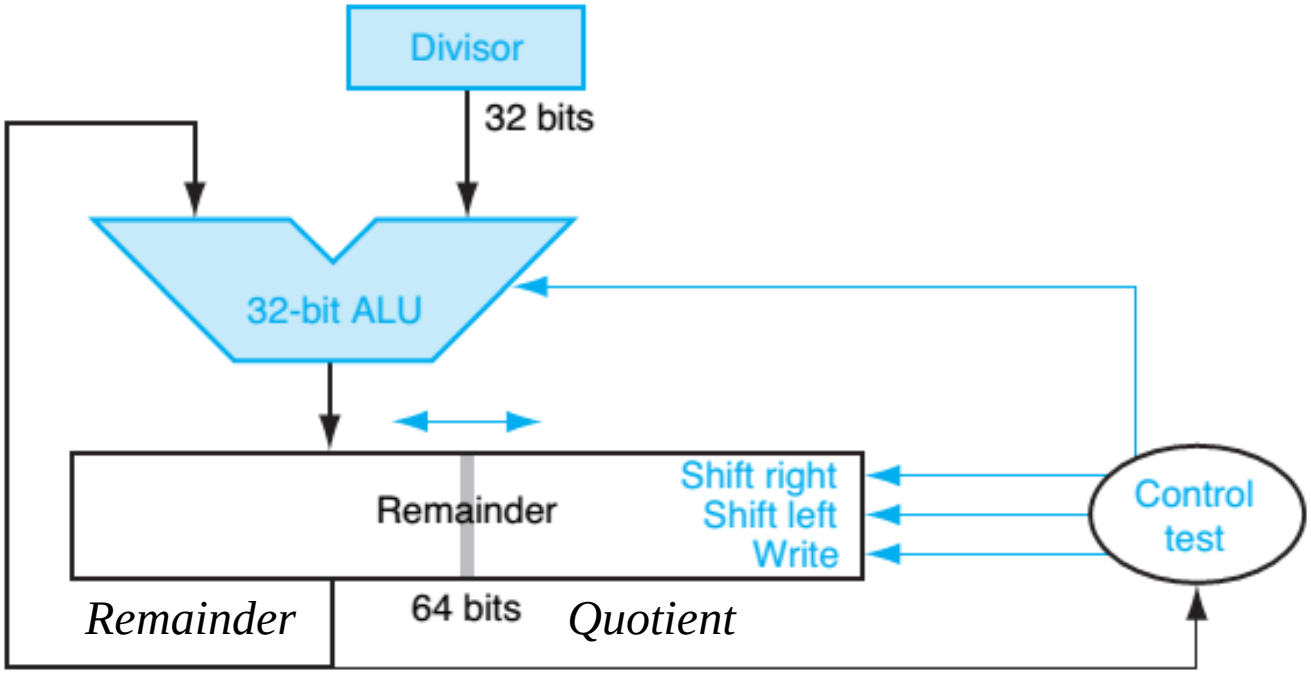
$7_{10} : 2_{10}$ **hay** $0111_2 : 0010_2$

Bảng thực hiện giải thuật phép chia theo từng bước

Iteration	Step	Quotient	Divisor	Remainder
0	Initial values	0000	0010 0000	0000 0111
1	1: Rem = Rem – Div	0000	0010 0000	①110 0111
	2b: Rem < 0 ⟹ +Div, sll Q, Q0 = 0	0000	0010 0000	0000 0111
	3: Shift Div right	0000	0001 0000	0000 0111
2	1: Rem = Rem – Div	0000	0001 0000	①111 0111
	2b: Rem < 0 ⟹ +Div, sll Q, Q0 = 0	0000	0001 0000	0000 0111
	3: Shift Div right	0000	0000 1000	0000 0111
3	1: Rem = Rem – Div	0000	0000 1000	①111 1111
	2b: Rem < 0 ⟹ +Div, sll Q, Q0 = 0	0000	0000 1000	0000 0111
	3: Shift Div right	0000	0000 0100	0000 0111
4	1: Rem = Rem – Div	0000	0000 0100	⓪000 0011
	2a: Rem ≥ 0 ⟹ sll Q, Q0 = 1	0001	0000 0100	0000 0011
	3: Shift Div right	0001	0000 0010	0000 0011
5	1: Rem = Rem – Div	0001	0000 0010	⓪000 0001
	2a: Rem ≥ 0 ⟹ sll Q, Q0 = 1	0011	0000 0010	0000 0001
	3: Shift Div right	0011	0000 0001	0000 0001

Phép Chia

Hiện thực phép chia nhanh



Hình 7: Hiện thực cách tính chia nhanh của các khối ở mức phần cứng

Phép Chia

Phép chia có dấu

❖ Ta xét bit trọng số cao nhất (bit dấu) nếu bit dấu của số bị chia và số chia trái dấu nhau thì bit dấu của thương tính bằng cách lấy phủ định bit dấu của thương.

❖ Điểm phức tập nhất trong phép chia có dấu là xác định bit dấu cho số dư. Các xác định bit dấu cho số dư bằng công thức sau:

Số bị chia = Thương x Số chia + Số dư

➔ **Số dư = Số bị chia – (Thương x Số chia)**

Ví dụ:

– 7 : +2 thì thương = -3, dư = –1

Kiểm tra kết quả:

–7 = –3 x 2 + (–1) = –6 – 1

Phép Chia

Phép chia trong MIPS

❖ Các khối thực hiện phép nhân, phép chia tương tư nhau như hình 3 (phép nhân) và hình 7 (phép chia).

❖ 1 thanh ghi 64 bit (chia 2 phần 32 bit trọng số cao và 32 bit trọng số thấp), 1 bộ ALU 32 bit thực hiện phép toán cộng, phép trừ.

❖ Sau khi thực hiện xong phép chia thì kết quả phép chia lưu trong thanh ghi 64 bit, trong đó 32 bit trọng số cao lưu trữ phần dư, 32 bit trọng số thấp lưu trữ phần thương.

❖ Để xử cho các số có dấu và số không dấu, MIPS có 2 lệnh: phép chia (**div**) , và phép chia không dấu (**divu**).

❖ Assembler cho phép lệnh chia sử dụng 3 thanh ghi: 2 lệnh **mflo, mfhi**. Kết quả 2 lệnh này được lưu trữ trong thanh ghi tổng quát.

PHÉP TOÁN SỐ HỌC TRÊN MÁY TÍNH

1. Giới thiệu
2. Phép cộng & Phép trừ
3. Phép Nhân
4. Phép chia
5. **Số chấm động**

Số Chấm Động

Định nghĩa:

Biểu diễn số thực:

$3.14159265\ldots_{ten}$ (pi)

$2.71828\ldots_{ten}$ (e)

0.000000001_{ten} or $1.0_{ten} \times 10^{-9}$ (seconds in a nanosecond)

$3{,}155{,}760{,}000_{ten}$ or $3.15576_{ten} \times 10^{9}$ (seconds in a typical century)

- Số thực biểu diễn dạng chuẩn gồm 2 phần: bên trái dấu chấm (phần nguyên – phần này chỉ có 1 ký số), và bên phải dấu chấm (phần lẻ).
- 1 số thực mà phần nguyên có số khác 0 gọi là số thực chuẩn.

  Ví dụ: 1.0_{ten} x 10^{-9}: số thực chuẩn

  0.1_{ten} x 10^{-8}: **không** phải số thực chuẩn

  10.0_{ten} x 10^{-10}: **không** phải số thực chuẩn
- Số nhị phân biểu diễn theo dạng chuẩn gọi là số thực dấu chấm động (**floating point**).

$$1.xxxxxxxxx_{two} \times 2^{yyyy}$$

Số thực dấu chấm động: Các phép toán trên máy tính thường thực hiện trên các số nhị phân.

Số Chấm Động

Biểu diễn số thực dấu chấm động

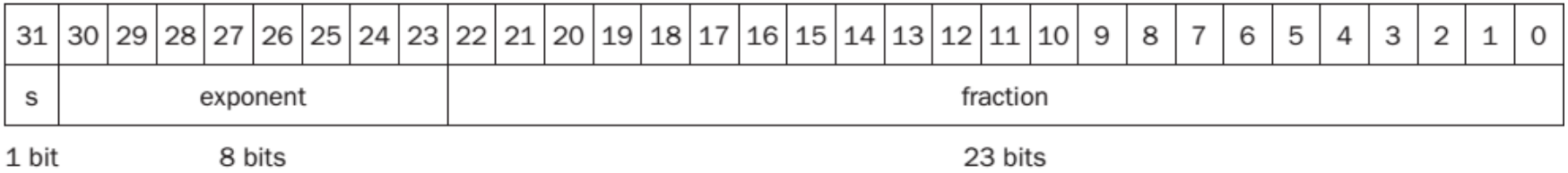
- Một Thiết kế biểu diễn số thực dấu chấm động phải thỏa mãn giữa phần kích thước phần lẽ và kích thước phần mũ.

  Cân bằng giữ độ chính xác và tầm trị có thể biểu diễn:

  - Tăng kích thước ở phần lẽ thì tăng độ chính xác.
  - Tăng kích thước phần mũ là tăng tầm trị biểu diễn.

- Kích thước của số dấu chấm động luôn bội số của kích thước 1 từ.

  Biểu diễn số thực dấu chấm động của MIPS (floating-point number):



Trong đó:

s là dấu của số thực dấu chấm động (1 nghĩa là âm, ngược lại 0 là dương)

Phần mũ (exponent) có kích thước là 8 bit (bao gồm luôn cả kích thước, và dấu của phần mũ)

Phần lẽ (fraction) là 1 số biểu diễn trong 23 bits

Tổng quát, số thực dấu chấm động biểu diễn như sau: $(-1)^S \times F \times 2^E$

Số Chấm Động

Biểu diễn số thực dấu chấm động

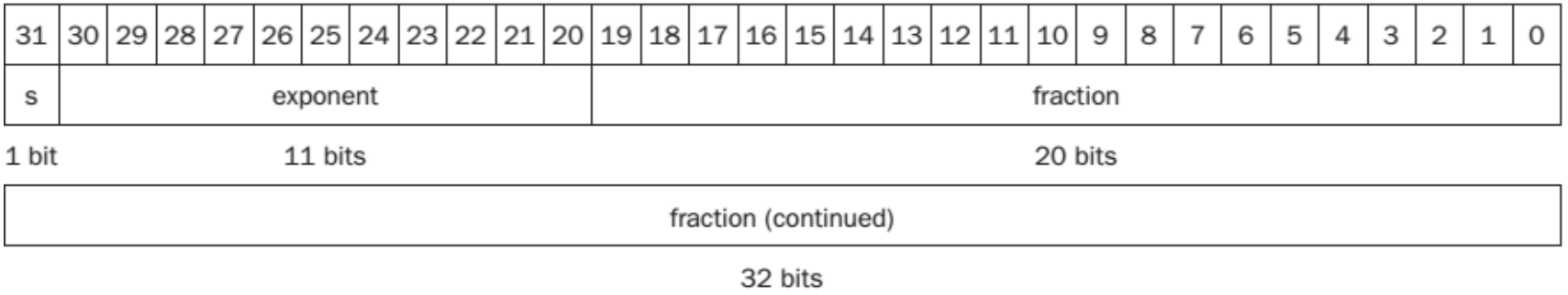
- **Tràn trên (Overflow):** trường hợp này xảy ra khi kích thước của số mũ lớn hơn kích thước giới hạn trên (số mũ dương).
- **Tràn dưới (Underflow):** trường hợp này xảy ra khi kích thước của số mũ nhỏ hơn kích thước giới hạn dưới (số mũ âm).

Hạn chế việc tràn trên, tràn cưới về số mũ. Trong lập trình ngôn ngữ C, có một loại số thuộc dạng double, các phép toán trên dạng số này thì có độ chính xác kép, còn cách biểu diễn trong slide trước gọi là độ chính xác đơn.

Độ chính xác kép (Double precision): một số thực dấu chấm động được biểu diễn ở dạng 64 bit.

Độ chính xác đơn (Single precision): một số thực dấu chấm động được biểu diễn ở dạng 32 bit..

Biểu diễn số thực độ chính xác kép:



Trong đó: **s** là dấu của số thực dấu chấm động (1 nghĩa là âm, 0 nghĩa là dương)

mũ (exponent) là số có kích thước ***11 bits*** (bao gồm biểu diễn luôn cả phần mũ âm)

phần lẻ (fraction) là số có độ dài ***52 bits***

Số Chấm Động

Biểu diễn số thực dấu chấm động

- Các định dạng trên không chỉ áp dụng cho MIPS, mà các biểu diễn trên là 1 phần của chuẩn biểu diễn số thực dấu chấm động IEEE754, chuẩn này được áp dụng cho hầu hết các máy tính được chế tạo từ năm 1980.

- Để tăng số bit có thể biểu diễn, các số này không lưu lại số 1 đứng trước dấu chấm. Các số 1 này được hiểu ngầm định là có trong cách biểu diễn theo chuẩn IEEE 754. Vậy số chính xác đơn thực chất lưu trữ 24 bit (trong đó có 1 bit giá trị 1 ở dạng ngầm định và 23 bit trị phần lẽ), và số chính xác kép thực chất lưu trữ trong 53 bit (trong đó có 1 bit giá trị 1 ở dạng ngầm định và 23 bit trị phần lẽ).

- Biểu diễn của số thực dấu chấm động khi thực hiện khôi phục từ dạng lưu trữ.

$$(-1)^S \times (1 + \text{Fraction}) \times 2^E$$

Từ công thức trên ta có công thức tổng quát (với Fraction là số bit đi từ trái qua phải trong phần lưu trữ, với s1 là bit có trong số cao nhất trong phần Fraction):

$$(-1)^S \times (1 + (s1 \times 2^{-1}) + (s2 \times 2^{-2}) + (s3 \times 2^{-3}) + (s4 \times 2^{-4}) + \ldots) \times 2^E$$

Số Chấm Động

Biểu diễn số thực dấu chấm động

- Phần mũ âm được biểu là ở dạng số bù hai hay các biểu diễn số âm khác thì bit có trọng số cao nhất trong phần mũ có giá trị là 1.

Ví dụ:

- $1.0_2 \times 2^{-1}$ số nảy biểu diễn số mũ âm – phần mũ giống như là số lớn nhất.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	.	.	.

- $1.0_{two} \times 2^{+1}$ số nảy biểu diễn số mũ dương – phần mũ giống như là số nhỏ nhất.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	.	.	.

Số Chấm Động

Biểu diễn số thực dấu chấm động

- Muốn biểu diễn số mũ âm nhất là 00 … 00_2 và số dương lớn nhất là 11 … 11_2.
- Biểu diễn số thiên vị được dùng trong trường hợp này. Biểu diễn số mũ bằng cách lấy số thiên vị cộng với số mũ bình thường, mục đích là biểu diễn các số không âm trong tầm trị mũ đi từ số âm nhất (00 … 00_2) đến số dương nhất (11 … 11_2).
- **IEEE 754** sử dụng số thiên vị 127 cho số thực chính xác đơn. Giả sử mũ của trừ 1 được tính thông qua số thiên vị (-1 + 127_{10}), hay 126_{10} = 0111 1110_2 , và số mũ +1 được biểu diễn (1+127), hay 128_{10} = 1000 0000_2 .
- Số thiên vị 1023 được sử dụng cho số thực chính xác kép.
- **Chú ý:** Số mũ thiên vị là giá trị biểu diễn phần mũ trong số thực dấu chấm động.

$$(-1)^S \times (1 + \text{Fraction}) \times 2^{(\text{Exponent} - \text{Bias})}$$

Dãy biểu diễn số độ chính xác đơn có tầm trị từ.

$\pm 1.0000\ 0000\ 0000\ 0000\ 0000\ 000_{two} \times 2^{-126}$

Đến số lớn nhất

$\pm 1.1111\ 1111\ 1111\ 1111\ 1111\ 111_{two} \times 2^{+127}$

Single precision		Double precision		Object represented
Exponent	Fraction	Exponent	Fraction	
0	0	0	0	0
0	Nonzero	0	Nonzero	± denormalized number
1–254	Anything	1–2046	Anything	± floating-point number
255	0	2047	0	± infinity
255	Nonzero	2047	Nonzero	NaN (Not a Number)

Hình 8: IEEE 754 mã hóa số thực dấu chấm động

Số Chấm Động

Biểu diễn số thực dấu chấm động

Ví dụ 1:

Show the IEEE 754 binary representation of the number -0.75_{ten} in single and double precision.

Số Chấm Động

Biểu diễn số thực dấu chấm động

Đáp án 1: The number -0.75_{ten} is also

$$-3/4_{ten} \text{ or } -3/2^2{}_{ten}$$

It is also represented by the binary fraction

$$-11_{two}/2^2{}_{ten} \text{ or } -0.11_{two}$$

In scientific notation, the value is

$$-0.11_{two} \times 2^0$$

and in normalized scientific notation, it is

$$-1.1_{two} \times 2^{-1}$$

The general representation for a single precision number is

$$(-1)^S \times (1 + \text{Fraction}) \times 2^{(\text{Exponent} - 127)}$$

Subtracting the bias 127 from the exponent of $-1.1_{two} \times 2^{-1}$ yields

$$(-1)^1 \times (1 + .1000\ 0000\ 0000\ 0000\ 0000\ 000_{two}) \times 2^{(126-127)}$$

The single precision binary representation of -0.75_{ten} is then

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	1	1	1	1	1	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

1 bit 8 bits 23 bits

Số Chấm Động

Biểu diễn số thực dấu chấm động

Đáp án 1:

The double precision representation is

$$(-1)^1 \times (1 + .1000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\text{two}) \times 2^{(1022-1023)}$$

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	1	1	1	1	1	1	1	1	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

1 bit | 11 bits | 20 bits

0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

32 bits

Số Chấm Động

Biểu diễn số thực dấu chấm động

Ví dụ 2: Chuyển đổi số nhị phân dấu chấm động đến số thập phân dấu chấm động

What decimal number is represented by this single precision float?

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	0	0	0	0	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	.	.	.

Trả lời 2:

The sign bit is 1, the exponent field contains 129, and the fraction field contains $1 \times 2^{-2} = 1/4$, or 0.25. Using the basic equation,

$$
\begin{aligned}
(-1)^S \times (1 + \text{Fraction}) \times 2^{(\text{Exponent} - \text{Bias})} &= (-1)^1 \times (1 + 0.25) \times 2^{(129-127)} \\
&= -1 \times 1.25 \times 2^2 \\
&= -1.25 \times 4 \\
&= -5.0
\end{aligned}
$$

Số Chấm Động

Phép toán cộng trên số thực dấu chấm động

Cộng số thực dấu chấm động. Cộng số thực dấu chấm động chuẩn trong hệ thập phân: $9.999_{10} \times 10^{1} + 1.610_{10} \times 10^{-1}$. Giả sử số thực dấu chấm động lưu trữ phần trị 4 chữ số, số mũ lưu trữ 2 chữ số.

Step 1. To be able to add these numbers properly, we must align the decimal point of the number that has the smaller exponent. Hence, we need a form of the smaller number, $1.610_{ten} \times 10^{-1}$, that matches the larger exponent. We obtain this by observing that there are multiple representations of an unnormalized floating-point number in scientific notation:

$$1.610_{ten} \times 10^{-1} = 0.1610_{ten} \times 10^{0} = 0.01610_{ten} \times 10^{1}$$

The number on the right is the version we desire, since its exponent matches the exponent of the larger number, $9.999_{ten} \times 10^{1}$. Thus, the first step shifts the significand of the smaller number to the right until its corrected exponent matches that of the larger number. But we can represent only four decimal digits so, after shifting, the number is really

$$0.016_{ten} \times 10^{1}$$

Số Chấm Động

Phép toán cộng trên số thực dấu chấm động

Step 2. Next comes the addition of the significands:

$$\begin{array}{rr} & 9.999_{ten} \\ + & 0.016_{ten} \\ \hline & 10.015_{ten} \end{array}$$

The sum is $10.015_{ten} \times 10^1$.

Step 3. This sum is not in normalized scientific notation, so we need to adjust it:

$$10.015_{ten} \times 10^1 = 1.0015_{ten} \times 10^2$$

Chú ý: kiểm tra phần mũ có bị tràn trên, tràn dưới ?

Step 4. Since we assumed that the significand can be only four digits long (excluding the sign), we must round the number. In our grammar school algorithm, the rules truncate the number if the digit to the right of the desired point is between 0 and 4 and add 1 to the digit if the number to the right is between 5 and 9. The number

$$1.0015_{ten} \times 10^2$$

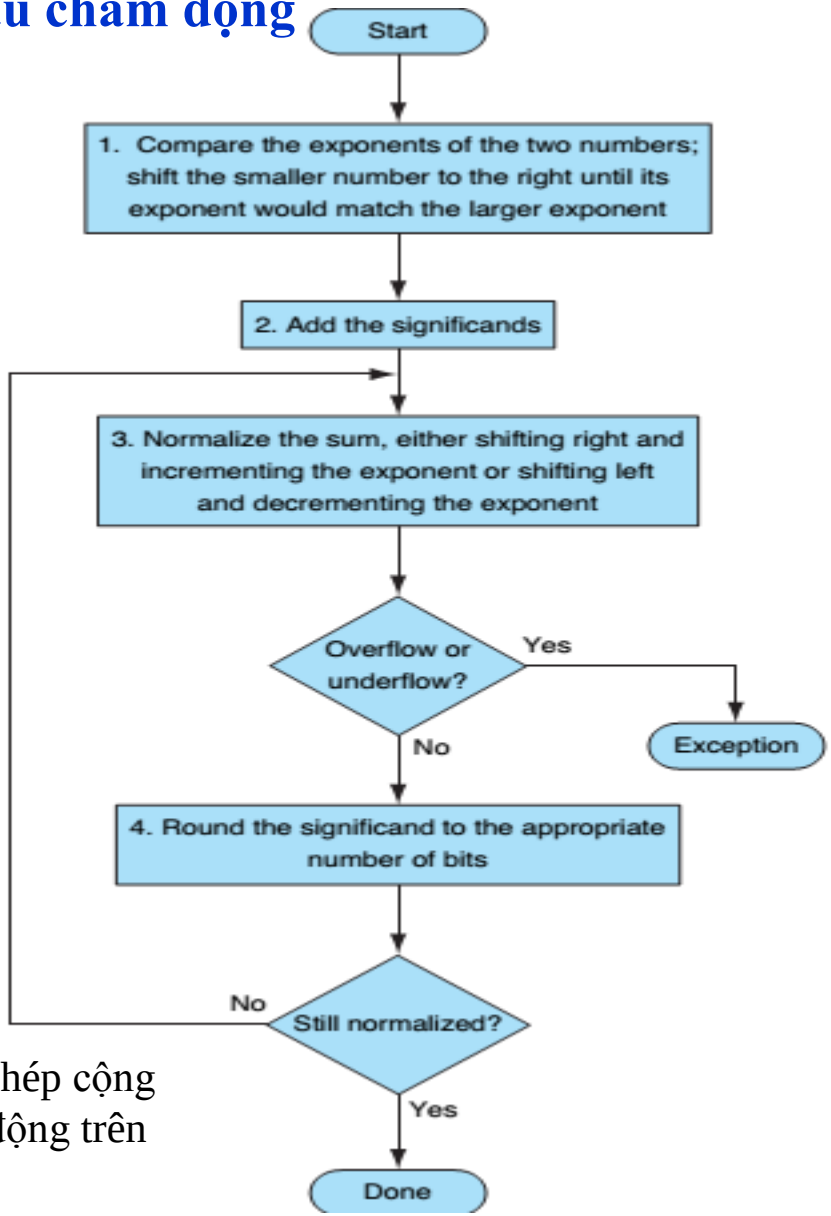
is rounded to four digits in the significand to

$$1.002_{ten} \times 10^2$$

Số Chấm Động

Phép toán cộng trên số thực dấu chấm động

Giải thuật thực hiện phép cộng trên số thực dấu chấm động trong hệ nhị phân



Hình 9. Giải thuật thực hiện phép cộng trên số thực dấu chấm động trên hệ số nhị phân

Số Chấm Động

Phép toán cộng trên số thực dấu chấm động

Ví dụ : Cộng 2 số thực dấu chấm động trong hệ nhị phân cho 2 số thập phân sau: 0.5_{10} và -0.4375_{10} theo giải thuật hình 9.

Đáp án:

Let's first look at the binary version of the two numbers in normalized scientific notation, assuming that we keep 4 bits of precision:

$$
\begin{aligned}
0.5_{ten} &= 1/2_{ten} &&= 1/2^{1}{}_{ten} \\
&= 0.1_{two} &&= 0.1_{two} \times 2^{0} &&= 1.000_{two} \times 2^{-1} \\
-0.4375_{ten} &= -7/16_{ten} &&= -7/2^{4}{}_{ten} \\
&= -0.0111_{two} &&= -0.0111_{two} \times 2^{0} &&= -1.110_{two} \times 2^{-2}
\end{aligned}
$$

Số Chấm Động

Phép toán cộng trên số thực dấu chấm động

Đáp án :

Now we follow the algorithm:

Step 1. The significand of the number with the lesser exponent ($-1.11_{two} \times 2^{-2}$) is shifted right until its exponent matches the larger number:

$$-1.110_{two} \times 2^{-2} = -0.111_{two} \times 2^{-1}$$

Step 2. Add the significands:

$$1.000_{two} \times 2^{-1} + (-0.111_{two} \times 2^{-1}) = 0.001_{two} \times 2^{-1}$$

Step 3. Normalize the sum, checking for overflow or underflow:

$$0.001_{two} \times 2^{-1} = 0.010_{two} \times 2^{-2} = 0.100_{two} \times 2^{-3} = 1.000_{two} \times 2^{-4}$$

Since $127 \geq -4 \geq -126$, there is no overflow or underflow. (The biased exponent would be $-4 + 127$, or 123, which is between 1 and 254, the smallest and largest unreserved biased exponents.)

Step 4. Round the sum:

$$1.000_{two} \times 2^{-4}$$

The sum already fits exactly in 4 bits, so there is no change to the bits due to rounding.

This sum is then

$$1.000_{two} \times 2^{-4} = 0.0001000_{two} = 0.0001_{two} = 1/2^{4}_{ten} = 1/16_{ten} = 0.0625_{ten}$$

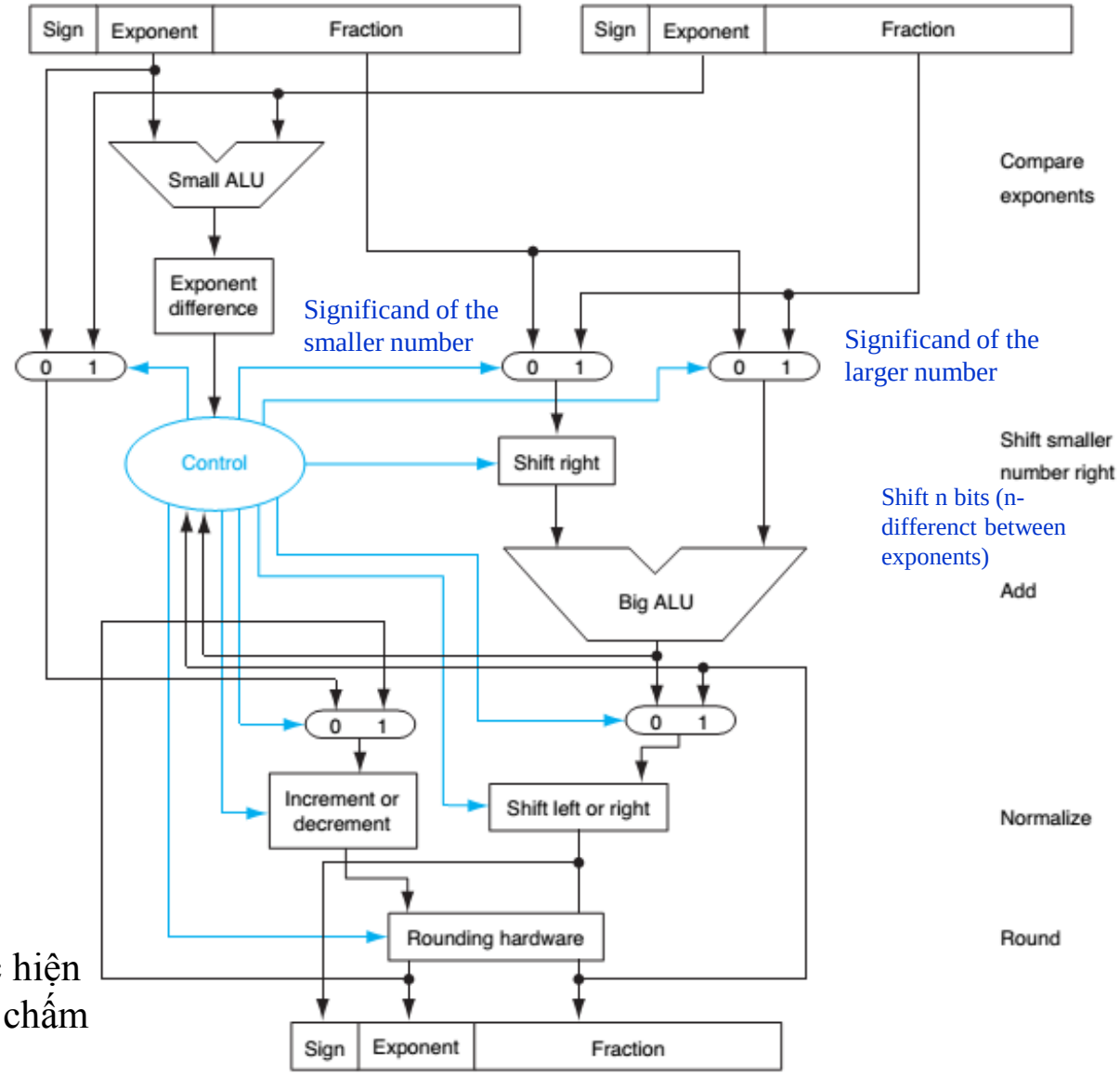
This sum is what we would expect from adding 0.5_{ten} to -0.4375_{ten}.

Số Chấm Động

Phép toán cộng trên số thực dấu chấm động

Kiến trúc phần cứng:

Chọn thành phần mũ lớn hơn



Hình 10: Sơ đồ khối cho việc thực hiện phép toán cộng số thực dấu chấm động trên hệ số nhị phân.

Số Chấm Động

Phép nhân trên số thực dấu chấm động

Ví dụ 1: Nhân số thực dấu chấm động. Nhân số thực dấu chấm động chuẩn trong hệ thập phân: $1.110_{10} \times 10^{10} * 9.200_{10} \times 10^{-5}$. Giả sử số thực dấu chấm động lưu trữ phần trị 4 chữ số, số mũ lưu trữ 2 chữ số.

Đáp án 1:

Step 1. Unlike addition, we calculate the exponent of the product by simply adding the exponents of the operands together:

$$\text{New exponent} = 10 + (-5) = 5$$

Let's do this with the biased exponents as well to make sure we obtain the same result: $10 + 127 = 137$, and $-5 + 127 = 122$, so

$$\text{New exponent} = 137 + 122 = 259$$

This result is too large for the 8-bit exponent field, so something is amiss! The problem is with the bias because we are adding the biases as well as the exponents:

$$\text{New exponent} = (10 + 127) + (-5 + 127) = (5 + 2 \times 127) = 259$$

Accordingly, to get the correct biased sum when we add biased numbers, we must subtract the bias from the sum:

$$\text{New exponent} = 137 + 122 - 127 = 259 - 127 = 132 = (5 + 127)$$

and 5 is indeed the exponent we calculated initially.

Số Chấm Động

Phép nhân trên số thực dấu chấm động

Step 2. Next comes the multiplication of the significands:

$$\begin{array}{r} 1.110_{ten} \\ \times\ 9.200_{ten} \\ \hline 0000 \\ 0000 \\ 2220 \\ 9990 \\ \hline 10212000_{ten} \end{array}$$

There are three digits to the right of the decimal point for each operand, so the decimal point is placed six digits from the right in the product significand:

$$10.212000_{ten}$$

Assuming that we can keep only three digits to the right of the decimal point, the product is 10.212×10^5.

Step 3. This product is unnormalized, so we need to normalize it:

$$10.212_{ten} \times 10^5 = 1.0212_{ten} \times 10^6$$

Chú ý: kiểm tra số mũ có bị tràn trên, tràn dưới ?

Số Chấm Động

Phép nhân trên số thực dấu chấm động

Step 4. We assumed that the significand is only four digits long (excluding the sign), so we must round the number. The number

$$1.0212_{ten} \times 10^6$$

is rounded to four digits in the significand to

$$1.021_{ten} \times 10^6$$

Step 5. The sign of the product depends on the signs of the original operands. If they are both the same, the sign is positive; otherwise, it's negative. Hence, the product is

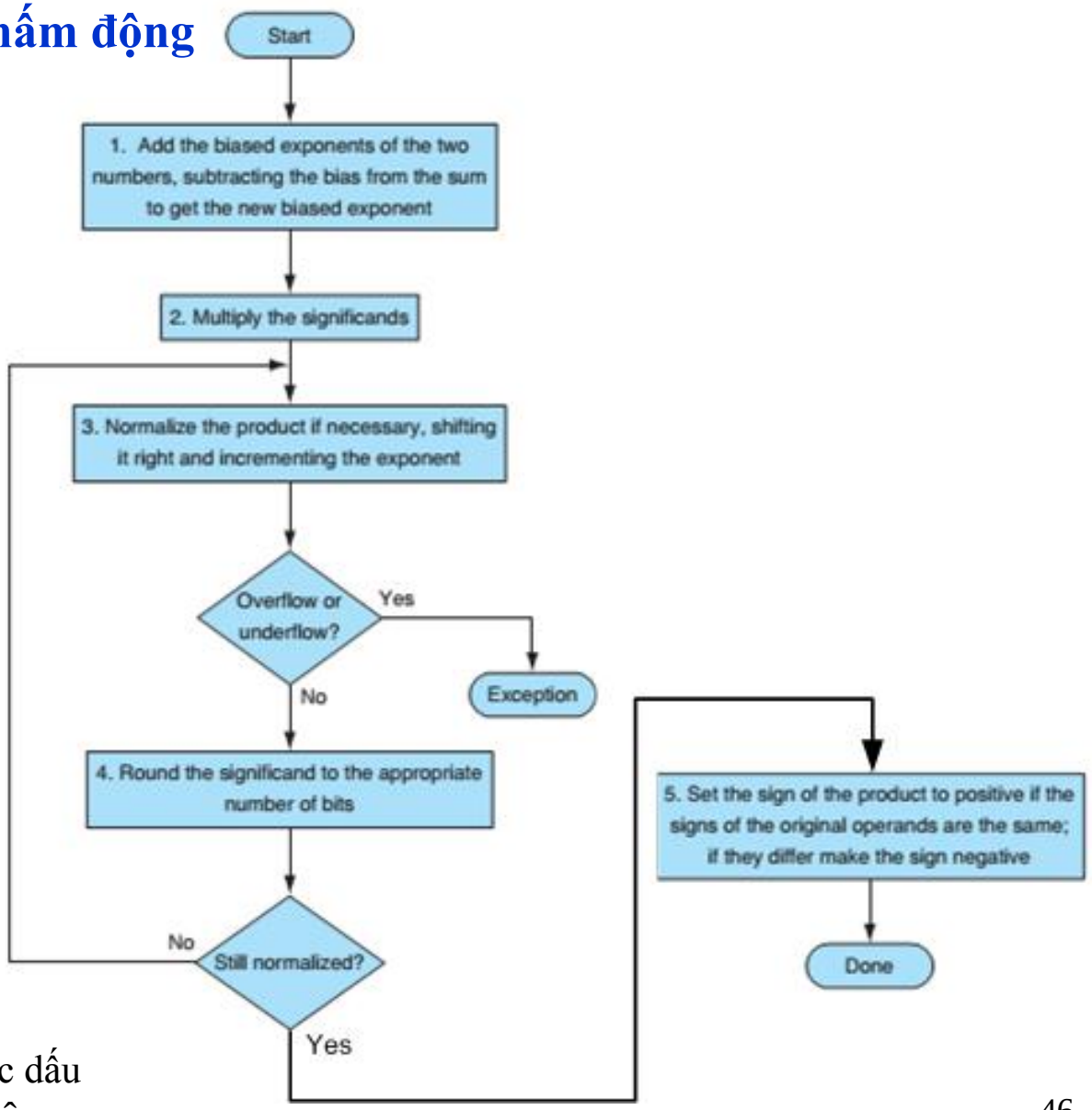
$$+1.021_{ten} \times 10^6$$

The sign of the sum in the addition algorithm was determined by addition of the significands, but in multiplication, the sign of the product is determined by the signs of the operands.

Số Chấm Động

Phép nhân trên số thực dấu chấm động

Giải thuật nhân số thực dấu chấm động trên hệ nhị phân có 5 bước giống như là ví dụ 1 trong phần này.



Hình 11: Giải thuật nhân số thực dấu chấm động trên hệ nhị phân

Số Chấm Động

Phép nhân trên số thực dấu chấm động

Ví dụ 2: nhân số thực dấu chấm động trên hệ nhị phân cho 2 số sau: 0.5_{10} và -0.4375_{10}.

Đáp án 2:

In binary, the task is multiplying $1.000_{two} \times 2^{-1}$ by $-1.110_{two} \times 2^{-2}$.

Step 1. Adding the exponents without bias:

$$-1 + (-2) = -3$$

or, using the biased representation:

$$(-1 + 127) + (-2 + 127) - 127 = (-1 - 2) + (127 + 127 - 127)$$
$$= -3 + 127 = 124$$

Step 2. Multiplying the significands:

$$\begin{array}{rl} & 1.000_{two} \\ \times & 1.110_{two} \\ \hline & 0000 \\ & 1000 \\ & 1000 \\ & 1000 \\ \hline & 1110000_{two} \end{array}$$

The product is $1.110000_{two} \times 2^{-3}$, but we need to keep it to 4 bits, so it is $1.110_{two} \times 2^{-3}$.

Số Chấm Động

Phép nhân trên số thực dấu chấm động

Ví dụ 2:

Step 3. Now we check the product to make sure it is normalized, and then check the exponent for overflow or underflow. The product is already normalized and, since $127 \geq -3 \geq -126$, there is no overflow or underflow. (Using the biased representation, $254 \geq 124 \geq 1$, so the exponent fits.)

Step 4. Rounding the product makes no change:

$$1.110_{two} \times 2^{-3}$$

Step 5. Since the signs of the original operands differ, make the sign of the product negative. Hence, the product is

$$-1.110_{two} \times 2^{-3}$$

Converting to decimal to check our results:

$$\begin{aligned} -1.110_{two} \times 2^{-3} &= -0.001110_{two} = -0.00111_{two} \\ &= -7/2^{5}{}_{ten} = -7/32_{ten} = -0.21875_{ten} \end{aligned}$$

The product of 0.5_{ten} and -0.4375_{ten} is indeed -0.21875_{ten}.

Câu hỏi

- Thực hiện phép nhân trên 3 thanh ghi cho các số 5 bit:
  - 4 x 5
  - 6 x 5
  - 2 x 9
- Thực hiện phép chia trên 3 thanh ghi cho các số 5 bit:
  - 12 : 5
  - 29 : 6
  - 25 : 4

Câu hỏi

- Chuyển các số thập phân sau sang dạng IEEE 754 – 32bit:
  - 2014
  - 0, 675
  - 10, 0125
- Chuyển các số thập phân sau sang dạng IEEE 754 – 64bit:
  - 2017
  - 0, 375
  - 10, 0425

Câu hỏi

- Cộng số thực dấu chấm động. Cộng số thực dấu chấm động chuẩn trong hệ thập phân, các số sau: (Giả sử số thực dấu chấm động lưu trữ phần lẻ 5 chữ số, số mũ lưu trữ 2 chữ số).
  - 10,5 + 32,25
  - 10 + 0.125
  - 1987 + 2001
- Nhân số thực dấu chấm động. Nhân số thực dấu chấm động chuẩn trong hệ thập phân, các số sau: (Giả sử số thực dấu chấm động lưu trữ phần lẻ 5 chữ số, số mũ lưu trữ 2 chữ số).
  - 10,5 x 32,25
  - 10 x 0.125
  - 1987 x 2001