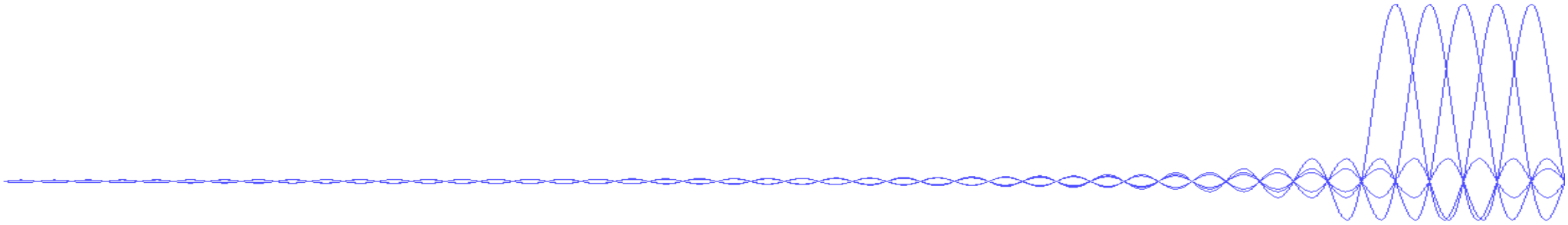
COMPUTER ENGINEERING

KIẾN TRÚC MÁY TÍNH

UIT
TRƯỜNG ĐẠI HỌC
CÔNG NGHỆ THÔNG TIN

PHÉP TOÁN SỐ HỌC TRÊN MÁY TÍNH



PHÉP TOÁN SỐ HỌC TRÊN MÁY TÍNH

Mục tiêu:

Hiểu các phép toán số học trên số nguyên và số thực dấu chấm động trong máy tính.

- **Với số nguyên:**
  - ✓ **Hiểu các phép toán cộng, trừ, nhân và chia**
  - ✓ **Cách thiết kế mạch nhân và chia**
- **Với số thực dấu chấm động:**
  - ✓ **Hiểu các phép toán cộng, trừ và nhân**
  - ✓ **Cách thiết kế mạch nhân**

Slide được dịch và các hình được lấy từ sách tham khảo:

Computer Organization and Design: The Hardware/Software Interface, Patterson, D. A., and J. L. Hennessy, Morgan Kaufman, Revised Fourth Edition, 2011.

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**

COMPUTER ENGINEERING

Giới thiệu

Các nội dung lưu trữ trong máy tính đều được biểu diễn ở dạng bit (hay dưới dạng nhị phân, là một chuỗi các ký tự 0, 1).

Trong chương 2, các số nguyên khi lưu trữ trong máy tính đều là các chuỗi nhị phân, hay các lệnh thực thi cũng phải lưu dưới dạng nhị phân. Vậy các dạng số khác thì biểu diễn như thế nào?

Ví dụ:

- ✓ Phân số và các số thực sẽ được biểu diễn và lưu trữ thế nào trong máy tính?
- ✓ Điều gì sẽ xảy ra nếu kết quả của một phép toán sinh ra một 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

COMPUTER ENGINEERING

Phép Cộng & Phép Trừ

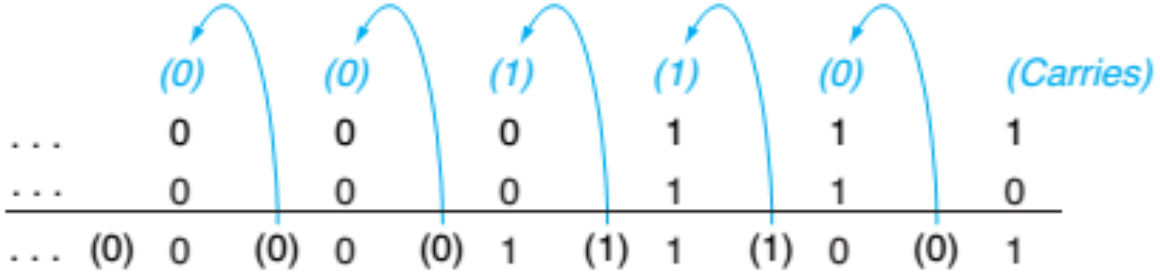
Phép cộng:

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.



COMPUTER ENGINEERING

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

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

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

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

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

Phép Cộng & Phép Trừ

Overflow (Tràn số)

Trong phép cộng và trừ, điều quan trọng cần lưu ý là phép toán có bị tràn hay không.

Hai trường hợp liên quan:

- Đối với số không dấu (Unsigned number)
- Đối với số có dấu (Signed number)

Phép Cộng & Phép Trừ

Xử lý tràn

❖ Các nhà thiết kế phần cứng phải cung cấp một cách để bỏ qua tràn hoặc phát hiện tràn trong các trường hợp cần thiết.

❖ Trong kiến trúc MIPS, mỗi lệnh thường có hai dạng lệnh tương ứng với xét overflow hay bỏ qua overflow:

- ✓ *Lệnh cộng (add), cộng số tức thời (addi), trừ (sub) là các lệnh có xét overflow, tức sẽ báo lỗi và phát ra một ngoại lệ (exception) nếu kết quả bị tràn.*
- ✓ *Lệnh cộng không dấu (addu), cộng số tức thời không dấu (addiu), và trừ không dấu (subu) không gây ra ngoại lệ tràn.*

Khi một chương trình đang thực thi, nếu bị tác động đột ngột (lỗi hoặc phải thi hành một tác vụ khác, ...), buộc phải dừng luồng chương trình đang chạy này và gọi đến một chương trình không định thời trước đó thì được gọi là một ***"interrupt"*** *hay một* ***"exception".***

Lưu ý: Trong một số hệ thống máy tính, thuật ngữ 'interrupt' được sử dụng như exception, nhưng ở một số hệ thống thì có sự phân biệt hai thuật ngữ này

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

COMPUTER ENGINEERING

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

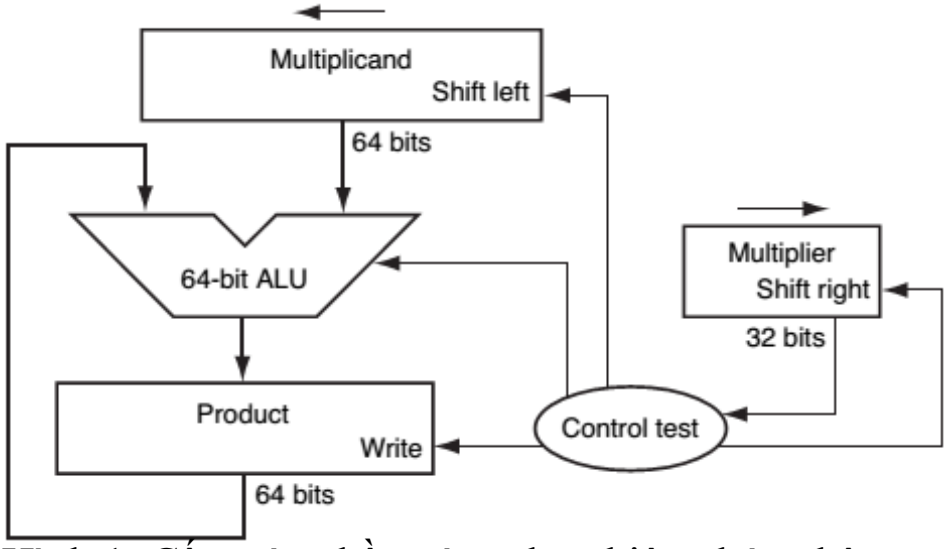
Ví dụ trên là nhân hai số đang ở dạng thập phân, nhưng các chữ số đều là 0 và 1. Phép nhân trên hai số nhị phân cũng tương tự, và luôn luôn có 2 trường hợp:

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

COMPUTER ENGINEERING

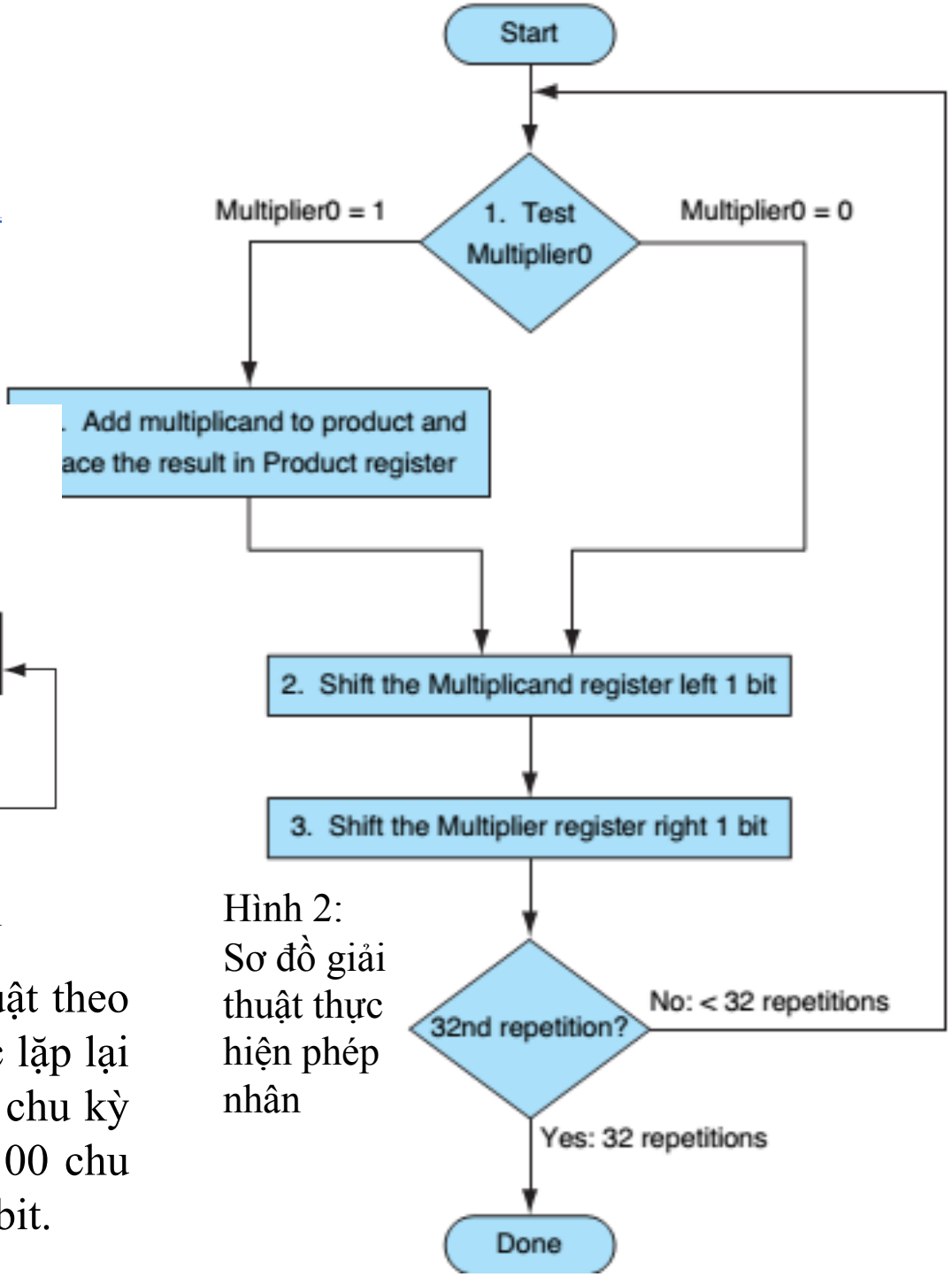
Phép nhân

Giải thuật thực hiện phép nhân theo cấu trúc phần cứng 3 thanh ghi (cho hai số 32 bit)



Hình 1: Cấu trúc phần cứng thực hiện phép nhân

Chú ý: khi thực hiện phép nhân cho giải thuật theo sơ đồ, ta thấy có 3 bước và 3 bước này được lặp lại 32 lần. Nếu mỗi bước được thực hiện bởi 1 chu kỳ xung clock thì giải thuật này yêu cầu gần 100 chu kỳ xung clock cho phép toán nhân hai số 32 bit.



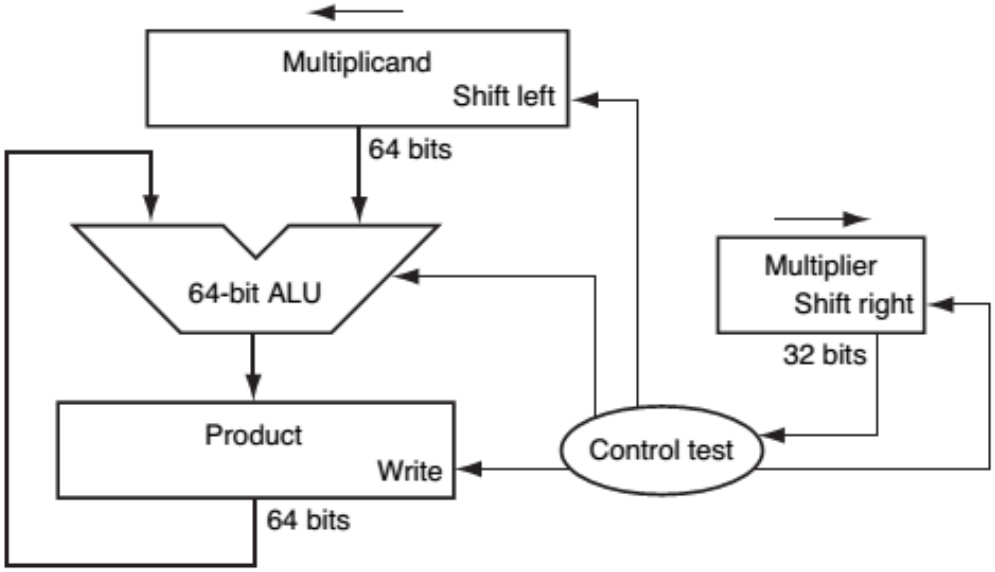
Hình 2: Sơ đồ giải thuật thực hiện phép nhân

COMPUTER ENGINEERING

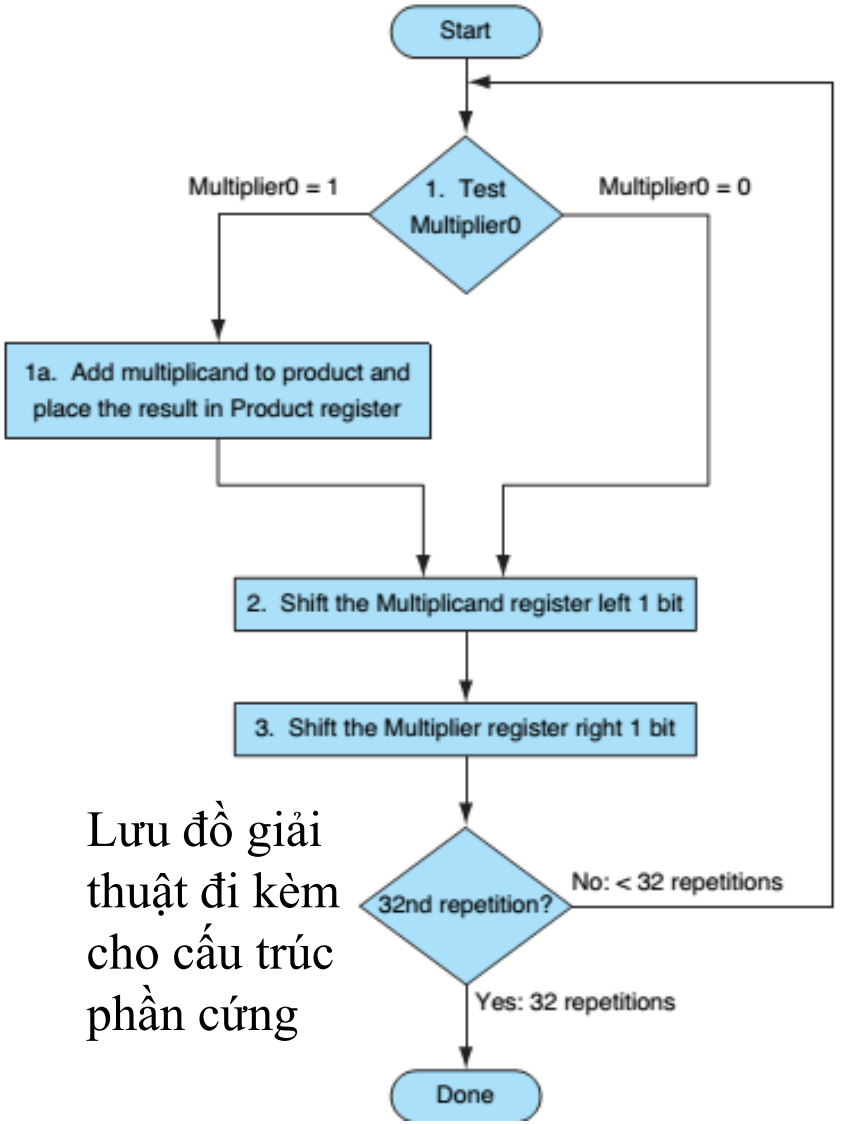
Ví dụ cho phép nhân (3 ví dụ)

Ví dụ 1:

Thực hiện phép nhân $2_{(10)}$ x $3_{(10)}$ (sử dụng số 4 bit không dấu) theo cấu trúc phần cứng như hình



+



Lưu đồ giải thuật đi kèm cho cấu trúc phần cứng

Ví dụ 1:

$2_{(10)}$ x $3_{(10)}$ = ?

$2_{(10)}$ = 0010 (multiplicand)

$3_{(10)}$ = 0011 (multiplier)

Cấu trúc phần cứng như hình vẽ là nhân 2 số 32 bits, kết quả là số 64 bits,

Có: thanh ghi multiplicand 64 bits

thanh ghi multiplier là 32 bits

thanh ghi product là 64 bits

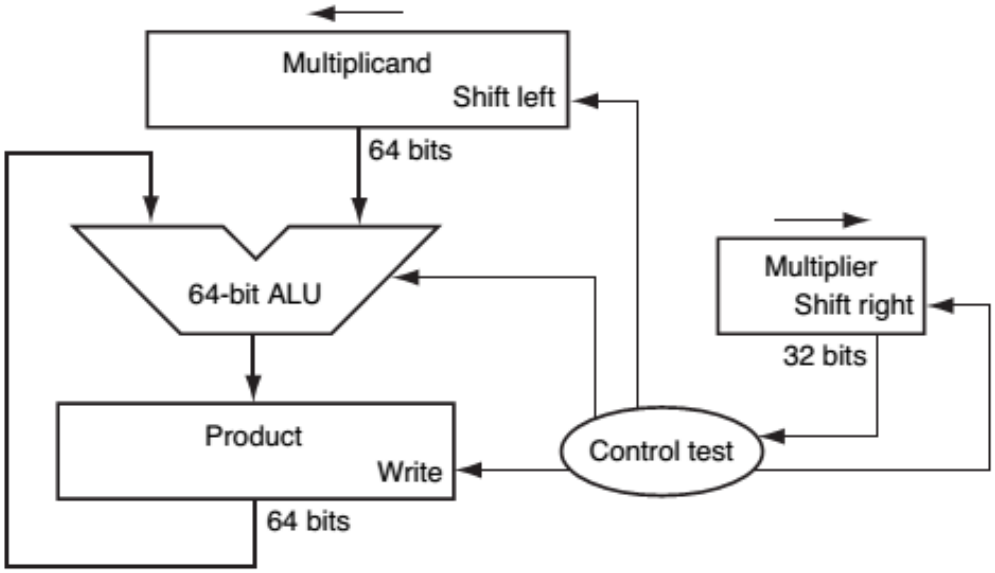
Ví dụ 1 yêu cầu nhân 2 số 4 bits không dấu, sử dụng cấu trúc phần cứng tương tự như hình, vậy kết quả phải là số 8 bits

=> thanh ghi multiplicand 8 bits (giá trị khởi tao 0000 0010)

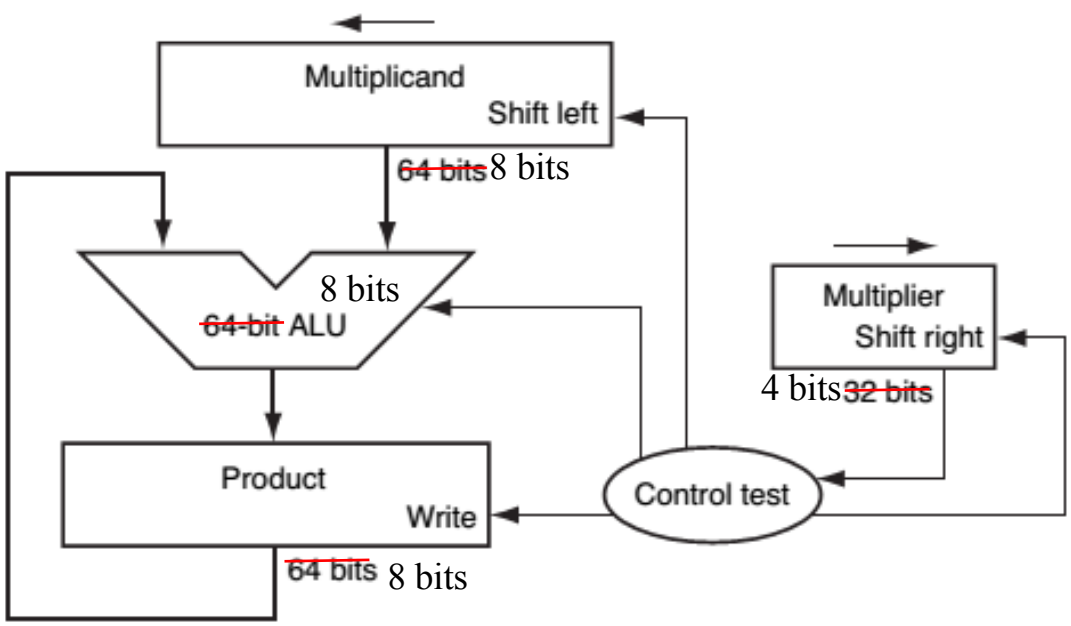
thanh ghi multiplier là 4 bits (giá trị khởi tạo 0011)

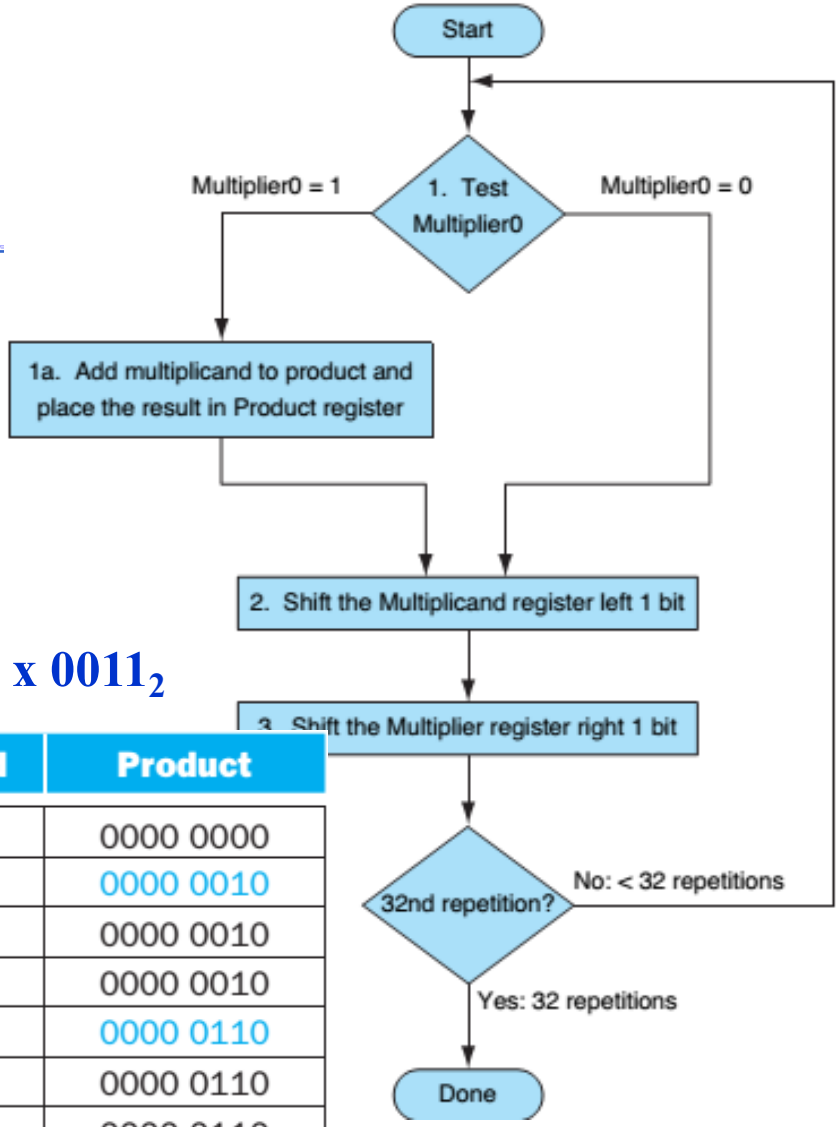
thanh ghi product là 8 bits (giá trị khởi tạo 0000 0000)

Iteration	Step	Multiplier	Multiplicand	Product
0	Khởi tạo	0011	0000 0010	0000 0000



- Sau khi khởi tạo xong. Mỗi vòng lặp (interation) sẽ gồm 3 bước:
  - B1. Kiểm tra bit 0 của multiplier xem có bằng 1 hay không; nếu bằng 1 thì product = product + multiplicand; nếu bằng 0, không làm gì cả
  - B2. Dịch trái Multiplicand 1 bit
  - B3. Dịch phải Multiplier 1 bit
- Số vòng lặp cho giải thuật này đúng bằng số bit dùng biểu diễn (ví dụ 1 yêu cầu dùng số 4 bit, thì có 4 vòng lặp)
- Sau khi kết thúc số vòng lặp, giá trị trong thanh ghi product chính là kết quả phép nhân





Bảng thực hiện từng bước giải thuật phép nhân 2 số: 0010_2 x 0011_2

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

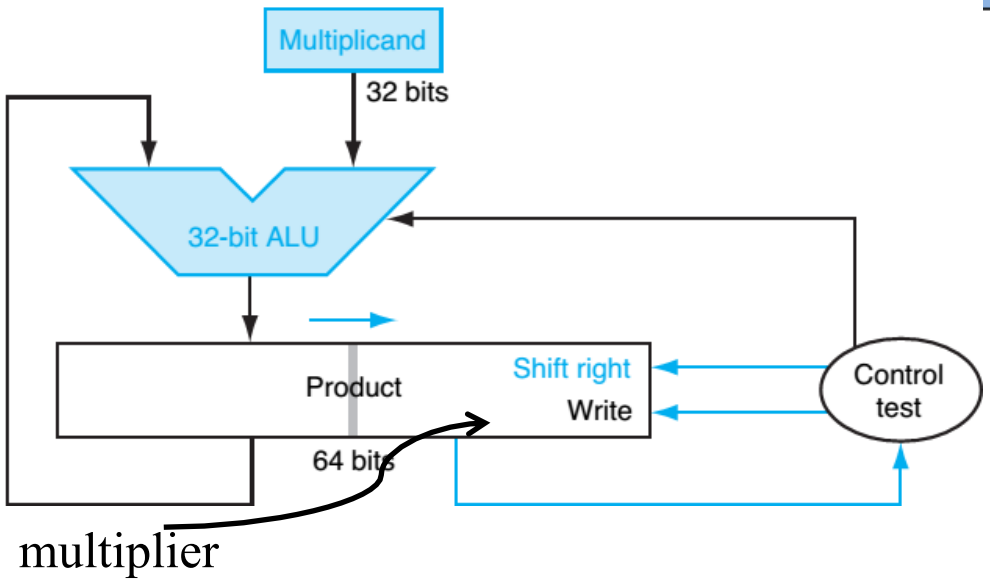
Phép Nhân

Lần lặp	Bước	Số nhân	Số bị nhân	Tích
0				
1				
2				
3				
4				

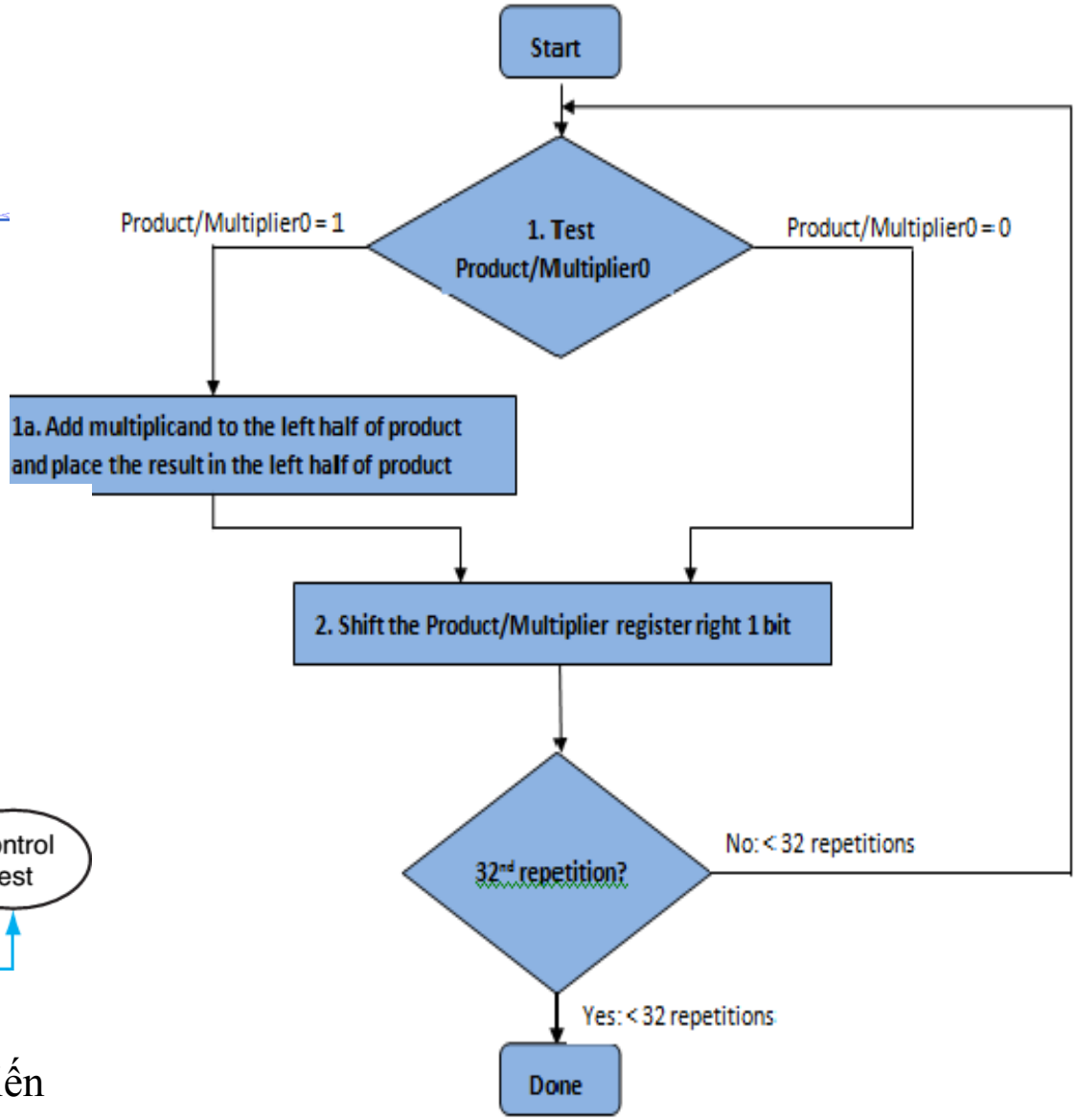
COMPUTER ENGINEERING

Phép Nhân

Giải thuật thực hiện phép nhân theo cấu trúc phần cứng có cải tiến 2 thanh ghi (với hai số 32 bit)



Cấu trúc phần cứng của phép nhân có cải tiến



- So với giải thuật trước đó thì thanh ghi số bị nhân, bộ ALU, thanh ghi số nhân tất cả đều 32 bits, chỉ có thanh ghi tích là khác – 64 bits;
- Trong mỗi vòng lặp, số chu kỳ xung clock tiêu tốn có thể giảm xuống chỉ còn 1 chu kỳ

Phép Nhân

Ví dụ 2:

Sử dung số 6 bit không dấu

$50_{(8)}$ x $23_{(8)}$ = ?

$50_{(8)}$ = 101000 (multiplicand)

$23_{(8)}$ = 010011 (multiplier)

Cấu trúc phần cứng như hình vẽ là nhân 2 số 32 bit, kết quả là số 64 bit,

Có: thanh ghi multiplicand 32 bit

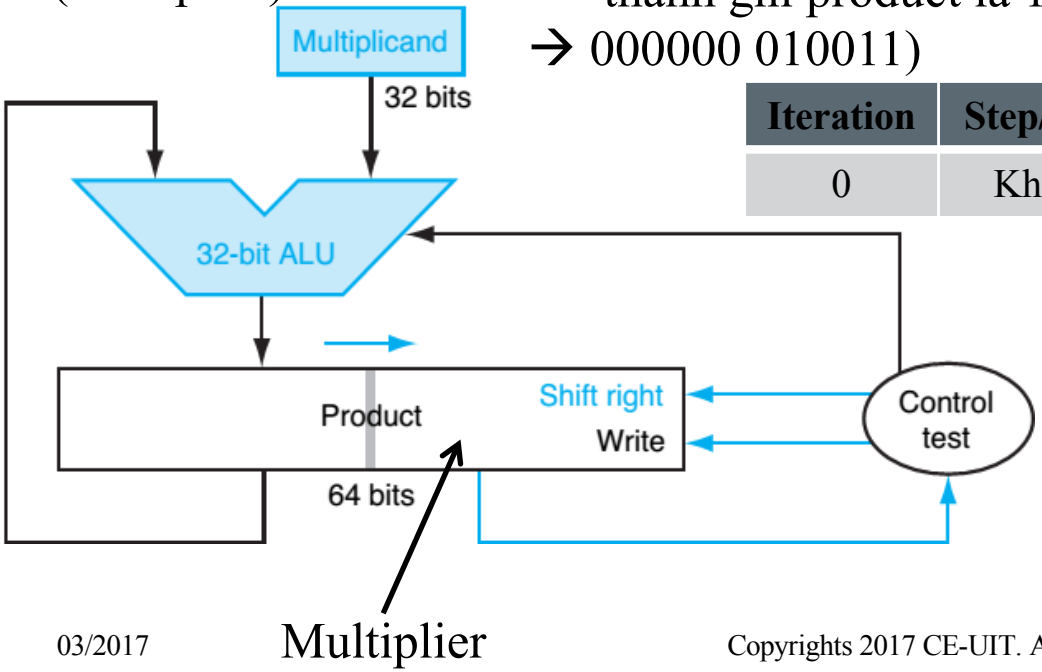
thanh ghi product 64 bit (khi khởi tạo, đưa multiplier vào 32 bit thấp của product, còn nữa cao khởi tạo 0)

Ví dụ 2 yêu cầu nhân 2 số 6 bit, sử dụng cấu trúc phần cứng tương tự như hình, vậy kết quả phải là số 12 bit

⇒ thanh ghi multiplicand 6 bit (giá trị khởi tao 101000)

thanh ghi product là 12 bit (6 bit thấp là multiplier, 6 bit cao là 0 → 000000 010011)

Iteration	Step/Action	Multiplicand	Product/Multiplier
0	Khởi tạo	101000	000000 010011



Ví dụ 2:

$50_{(8)}$ x $23_{(8)}$ = ?

$50_{(8)}$ = 101000 (multiplicand)

$23_{(8)}$ = 010011 (multiplier)

Cấu trúc phần cứng như hình vẽ là nhân 2 số 32 bits, kết quả là số 64 bits,

Có: thanh ghi multiplicand 32 bits

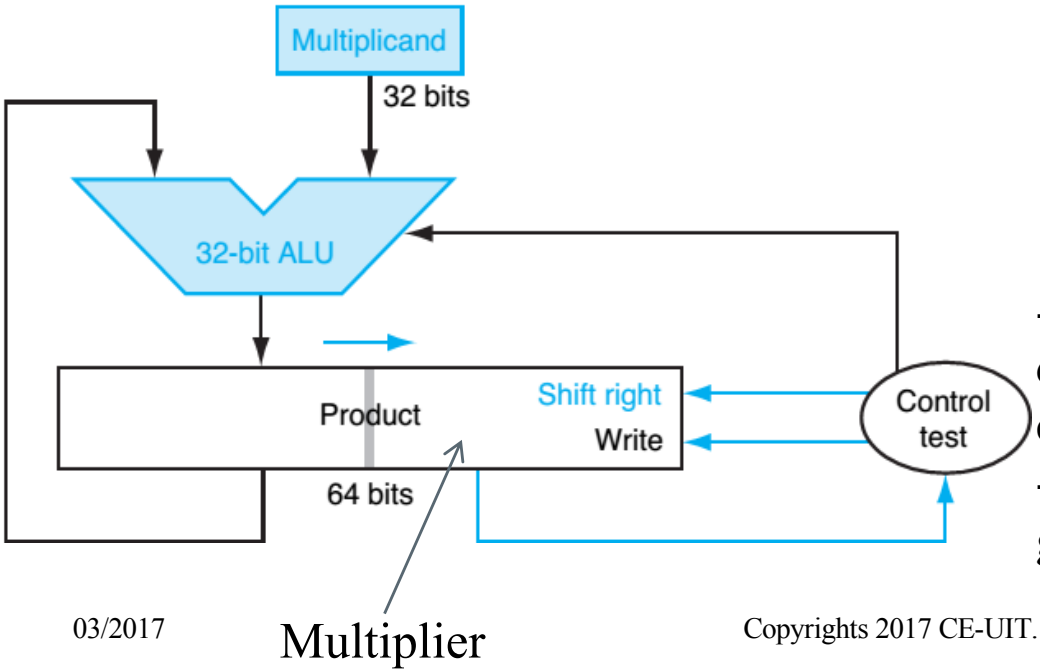
thanh ghi product 64 bits (khi khởi tạo, đưa multiplier vào 32bits thấp của product, còn nữa cao khởi tạo 0)

Ví dụ 2 yêu cầu nhân 2 số 6 bits, sử dụng cấu trúc phần cứng tương tự như hình, vậy kết quả phải là số 12 bits

⇒ thanh ghi multiplicand 6 bits (giá trị khởi tao 101000)

thanh ghi product là 12 bits (6 bit thấp là multiplier, 6 bit cao là 0 → 000000 010011)

Iteration	Step/Action	Multiplicand	Product/Multiplier
0	Khởi tạo	101000	000000 010011



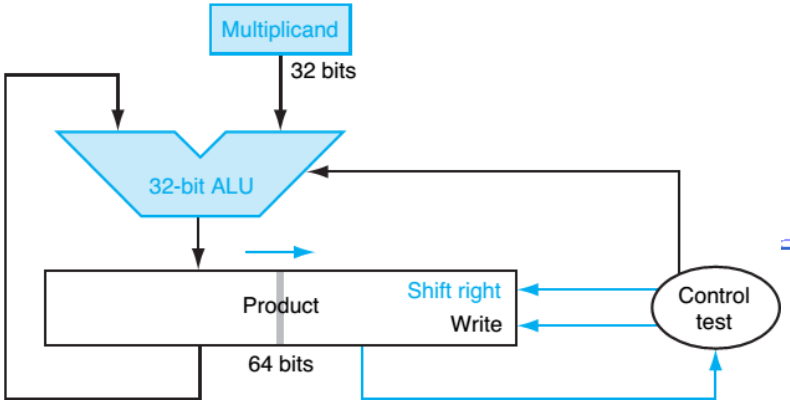
- Sau khi khởi tạo xong. Mỗi vòng lặp (interation) sẽ gồm 2 bước:
  - B1. Kiểm tra bit 0 của Product/multiplier xem có bằng 1 hay không; nếu bằng 1 thì nữa cao của product/multiplier = nữa cao của product/multiplier + multiplicand; nếu bằng 0, không làm gì cả
  - B2. Dịch phải Product/Multiplier 1 bit
- Số vòng lặp cho giải thuật này đúng bằng số bit dùng biểu diễn (ví dụ 2 yêu cầu dùng số 6 bit, thì có 6 vòng lặp)
- Sau khi kết thúc số vòng lặp, giá trị trong thanh ghi product chính là kết quả phép nhân

Iteration	Step/Action	Multiplicand	Product/Multiplier
0	Khởi tạo	101000	000000 010011
1	1a: 1 => Nữa cao của Product/Multiplier = Nữa cao Product/Multiplier + Multiplicand	101000	101000 010011
	2. Shift right Product/Multiplier	101000	010100 001001
2	1a: 1 => Nữa cao của Product/Multiplier = Nữa cao Product/Multiplier + Multiplicand	101000	111100 001001
	2. Shift right Product/Multiplier	101000	011110 000100
3	1: 0 => không làm gì	101000	011110 000100
	2. Shift right Product/Multiplier	101000	001111 000010
4	1: 0 => không làm gì	101000	001111 000010
	2. Shift right Product/Multiplier	101000	000111 100001
5	1a: 1 => Nữa cao của Product/Multiplier = Nữa cao Product/Multiplier + Multiplicand	101000	101111 100001
	2. Shift right Product/Multiplier	101000	0101111 10000
6	1: 0 => không làm gì	101000	0101111 10000
	2. Shift right Product/Multiplier	101000	00101111 1000

Kết quả phép nhân

Hoặc có thể trình bày ngắn gọn như bảng sau:

Step	Action	Multiplicand	Product/Multiplier
0	Initial Vals	101 000	000 000 010 011
1	Prod = Prod + Mcand	101 000	101 000 010 011
	Rshift Product	101 000	010 100 001 001
2	Prod = Prod + Mcand	101 000	111 100 001 001
	Rshift Mplier	101 000	011 110 000 100
3	lsb = 0, no op	101 000	011 110 000 100
	Rshift Mplier	101 000	001 111 000 010
4	lsb = 0, no op	101 000	001 111 000 010
	Rshift Mplier	101 000	000 111 100 001
5	Prod = Prod + Mcand	101 000	101 111 100 001
	Rshift Mplier	101 000	010 111 110 000
6	lsb = 0, no op	101 000	010 111 110 000
	Rshift Mplier	101 000	001 011 111 000



Ví dụ 3: $50_{(16)}$ x $23_{(16)}$, sử dung số 8 bit không dấu

Iteration	Step	Multiplicand	Product/ Multiplier
0	Initial values	0101 0000	0000 0000 0010 0011
1	Prod = Prod + Mcand	0101 0000	0101 0000 0010 0011
	Shift right Product	0101 0000	0010 1000 0001 0001
2	Prod = Prod + Mcand	0101 0000	0111 1000 0001 0001
	Shift right Product	0101 0000	0011 1100 0000 1000
3	lsb = 0, no op	0101 0000	0011 1100 0000 1000
	Shift right Product	0101 0000	0001 1110 0000 0100
4	lsb = 0, no op	0101 0000	0001 1110 0000 0100
	Shift right Product	0101 0000	0000 1111 0000 0010
5	lsb = 0, no op	0101 0000	0000 1111 0000 0010
	Shift right Product	0101 0000	0000 0111 1000 0001
6	lsb = 0, no op	0101 0000	0101 0111 1000 0001
	Shift right Product	0101 0000	0010 1011 1100 0000
7	lsb = 0, no op	0101 0000	0010 1011 1100 0000
	Shift right Product	0101 0000	0001 0101 1110 0000
8	lsb = 0, no op	0101 0000	0001 0101 1110 0000
	Shift right Product	0101 0000	0000 1010 1111 0000

Phép Nhân

Phép nhân có dấu

❖ Cách đơn giản để thực hiện phép nhân có dấu là tách phần trị tuyệt đối và dấu của số bị nhân và số nhân ra.

- Lấy phần trị tuyệt đối dương tương ứng của số nhân và số bị nhân nhân nhau
- Sau đó xét dấu cho tích dựa vào dấu của số nhân và số bị nhân (có thể dùng phép XOR)

Phép Nhân

Phép nhân trong MIPS

❖MIPS sử dụng hai thanh ghi đặc biệt 32 bit là ***Hi*** và ***Lo*** để chứa 64 bit kết quả của phép nhân

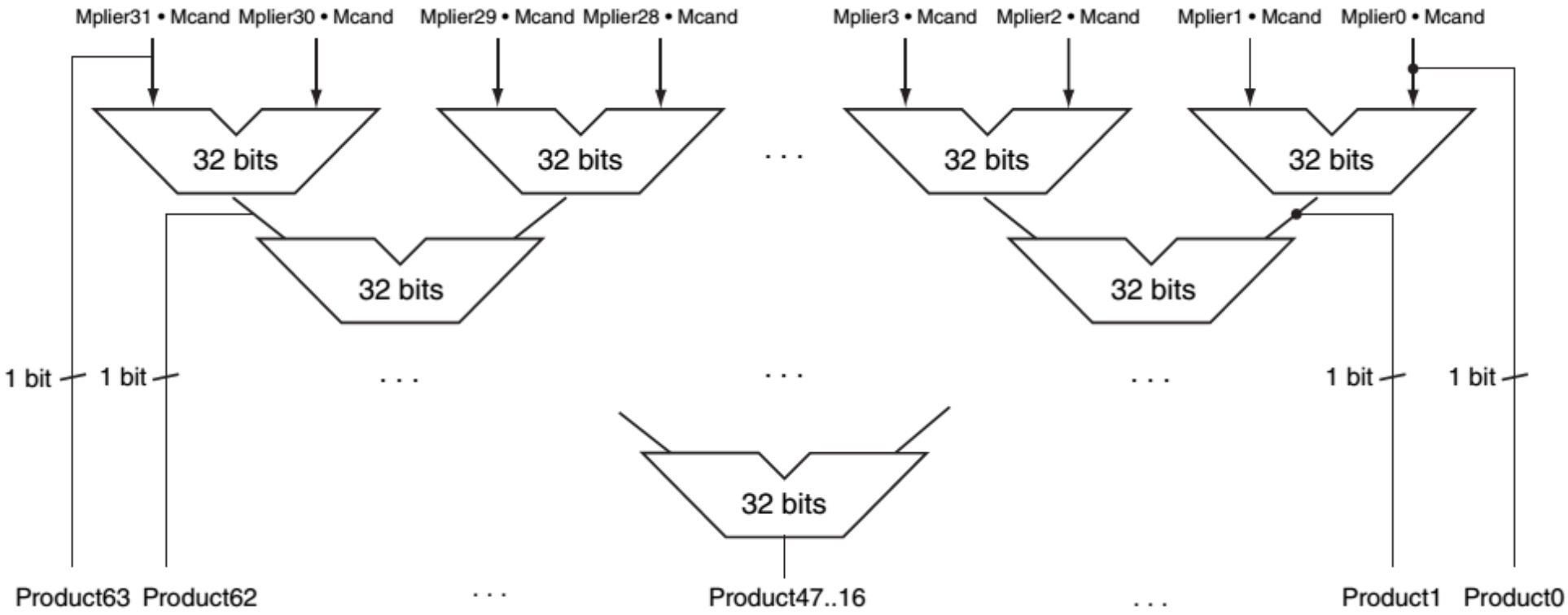
Để lấy giá trị từ thanh ghi ***Hi*** và ***Lo*** ra một thanh ghi khác, sử dụng hai lệnh dành riêng là ***mfhi*** mà ***mflo***

❖ Nhân hai số không dấu, MIPS cung cấp lệnh ***multu***. Nhân hai số có dấu, MIPS cung cấp lệnh ***mult***

Phép Nhân

Giới thiệu một ý tưởng cải tiến phép nhân: Phép nhân theo cách hiện thực tính nhanh

(Sinh viên tự tham khảo thêm)



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

COMPUTER ENGINEERING

Phép Chia

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

Ví dụ:

1001_{ten} Quotient

Divisor 1000_{ten} | 1001010_{ten} Dividend

-1000

10

101

1010

-1000

10_{ten} Remainder

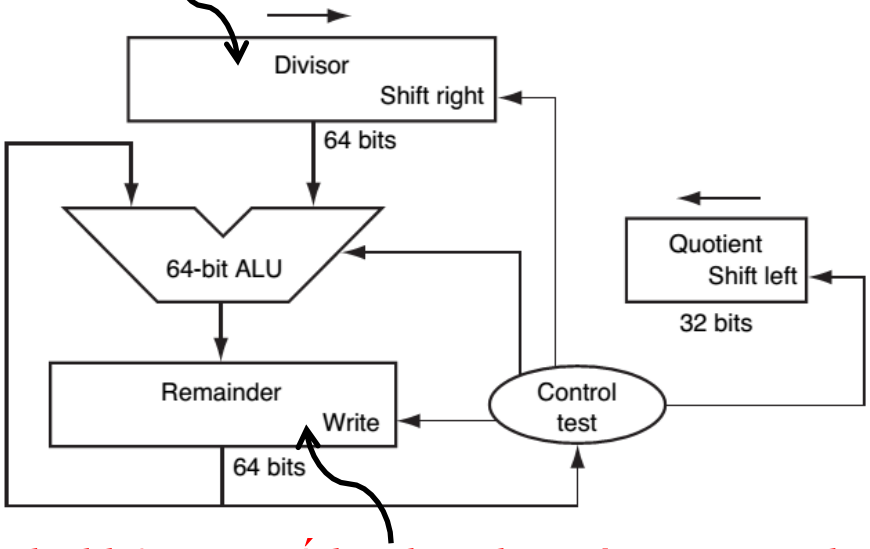
Divisor: số chia
Dividend: số bị chia
Quotient: thương số
Remainder: số dư

COMPUTER ENGINEERING

Phép Chia

Giải thuật thực hiện phép chia trên phần cứng

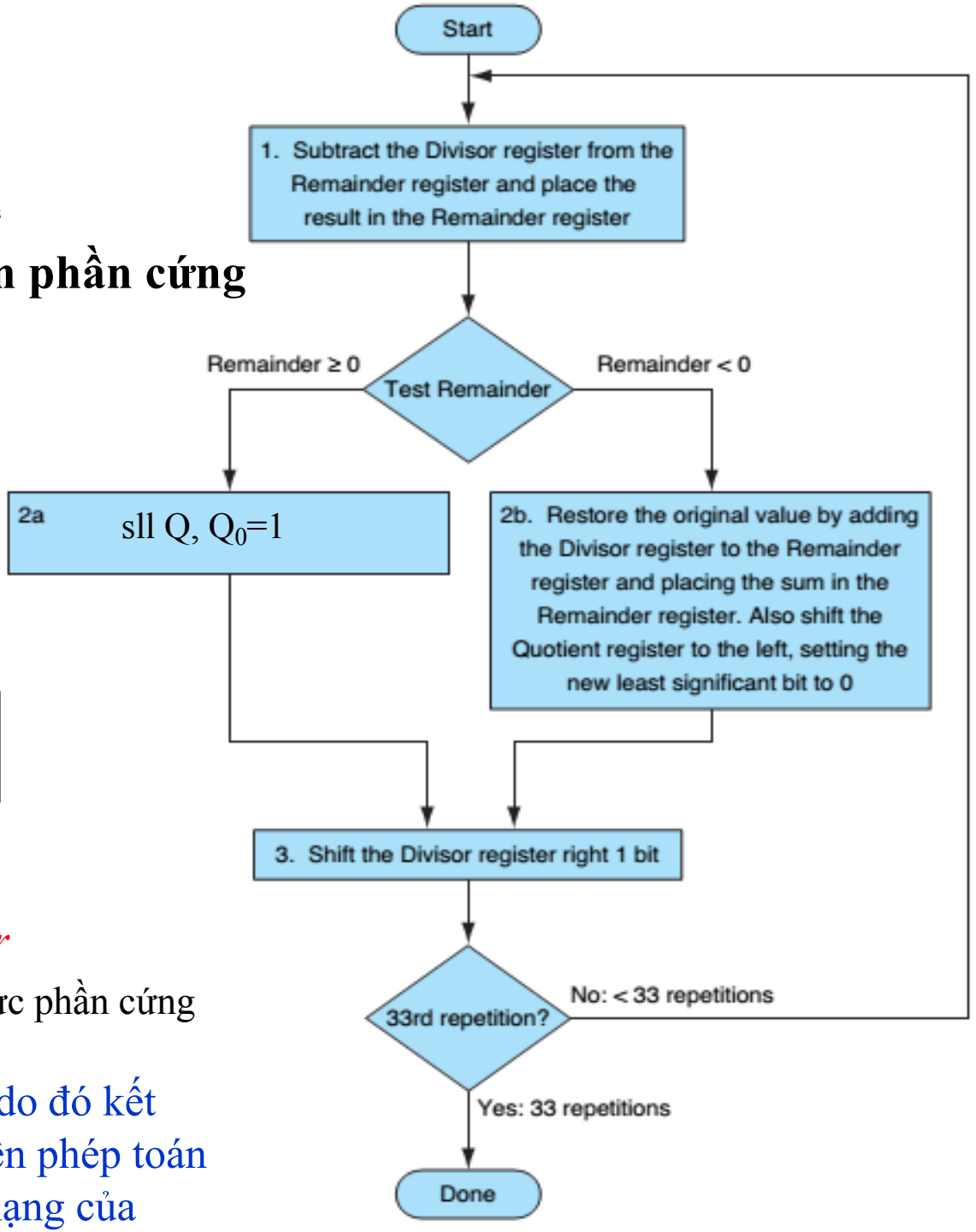
Khi khởi tạo, số chia đưa vào nữa cao Divisor



Khi khởi tạo, số bị chia đưa vào Remainder

Hình 1. 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.

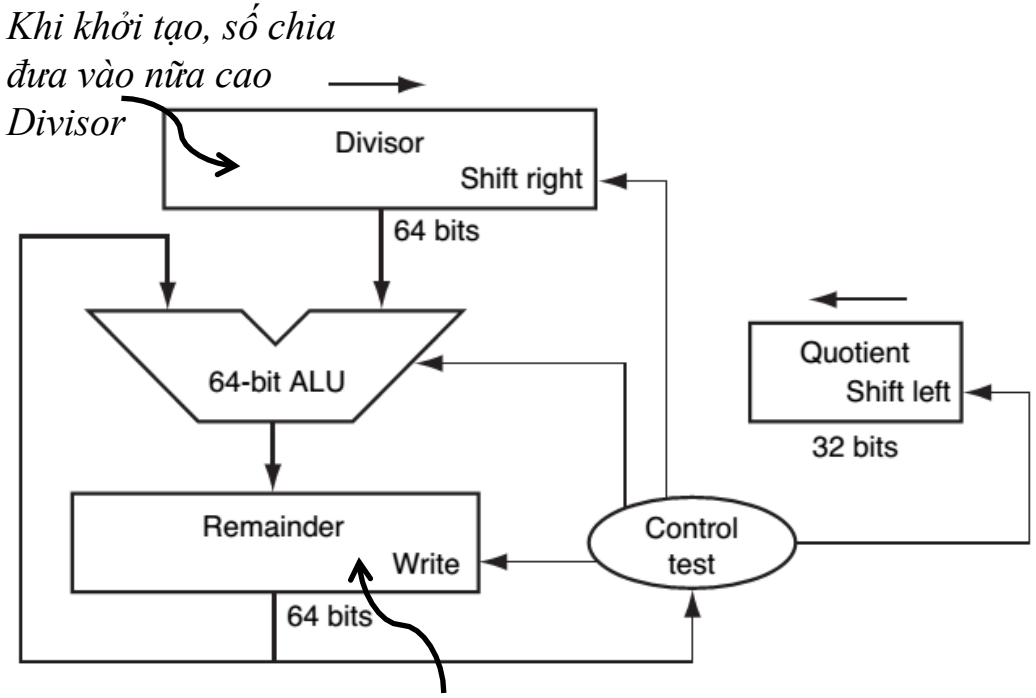


Hình 2. Lưu đồ giải thuật của phép chia

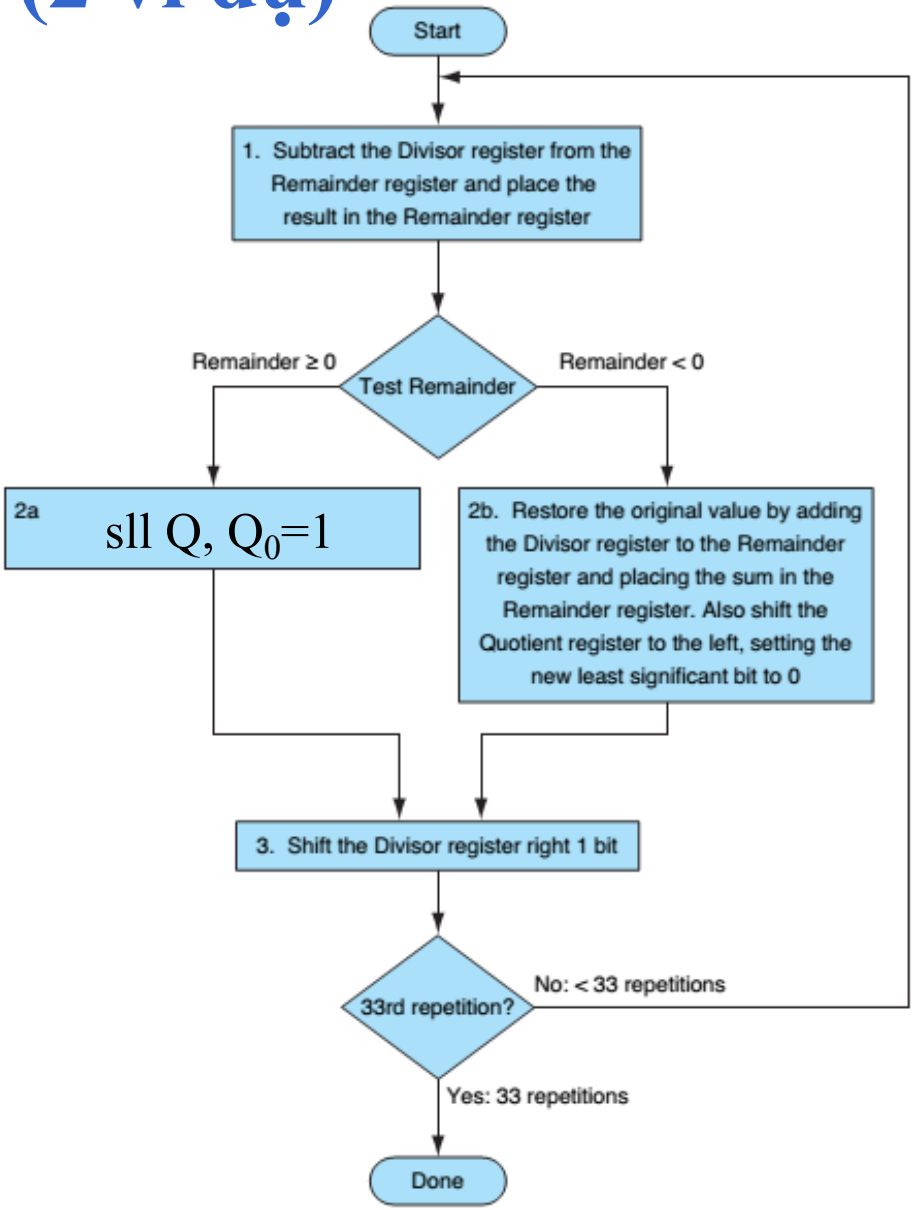
Ví dụ cho phép chia (2 ví dụ)

Ví dụ 1:

Thực hiện phép chia $50_{(8)}/23_{(8)}$ (sử dụng số 6 bit không dấu) theo cấu trúc phần cứng như hình

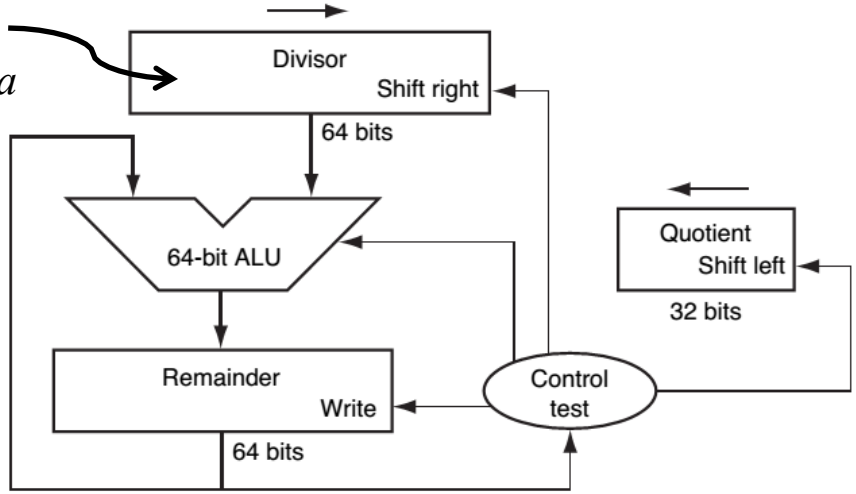


Khi khởi tạo, số bị chia đưa vào Remainder



Lưu đồ giải thuật đi kèm cho cấu trúc phần cứng

Khi khởi tạo, số chia đưa vào nữa cao Divisor



Ví dụ 1:

$50_{(8)}/23_{(8)} = ?$

Dividend = $50_8 = 101\ 000_2$

Divisor = $23_8 = 010\ 011_2$

Cấu trúc phần cứng như hình vẽ là đang làm việc trên phép chia số 32 bits

Có: thanh ghi divisor 64 bits

thanh ghi quotient là 32 bits

thanh ghi remainder là 64 bits

Ví dụ 1 yêu cầu phép chia dùng số 6 bits không dấu, sử dụng cấu trúc phần cứng tương tự như hình, vậy các thanh ghi trong ví dụ cần được khởi tao với số bit tương ứng:

=> thanh ghi divisor 12 bits (giá trị khởi tao **010011**000000 – **6 bits cao là giá trị của divisor, 6 bits thấp đưa 0 vào**)

thanh ghi quotient là 6 bits (giá trị khởi tạo 000000)

thanh ghi remainder là 12 bits (giá trị khởi tạo 000000**101000** - 6 bits cao đưa 0 vào, 6 bits thấp đưa dividend vào)

-Sau khi khởi tạo xong. Mỗi vòng lặp (interation) sẽ gồm 3 bước:

- B1. Lấy toàn bộ remainder trừ divisor (hiệu lưu đè lên giá trị remainder hiện đang có)
- B2. Kiểm tra hiệu vừa tính ở trên là âm hay dương (kiểm tra bit trọng số cao nhất, nếu 1 là âm, nếu 0 là dương):

  Nếu âm:

  - Lấy giá trị hiện tại của remainder cộng với divisor, tổng lưu lại vào remainder
  - Dich trái quotient 1 bit
  - Thêm 0 vào bit 0 của quotient (thật ra thao tác này không cần, vì dịch trái 1 bit mặc định đã thêm 0 vào bit 0 của nó)

  Nếu dương:

  - Dich trái quotient 1 bit
  - Chuyển bit 0 của quotient thành 1
- B3. Dịch phải Divisor 1 bit

- Số vòng lặp cho giải thuật này đúng bằng số bit dùng biểu diễn + 1 (ví dụ 1 yêu cầu dùng số 6 bit, thì có 7 vòng lặp)

- Sau khi kết thúc số vòng lặp, giá trị trong thanh ghi quotient chính là kết quả phép chia, giá trị trong remainder là phần dư

Step	Action	Quotient	Divisor	Remainder
0	Initial Vals (Giá trị khởi tạo)	000 000	010 011 000000	000000 101000

Ví dụ 1:

$50_{(8)}/23_{(8)} = ?$

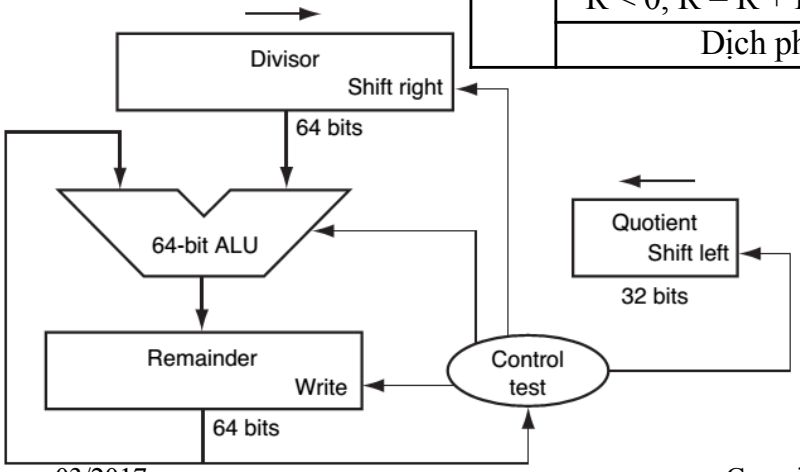
Dividend = 50_8 = $101\ 000_2$

Divisor = 23_8 = $010\ 011_2$

Step	Action	Quotient	Divisor	Remainder
0	Initial Vals	000 000	010 011 000 000	000 000 101 000
1	R = R – D	000 000	010 011 000 000	**1**01 101 101 000
	R < 0, R = R + D, dịch trái Q 1 bit	000 000	010 011 000 000	000 000 101 000
	Dịch phải D 1 bit	000 000	001 001 100 000	000 000 101 000
2	R = R – D	000 000	001 001 100 000	**1**10 111 001 000
	R < 0, R = R + D, dịch trái Q 1 bit	000 000	001 001 100 000	000 000 101 000
	Dịch phải D 1 bit	000 000	000 100 110 000	000 000 101 000
3	R = R – D	000 000	000 100 110 000	**1**11 011 111 000
	R < 0, R = R + D, dịch trái Q 1 bit	000 000	000 100 110 000	000 000 101 000
	Dịch phải D 1 bit	000 000	000 010 011 000	000 000 101 000
4	R = R – D	000 000	000 010 011 000	**1**11 110 010 000
	R < 0, R = R + D, dịch trái Q 1 bit	000 000	000 010 011 000	000 000 101 000
	Dịch phải D 1 bit	000 000	000 001 001 100	000 000 101 000
5	R = R – D	000 000	000 001 001 100	**1**11 110 111 100
	R < 0, R = R + D, dịch trái Q 1 bit	000 000	000 001 001 100	000 000 101 000
	Dịch phải D 1 bit	000 000	000 000 100 110	000 000 101 000
6	R = R – D	000 000	000 000 100 110	**0**00 000 000 010
	R> 0, dịch trái Q 1 bit, $Q_0 = 1$	000 001	000 000 100 110	000 000 000 010
	Dịch phải D 1 bit	000 001	000 000 010 011	000 000 000 010
7	R = R – D	000 001	000 000 010 011	**1**11 111 101 111
	R < 0, R = R + D, dịch trái Q 1 bit	000 010	000 000 010 011	000 000 000 010
	Dịch phải D 1 bit	000 010	000 000 001 001	000 000 000 010

Thương số

Phần dư



Ký hiệu: Q, D và R lần lượt là viết tắt của Quotion, Divisor và Remainder

COMPUTER ENGINEERING

Phép Chia

Giải thuật thực hiện phép chia trên phần cứng

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

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

tức Remainder = Remaider + Divisor

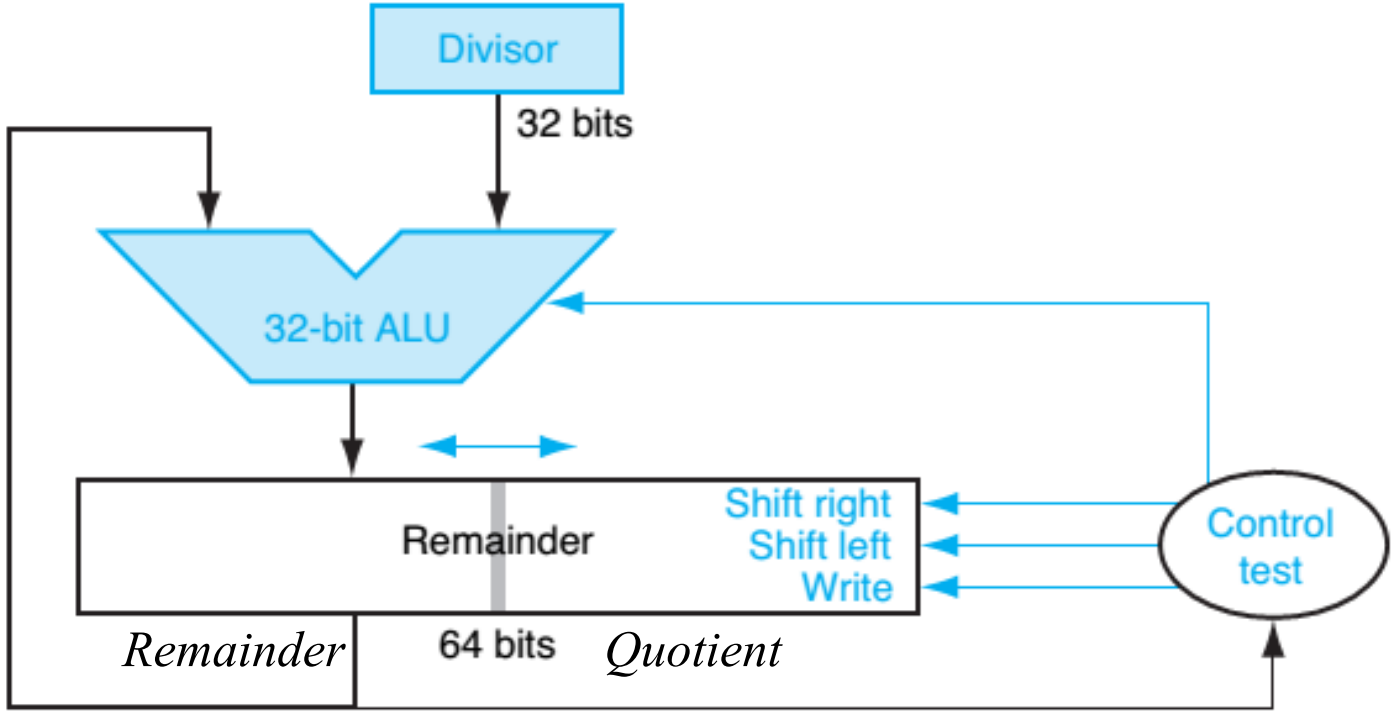
Bảng thực hiện giải thuật phép chia theo từng bước

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

Giải thuật thực hiện phép chia trên phần cứng có cải tiến

(Sinh viên tự tham khảo thêm)



Cấu trúc phần cứng phép chia có cải tiến

Phép Chia

Phép chia có dấu

Nếu phép chia có dấu

- Bước 1. Bỏ qua dấu, thực hiện phép chia thông thường
- Bước 2. Xét dấu
  - ✓ Dấu của thương sẽ trái với dấu hiện tại nếu dấu của số chia và số bị chia trái ngược nhau
  - ✓ Dấu của số dư:

Các xác định bit dấu cho số dư bằng công thức sau:

Số bị chia = Thương × Số chia + Số dư

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

Ví dụ: – 7 : +2 thì thương = – 3, dư = –1

Kiểm tra kết quả:

–7 = –3 × 2 + (–1) = –6 – 1

Phép Chia

Phép chia trong MIPS

- Trong cấu trúc phần cứng cho phép nhân có cải tiến, hai thanh ghi ***Hi*** và ***Lo*** được ghép lại để hoạt động như thanh ghi 64 bit của Product/Multiplier

  Quan sát cấu trúc phần cứng cho phép nhân có cải tiến và phép chia có cải tiến, rõ ràng hai cấu trúc này tương tự nhau.

  Từ đó, MIPS cũng sử dụng hai thanh ghi ***Hi*** và ***Lo*** cho cả phép nhân và chia.

- Sau khi phép chia thực hiện xong:
  - ✓ ***Hi*** chứa phần dư
  - ✓ ***Lo*** chứa thương số
- Để xử lý cho các số có dấu và số không dấu, MIPS có 2 lệnh: phép chia có dấu (***div***), và phép chia không dấu (***divu***).

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**

Tổng kết:

- Hiểu quy tắc thực hiện các phép toán số học (cộng, trừ, nhân và chia) trên số nguyên trong máy tính
- Hiểu cách thiết kế mạch nhân và chia cơ bản cho số nguyên trong 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 thực dấu chấm động**

Số thực dấu 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)

- ***Scientific notation:*** Một số thực được gọi là "scientific notation" khi bên trái dấu chấm có đúng 1 chữ số.
- ***Normalized number:*** Một số thực được gọi là "***Normalized number***" (dạng chuẩn) khi số này được viết trong "scientific notation" và chữ số bên trái dấu chấm không phải là 0.

  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

COMPUTER ENGINEERING

Số thực dấu chấm động

Định nghĩa:

- Trong máy tính, các số nhị phân phải được đưa về dạng chuẩn như sau:

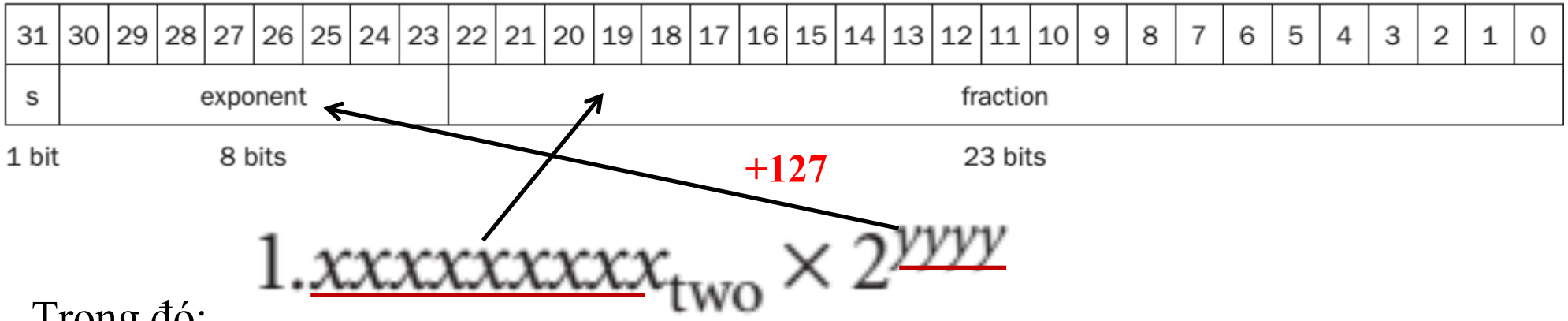
$$1.xxxxxxxxxx_{two} \times 2^{yyyy}$$

COMPUTER ENGINEERING

Số thực dấu chấm động

Biểu diễn số thực dấu chấm động

Biểu diễn số thực dấu chấm động theo chuẩn **IEEE 754** (với độ chính xác đơn)
(chuẩn này được áp dụng cho hầu hết các máy tính được chế tạo từ năm 1980)



Trong đó:

s biểu diễn 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. Exponent là biểu diễn quá 127 của *yyyy* (*excess-127* hoặc *bias of 127*).

Phần lẻ (fraction) dùng 23 bits để biểu diễn cho *xxxxxxxx*

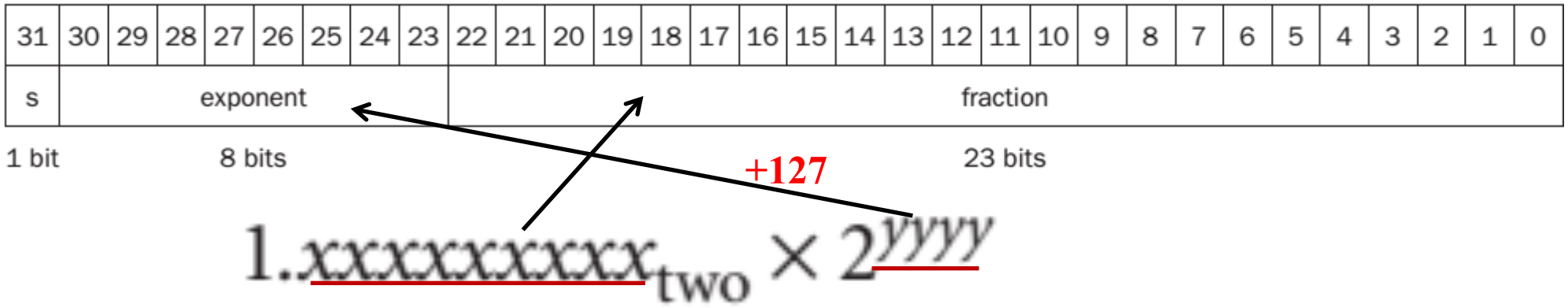
Tổng quát, số thực dấu chấm động được tính dựa theo (với Bias = 127):

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

Số thực dấu chấm động

Biểu diễn số thực dấu chấm động

Biểu diễn số thực dấu chấm động theo chuẩn IEEE 754 (với độ chính xác đơn)
(chuẩn này được áp dụng cho hầu hết các máy tính được chế tạo từ năm 1980)



Tổng quát, số thực dấu chấm động được tính dựa theo:

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

Hoặc:

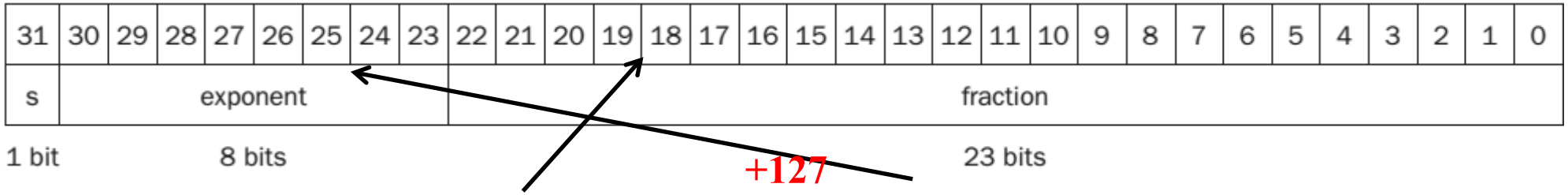
$$(-1)^S \times (1 + (s1 \times 2^{-1}) + (s2 \times 2^{-2}) + (s3 \times 2^{-3}) + (s4 \times 2^{-4}) + \ldots) \times 2^E$$

(với s1, s2, s3 … là các bit lần lượt từ trái sang phải của fraction)

Số thực dấu chấm động

Biểu diễn số thực dấu chấm động

Biểu diễn số thực dấu chấm động theo chuẩn IEEE 754 (với độ chính xác đơn)
(chuẩn này được áp dụng cho hầu hết các máy tính được chế tạo từ năm 1980)

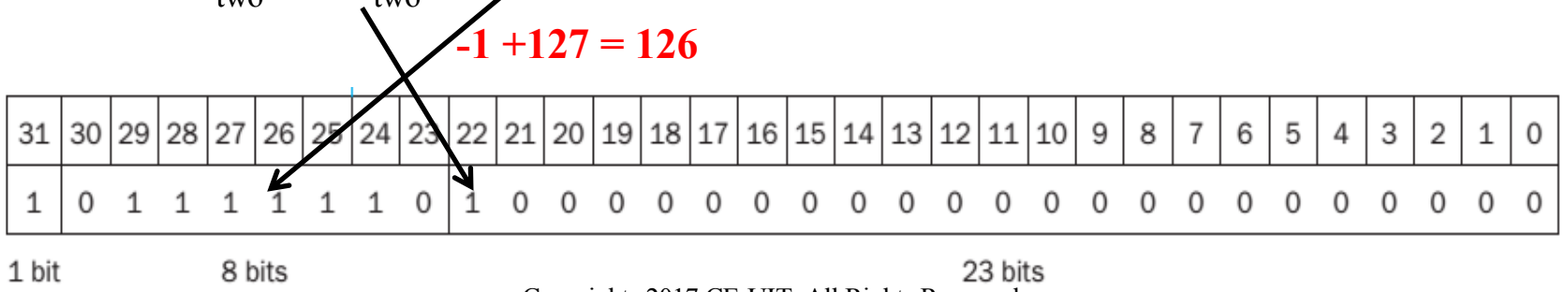


$$1.xxxxxxxxxx_{two} \times 2^{yyyy}$$

Ví dụ: Số -0.75 sẽ được biểu diễn trong máy tính như thế nào nếu dùng chuẩn IEEE 754 với độ chính xác đơn

$-0.75_{ten} = -3/4_{ten} = -3/2^2_{ten} = -11_{two}/2^2_{ten} = -0.11_{two}$

Chuẩn hóa: $0.11_{two} = 1.1_{two} \times 2^{-1}$



COMPUTER ENGINEERING

Số thực dấu chấm động

Ví dụ: Cho biểu diễn số dấu chấm động với độ chính xác đơn như hình sau, hỏi số tương ứng với biểu diễn này trong hệ thập phân là bao nhiêu?

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:

bit dấu s là 1

exponent chứa 129

Số tương ứng: $(-1)^s \times (1+ \text{fraction}) \times 2^{(\text{exponent} - 127)}$

$= (-1)^1 \times (1+ 0.01) \times 2^{(129 - 127)}$

$= (-1.01 \times 2^2)_{two} = -5.0_{ten}$

Số thực dấu 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).

Nhằm hạn chế việc tràn trên hoặc tràn dưới về số mũ, IEEE 754 giới thiệu thêm một cách biểu diễn số thực dấu chấm động, vơi trường exponent mở rộng lên tới 11 bits. Cách biểu diễn này gọi là IEEE 754 với độ chính xác kép

- **Độ chính xác đơn (Single precision):** một số thực dấu chấm động được biểu diễn ở dạng 32 bit.
- **Độ chính xác kép (Double precision):** một số thực dấu chấm động được biểu diễn ở dạng 64 bit.

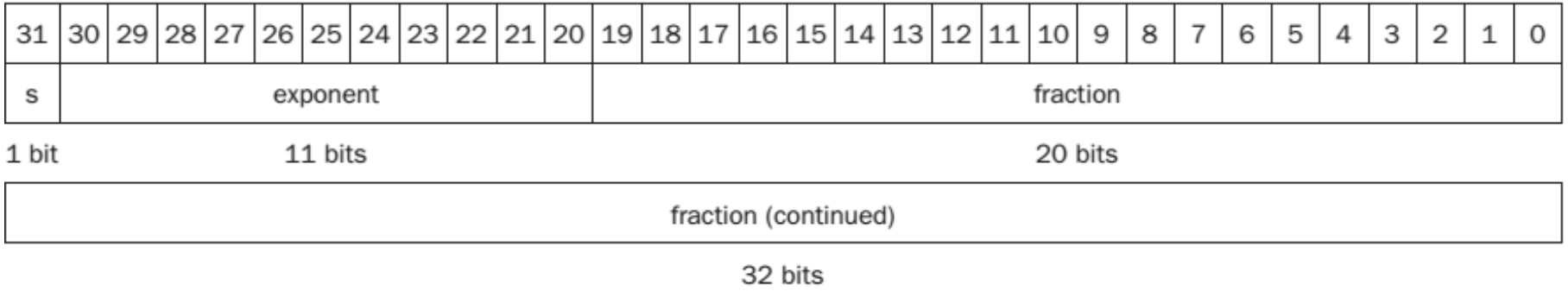
*Chú ý: Trong lập trình ngôn ngữ C, các số thực dạng **float** sẽ được định dạng theo kiểu độ chính xác đơn, còn các số dạng **double** sẽ được định dạng theo kiểu độ chính xác kép*

COMPUTER ENGINEERING

Số thực dấu chấm động

Biểu diễn số thực dấu chấm động

Biểu diễn số thực dấu chấm động theo chuẩn **IEEE 754** (với độ chính xác kép)



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

Trong đó:

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

Phần mũ (exponent) có kích thước là 11 bits. Exponent là biểu diễn quá 1023 của yyyy (*excess-1023* hoặc *bias of 1023*).

Phần lẻ (fraction) dùng 52 bits để biểu diễn

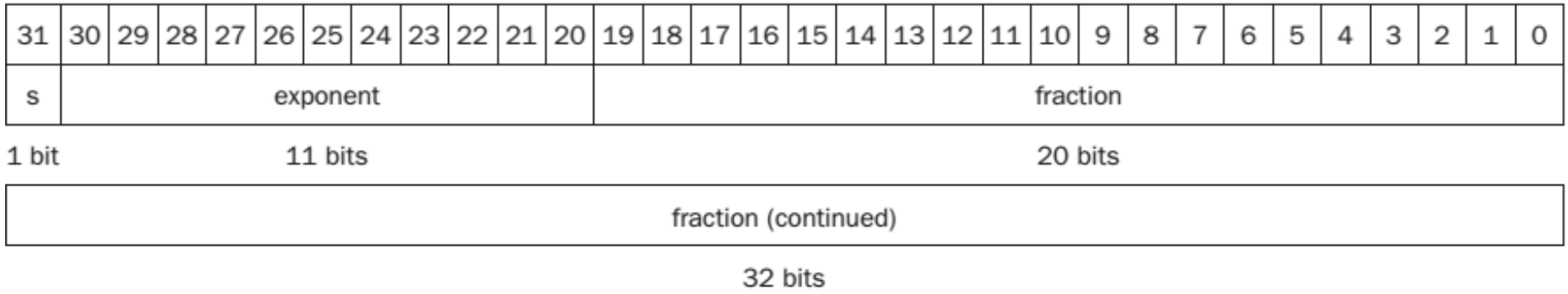
Tổng quát, số thực dấu chấm động được tính dựa theo (với Bias = 1023):

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

Số thực dấu chấm động

Biểu diễn số thực dấu chấm động

Biểu diễn số thực dấu chấm động theo chuẩn **IEEE 754** (với độ chính xác kép)

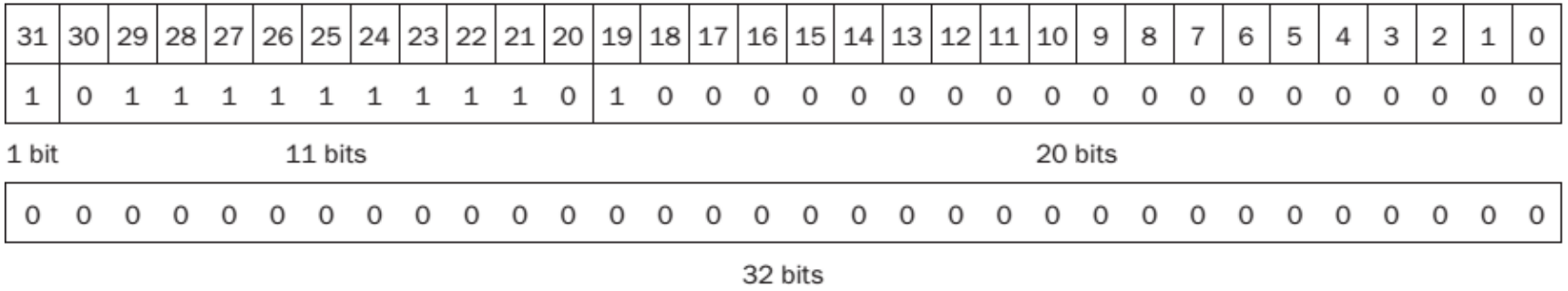


$$1.xxxxxxxxxx_{two} \times 2^{yyyy}$$

Ví dụ: Số -0.75 sẽ được biểu diễn trong máy tính như thế nào nếu dùng chuẩn IEEE 754 với độ chính xác kép

$-0.75_{ten} = -3/4_{ten} = -3/2^2_{ten} = -11_{two}/2^2_{ten} = 0.11_{two}$

Chuẩn hóa: $0.11_{two} = 1.1_{two} \times 2^{-1}$ **(phần exponent = -1 + 1023 = 1022 = 01111111110)**



Số thực dấu chấm động

Biểu diễn số thực dấu chấm động

Tại sao IEEE 754 không sử dụng biểu diễn dạng bù hai cho phần mũ mà dùng dạng bias-of-127 cho độ chính xác đơn và và bias-of-1023 cho độ chính xác kép?

Ví dụ: giả sử dùng bù 2 để biểu diễn phần mũ cho 2 số sau:

$1.0_{two} \times 2^{-1}$

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	.	.	.

$1.0_{two} \times 2^{+1}$

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	.	.	.

➔ Khi nhìn vào phần mũ của $1.0_{two} \times 2^{-1}$ thì nó lại giống như là số rất lớn (thực chất lại là nhỏ), còn trong khi nhìn vào phần mũ của $1.0_{two} \times 2^{+1}$ thì nó lại giống như là số nhỏ (thực chất lại là lớn) ➔ vì vậy IEEE 754 chọn cách biểu diễn dùng bias-of-127 cho độ chính xác đơn thay vì bù 2

COMPUTER ENGINEERING

Số thực dấu chấm động

Biểu diễn số thực dấu chấm động dùng IEEE 754

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

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

Số nhỏ nhất:

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

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)

IEEE 754 mã hóa số thực dấu chấm động

Số thực dấu chấm động

Biểu diễn số thực dấu chấm động

Các vấn đề cần lưu ý:

Rõ ràng, trong một biểu diễn số thực dấu chấm động nếu

- Tăng số bit chứa phần fraction thì tăng độ chính xác.
- Tăng kích thước phần exponent là tăng tầm trị biểu diễn.

➔ Vì vậy, khi thiết kế một biểu diễn/thể hiện cho số dấu chấm động (ví dụ không sử dụng IEEE 754) thì tùy vào mục đích sử dụng mà lựa chọn số giới hạn cho fraction và exponent sao cho phù hợp nhất.

COMPUTER ENGINEERING

Số thực dấu chấm động

Biểu diễn số thực dấu chấm động

Các vấn đề cần lưu ý:

- Số thực dấu chấm động dạng nhị phân (binary floating-point) dạng chuẩn

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

- Số thực dấu chấm động dạng thập phân (decimal floating-point) dạng chuẩn:

$$1.xxxxxxxxx_{ten} \times 10^{yyyy}$$

***yyyy*: exponent (phần mũ)**

***xxxxxxxxx*: fraction (tạm dịch là phần phân số/lẻ)**

***1.xxxxxxxxx*: significand (tạm dịch là phần trị)**

Số thực dấu chấm động

Phép toán cộng trên số thực dấu chấm động

Ví dụ: Thực hiện cộng hai số thực dấu chấm động chuẩn trong hệ thập phân sau

$$9.999_{ten} \times 10^{1} + 1.610_{ten} \times 10^{-1}.$$

Giả sử số thực dấu chấm động lưu trữ phần trị (significand) dùng 4 chữ số, phần số mũ (exponent) lưu trữ dùng 2 chữ số.

Bước 1.

Điều chỉnh sao cho phần mũ của hai số hạng trở thành bằng nhau
(Lấy số hạng có số mũ nhỏ hơn điều chỉnh theo số hạng có số mũ lớn hơn)

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

Vì significand chỉ cho phép dùng 4 chữ số, nên $0.01610_{ten} \times 10^{1}$ làm tròn thành 0.016×10^{1}

(quy tắc làm tròn tùy vào đề bài yêu cầu. Trong ví dụ này, làm tròn theo quy tắc nếu chữ số bên phải của phần bỏ đi lớn hơn hoặc bằng 5 thì chữ số bên trái nhất của phần còn lại tăng lên 1)

Số thực dấu chấm động

Phép toán cộng trên số thực dấu chấm động

Bước 2.

Thực hiện cộng phần significand của hai số hạng

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

Tổng là $10.015_{ten} \times 10^{1}$

Bước 3.

- Chuyển tổng về dạng chuẩn hóa: $10.015_{ten} \times 10^{1} = 1.0015 \times 10^{2}$
- Kiểm tra phần mũ có bị tràn trên, tràn dưới ? => không tràn

(Nếu tràn, phép toán sẽ tạo ra một ngoại lệ (exception) và dừng)

Bước 4.

Làm tròn tổng: vì significand chỉ cho phép dùng 4 chữ số, nên 1.0015×10^{2} làm tròn thành 1.002×10^{2}

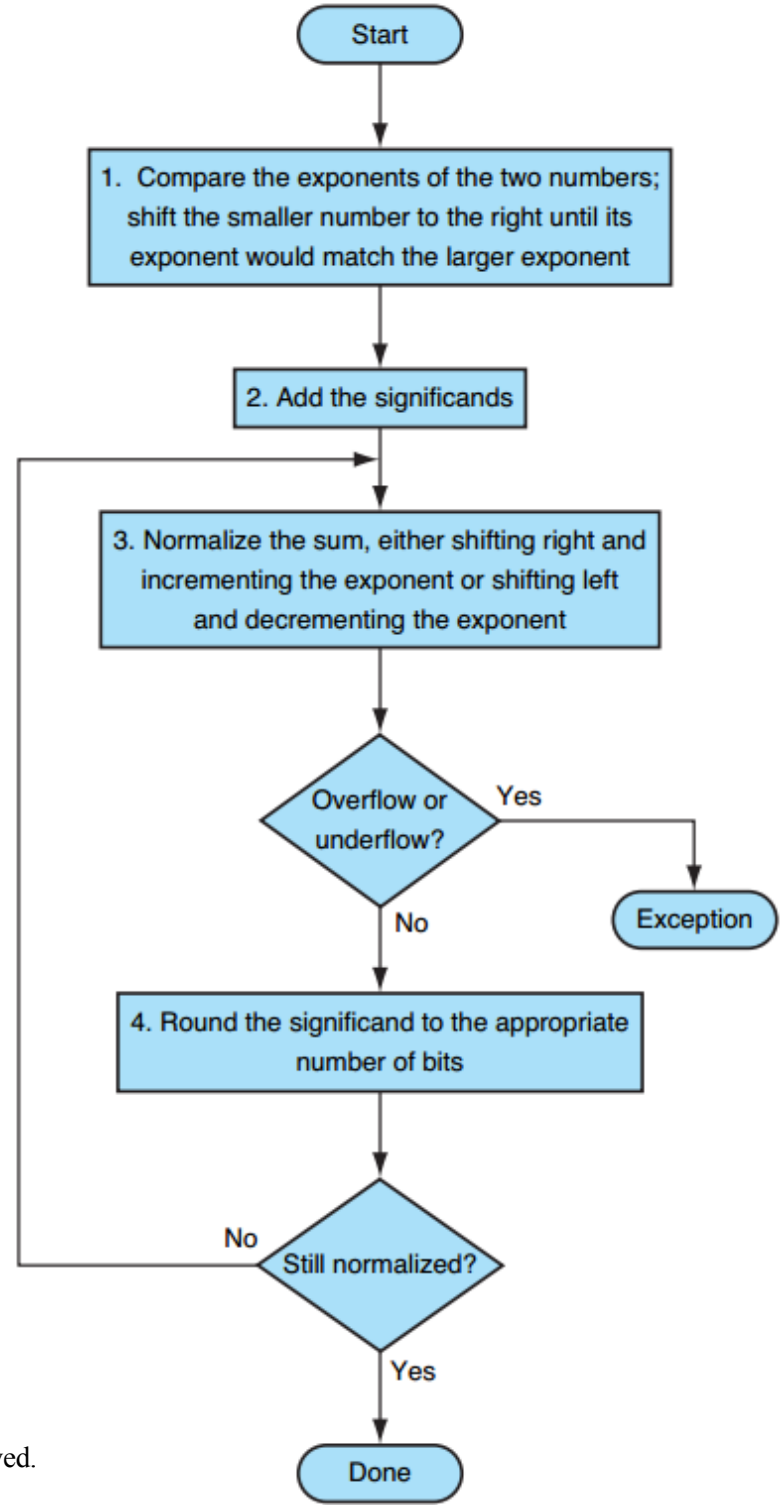
Lưu ý: Việc làm tròn trong một số trường hợp có thể làm tổng mất đi dạng chuẩn hóa. Vì vậy sau khi làm tròn, phải kiểm tra xem tổng có còn trong dạng chuẩn hóa hay không, nếu không, quay lại bước 3

COMPUTER ENGINEERING

Số thực dấu 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* tương tự như cho số *hệ thập phân*



Số thực dấu 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 lưu đồ giải thuật.

Giả sử phần significant dùng 4 bits lưu trữ, còn phần mũ lưu trữ như IEEE 754 độ chính xác đơn.

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 thực dấu chấm động

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.

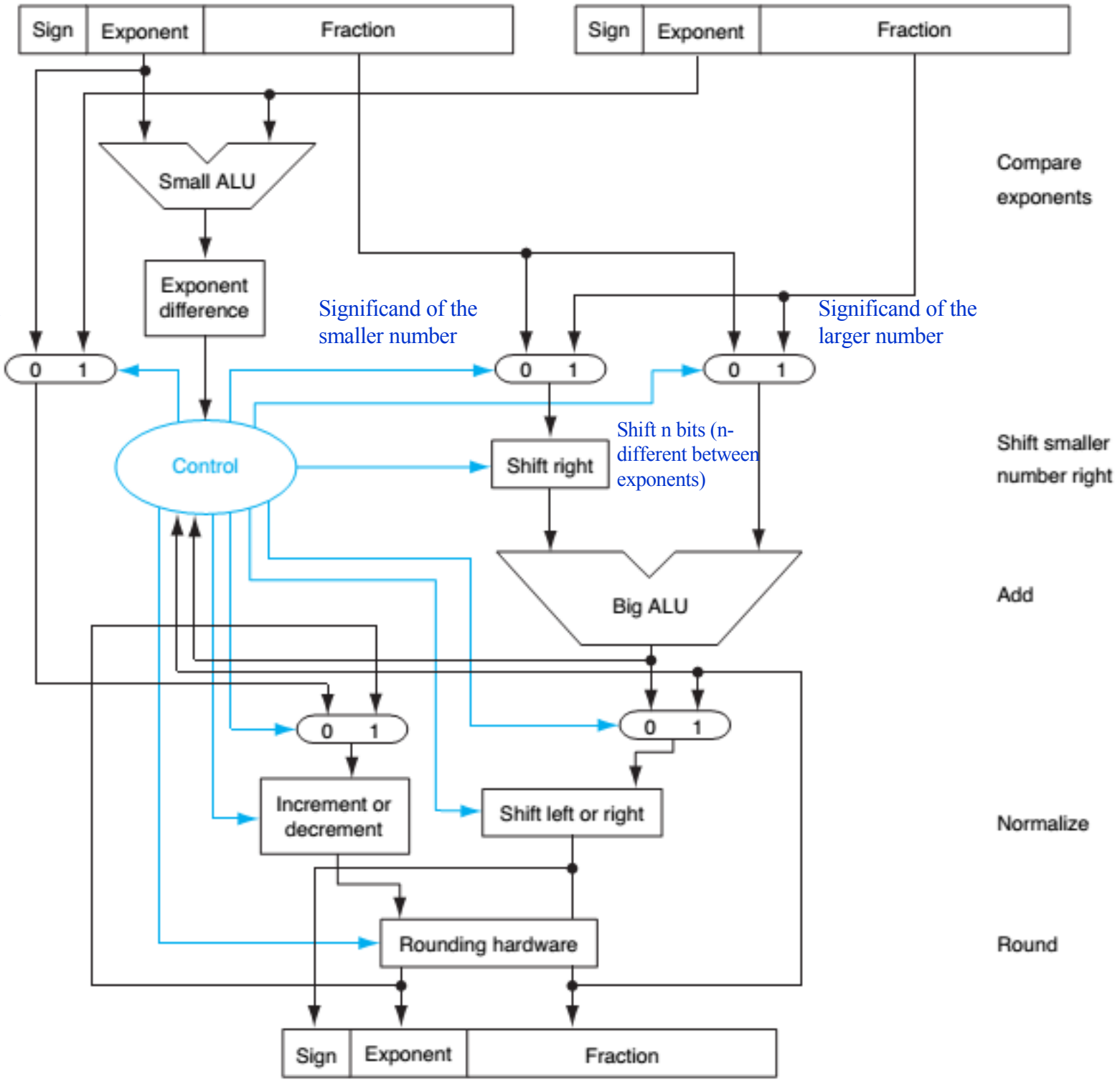
This sum is then

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

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

COMPUTER ENGINEERING

Kiến trúc phần cứng phép nhân hai số floating-point:



Số thực dấu chấm động

Phép nhân trên số thực dấu chấm động

Ví dụ: Thực hiện phép nhân hai số thực dấu chấm động chuẩn trong hệ thập phân sau: $(1.110_{10} \times 10^{10}) \times (9.200_{10} \times 10^{-5})$

Giả sử số thực dấu chấm động lưu trữ phần trị dùng 4 chữ số và phần mũ dùng 2 chữ số.

Đáp án:

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

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

Số thực dấu 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:

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 thực dấu 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 thực dấu chấm động

Phép nhân trên số thực dấu chấm động

Việc thực hiện phép nhân trên số thực dấu chấm động nhị phân cũng tương tự như ví dụ trên, nhưng lưu ý phần mũ khi được lưu theo định dạng IEEE 754

Ví dụ:

Cần nhân hai số thực dấu chấm động đang được lưu trữ theo IEEE 754 độ chính xác đơn, biết 8 bit phần mũ của số thứ nhất trong lưu trữ có giá trị là 137_{ten} và 8 bit phần mũ của số thứ hai trong lưu trữ có giá trị là 122_{ten}.

Phần mũ của tích khi lưu trữ :

$$137_{ten} + 122_{ten} = 259_{ten}$$

Giá trị 259_{ten} đúng hay sai? ➔ Sai

Giá trị đúng của tích trong lưu trữ phải là:

$$(137_{ten} + 122_{ten}) - 127_{ten} = 132_{ten}$$

Vì thực chất:

- Số mũ của số thứ nhất là 10. Khi được lưu trữ theo IEEE 754, phần mũ lưu 10 + 127 = 137
- Số mũ của số thứ hai là -5. Khi được lưu trữ theo IEEE 754, phần mũ lưu -5 + 127 = 122

Số mũ của tích phải là 10 + (-5) = 5 Và nếu được lưu trữ theo IEEE 754, phần mũ của tích lưu 5 + 127 = 132

Vì vậy nếu lấy 137 + 122 thì 127 đã được cộng hai lần

COMPUTER ENGINEERING

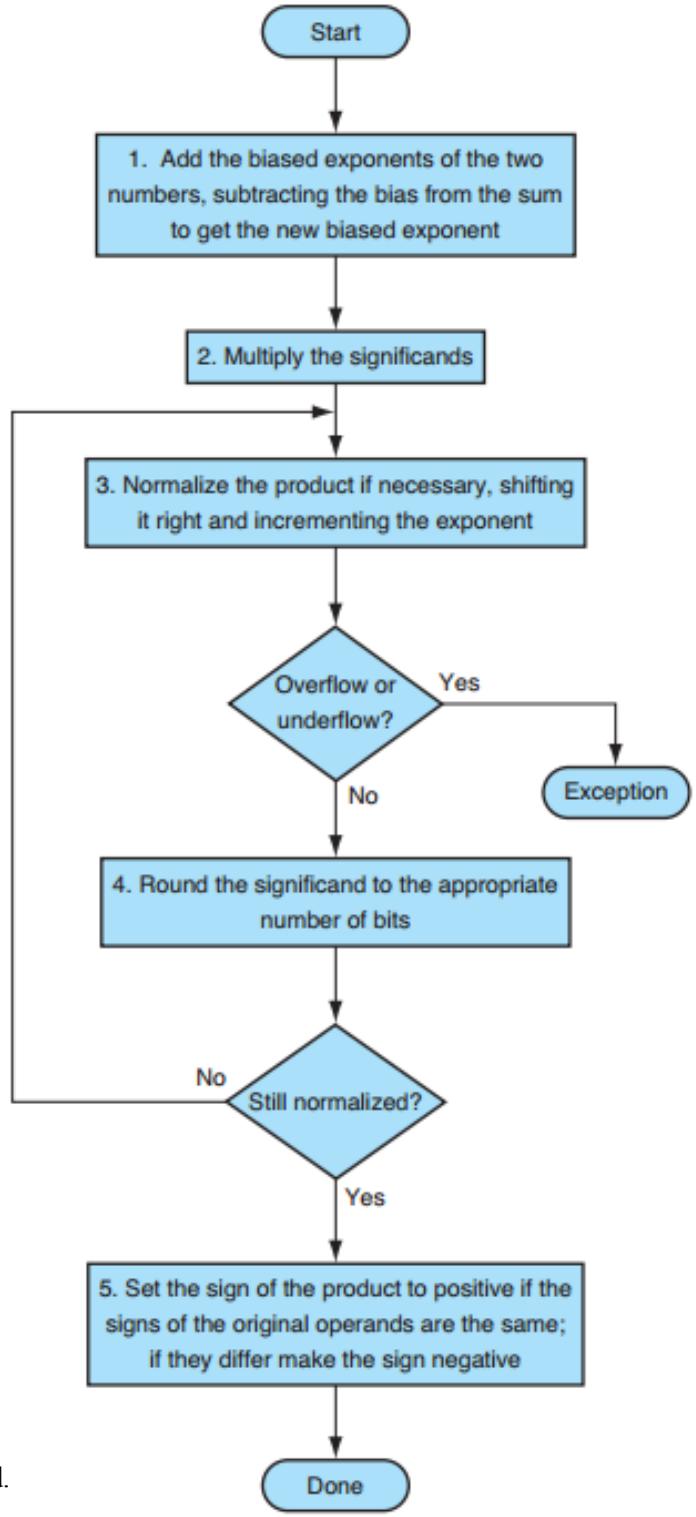
Số thực dấu chấm động

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 phép nhân số trong hệ thập phân.

Nhưng lưu ý: Bước 1 khi cộng hai exponent của hai số, **nhớ trừ đi số *bias***

- *Nếu IEEE 754 độ chính xác đơn: bias = 127*
- *Nếu IEEE 754 độ chính xác kép: bias = 1023*



Số thực dấu chấm động

Phép nhân trên số thực dấu chấm động

Ví dụ: nhân số thực dấu chấm động trên hệ nhị phân cho 2 số sau:

$$\mathbf{0.5_{10}}\ \textbf{và}\ \mathbf{-0.4375_{10}}.$$

Biết các số dấu chấm động dùng lưu trữ theo IEEE 754 độ chính xác đơn, nhưng phần significant chỉ cho phép 4 bits

Số thực dấu chấm động

Phép nhân trên số thực dấu chấm động

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:

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

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 thực dấu chấm động

Phép nhân trên số thực dấu chấm động

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}.

COMPUTER ENGINEERING

Số thực dấu chấm động

Sinh viên tìm hiểu:

- **Phép chia với số floating-point trong MIPS**
- **Các lệnh làm việc với số floating-point trong MIPS**
  - Các lệnh liên quan đến số floating-point
  - Phân biệt các lệnh:

    mult, multu, mul.s, mul.d

    div, divu, div.s, div.d

PHÉP TOÁN SỐ HỌC TRÊN MÁY TÍNH

- **Lý thuyết: Đọc sách tham khảo**
  - Mục: 3.5
  - Sách: *Computer Organization and Design: The Hardware/Software Interface*, Patterson, D. A., and J. L. Hennessy, Morgan Kaufman, Revised Fourth Edition, 2011.

- **Bài tập:** file đính kèm

Tổng kết:

- Hiểu cách biểu diễn số thực dấu chấm động theo IEEE 754 trong máy tính theo:
  - ✓ Độ chính xác đơn
  - ✓ Độ chính xác kép
- Hiểu cách máy tính thực hiện cộng, trừ, nhân chia trên số thực dấu chấm động
- Hiểu cách thiết kế một mạch cộng hai số thực dấu chấm động cơ bản