



# KIẾN TRÚC MÁY TÍNH

KHOA HỌC & KỸ THUẬT MÁY TÍNH



Võ Tấn Phương

<http://www.cse.hcmut.edu.vn/vtphuong>

# Chapter 4.0

## Các phép số học

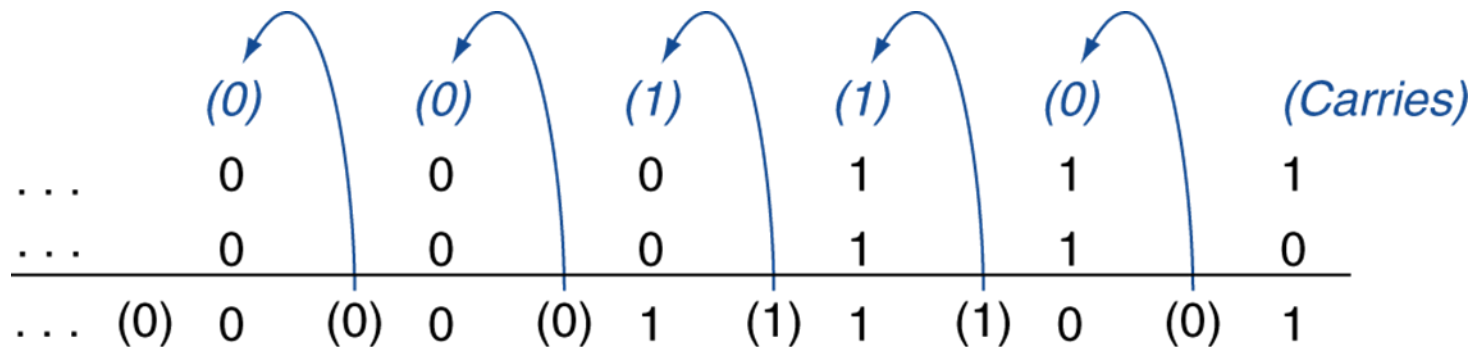
# Các phép số học

- ❖ Các phép tính trên số nguyên
  - ✧ Cộng và Trừ
  - ✧ Nhân và Chia
  - ✧ Xử lý tràn
- ❖ Số thực với dấu chấm di động (Floating-Point)
  - ✧ Cách biểu diễn và các phép tính

# Nhắc lại mạch số

Môn học:

- ❖ Nhập môn điện toán (Năm I)
- ❖ Thiết kế hệ thống số

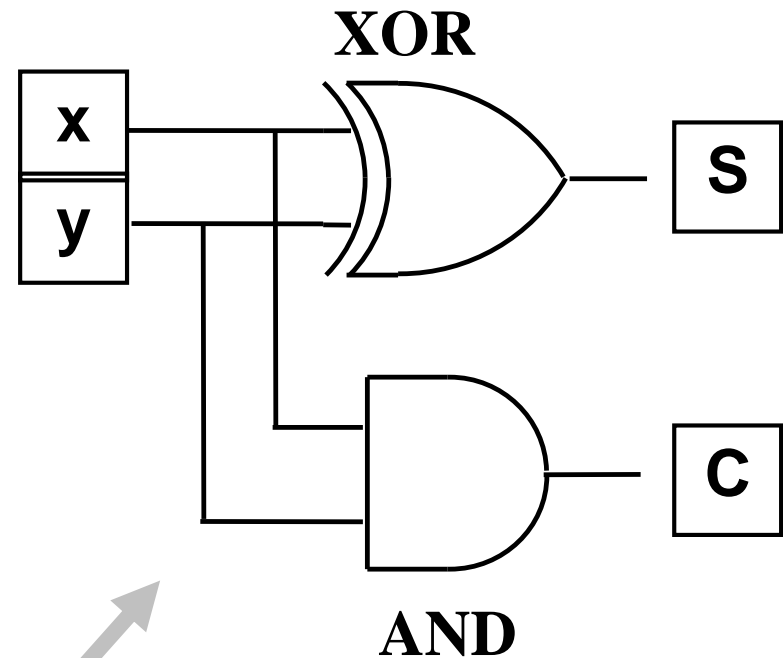


# Mạch Half Adder

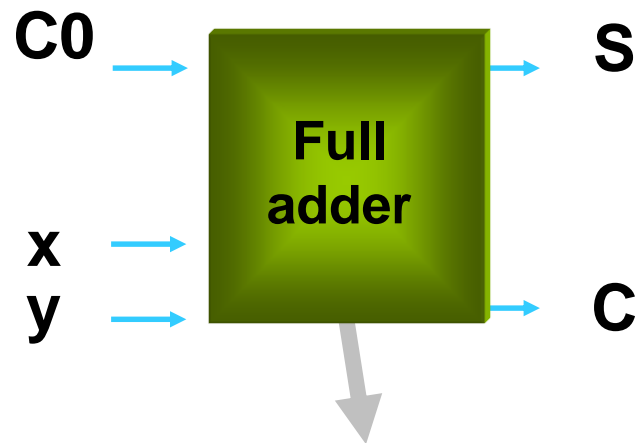


XOR AND

x	y	S	C
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1



# Mạch Full Adder



$$S = x + y + C0$$

$$S = (x + y) + C0$$

$$\text{Tính: } S1 = x + y$$

$$\text{Tính: } S2 = S1 + C0$$

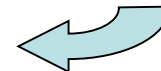
Half adder 1

Half adder 2

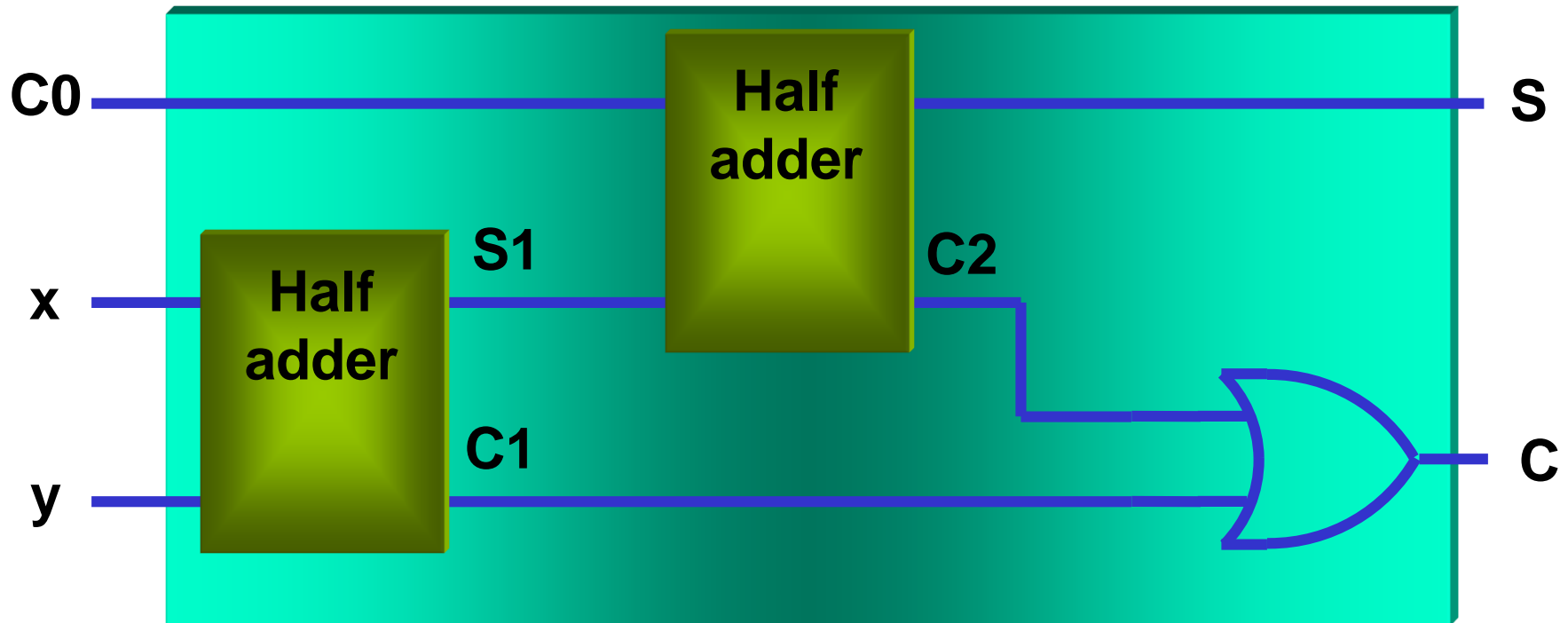
# Full adder (2)

$C_0$	x	y	S	C	$C_0$	$S_1$	$C_1$	$C_2$	C
0	0	0	0	0	0	0	0	0	0
0	0	1	1	0	0	1	0	0	0
0	1	0	1	0	0	1	0	0	0
0	1	1	0	1	0	0	1	0	1
1	0	0	1	0	1	0	0	0	0
1	0	1	0	1	1	1	0	1	1
1	1	0	0	1	1	1	0	1	1
1	1	1	1	1	1	0	1	0	1

**$C = 1$  when  $C_1 = 1$  or  $C_2 = 1$**

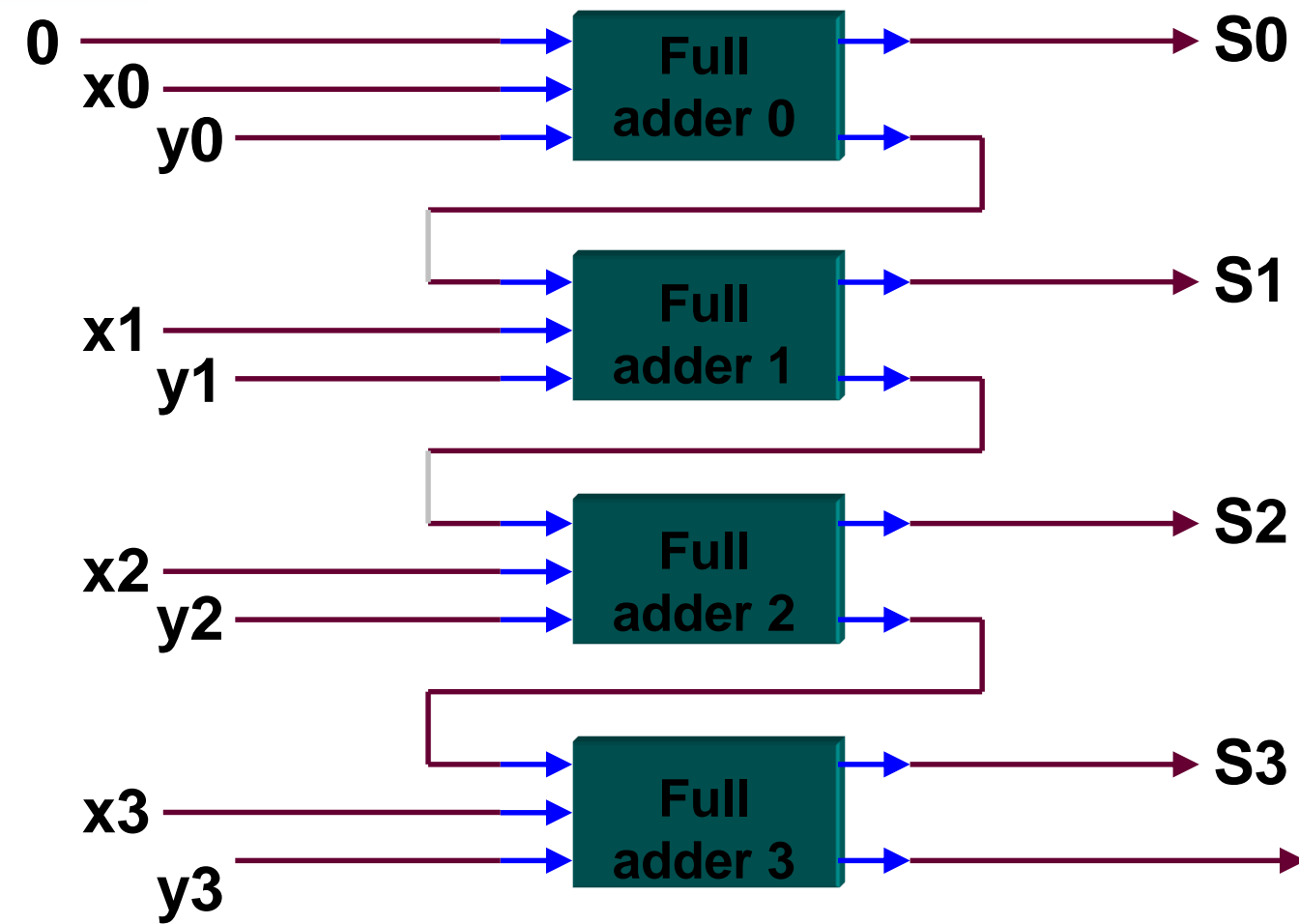


# Full adder (3)





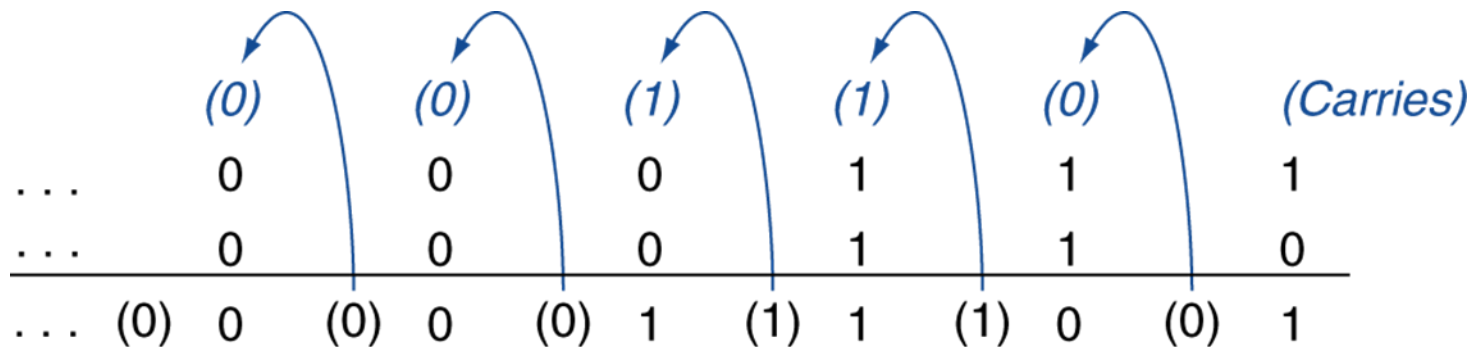
# Cộng nhiều Bits



$$\begin{array}{r} x_3x_2x_1x_0 \\ + y_3y_2y_1y_0 \\ \hline C S_3S_2S_1S_0 \end{array}$$

# Phép cộng số nguyên

❖ Ví dụ:  $7 + 6$



- Tràn nếu kết quả tràn ngưỡng
  - Cộng 2 toán hạng trái dấu: không tràn
  - Cộng 2 toán hạng đều dương
    - Tràn nếu bit dấu của kết quả là 1
  - Cộng 2 toán hạng đều âm
    - Tràn nếu bit dấu của kết quả là 0

# Phép trừ số nguyên

❖ Cộng số âm của toán hạng thứ 2

❖ Ví dụ:  $7 - 6 = 7 + (-6)$

$$\begin{array}{r} +7: \quad 0000 \ 0000 \ \dots \ 0000 \ 0111 \\ -6: \quad 1111 \ 1111 \ \dots \ 1111 \ 1010 \\ \hline +1: \quad 0000 \ 0000 \ \dots \ 0000 \ 0001 \end{array}$$

❖ Tràn nếu kết quả vượt ngưỡng

- ✧ Phép trừ 2 toán hạng cùng dấu, không bao giờ tràn
- ✧ Trừ 1 toán hạng âm với 1 toán hạng dương
  - Tràn nếu bit dấu của kết quả là 0
- ✧ Trừ 1 toán hạng dương với 1 toán hạng âm
  - Tràn nếu bit dấu của kết quả là 1

- ❖ Một số ngôn ngữ (như C) không xử lý tràn
  - ✧ Sử dụng lệnh MIPS: `addu`, `addui`, `subu`
- ❖ Các ngôn ngữ khác (như Ada, Fortran) yêu cầu xử lý tràn bằng ngoại lệ
  - ✧ Sử dụng lệnh MIPS: `add`, `addi`, `sub`
  - ✧ Khi có tràn, bẫy bằng ngoại lệ & xử lý:
    - Cất PC vào thanh ghi exception PC (EPC)
    - Nhảy đến chương trình xử lý tràn
    - Dùng `mfc0` khôi phục giá trị EPC value, trở về sau khi xử lý tràn

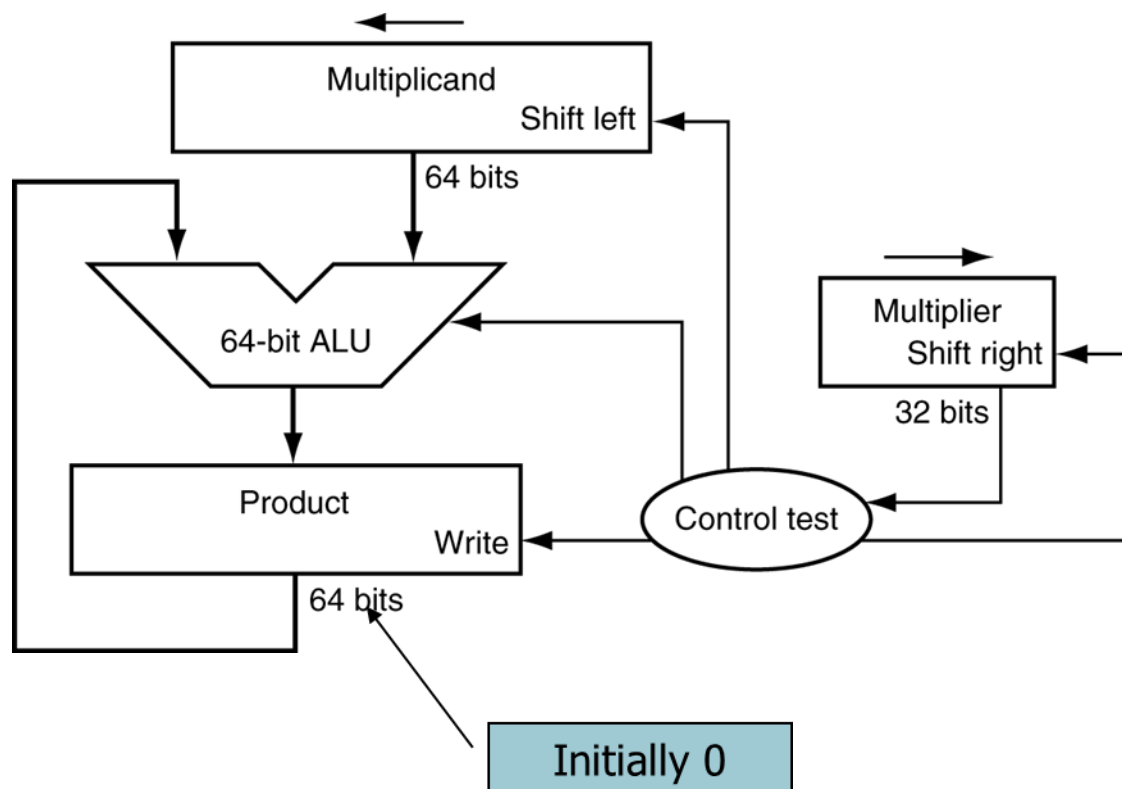
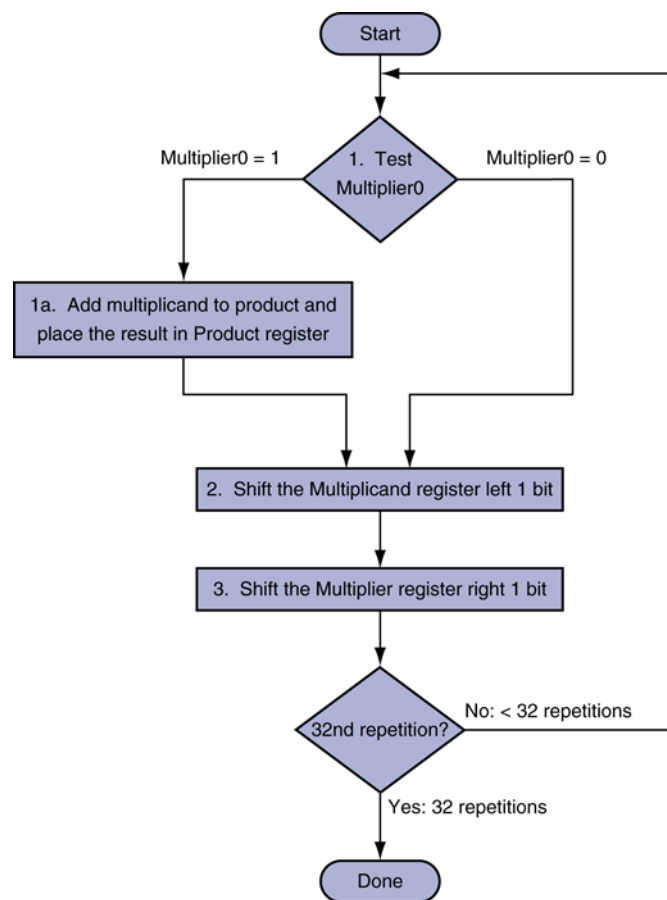
# Phép nhân

❖ Bắt đầu bằng phép nhân thuần túy

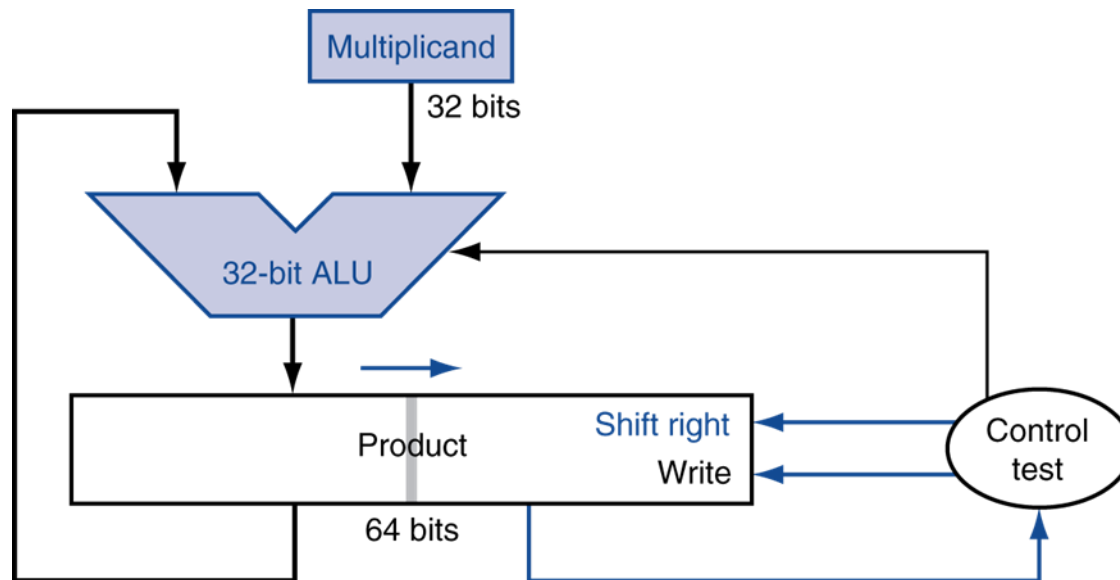
$$\begin{array}{r} \text{multiplicand} \quad 1000 \\ \text{multiplier} \quad \times 1001 \\ \hline 1000 \\ 0000 \\ 0000 \\ 1000 \\ \hline \text{product} \quad 1001000 \end{array}$$

Length of product is  
the sum of operand  
lengths

# Phần cứng thực hiện nhân

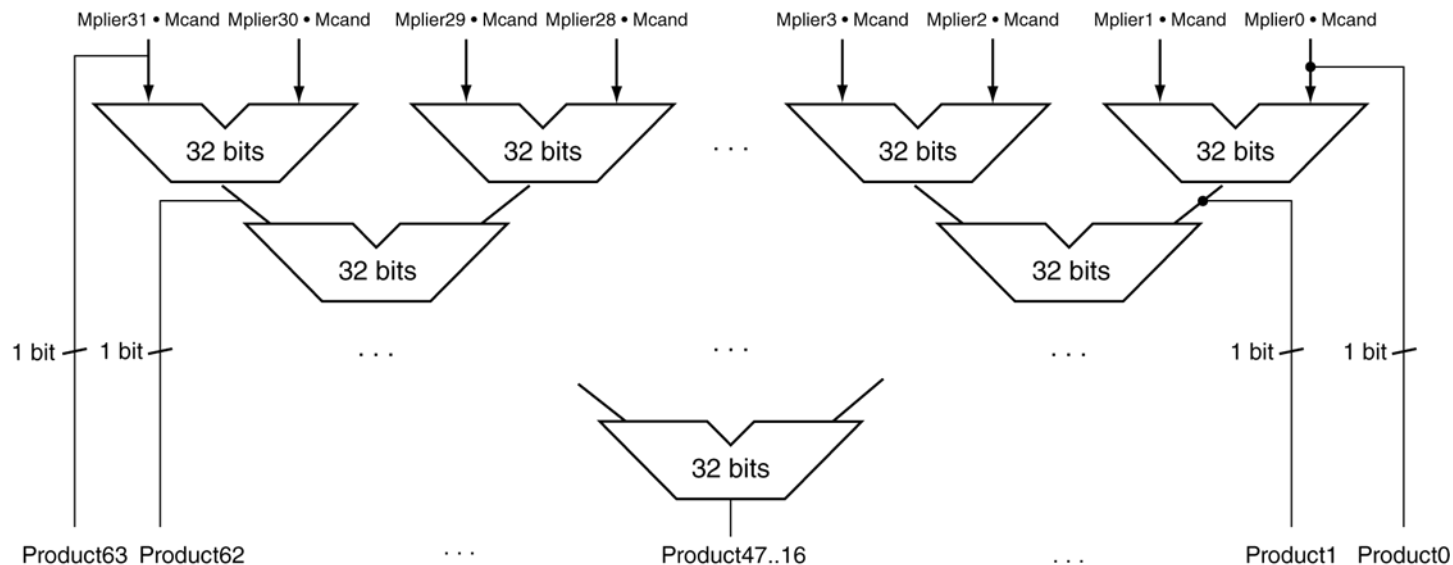


## ❖ Các bước song song: add/shift



- Một chu kỳ cho mỗi phép cộng (tích thành phần)
  - Có thể chấp nhận khi tần xuất thấp

- ❖ Sử dụng nhiều bộ cộng cùng lúc
- ✧ Cost/performance tradeoff



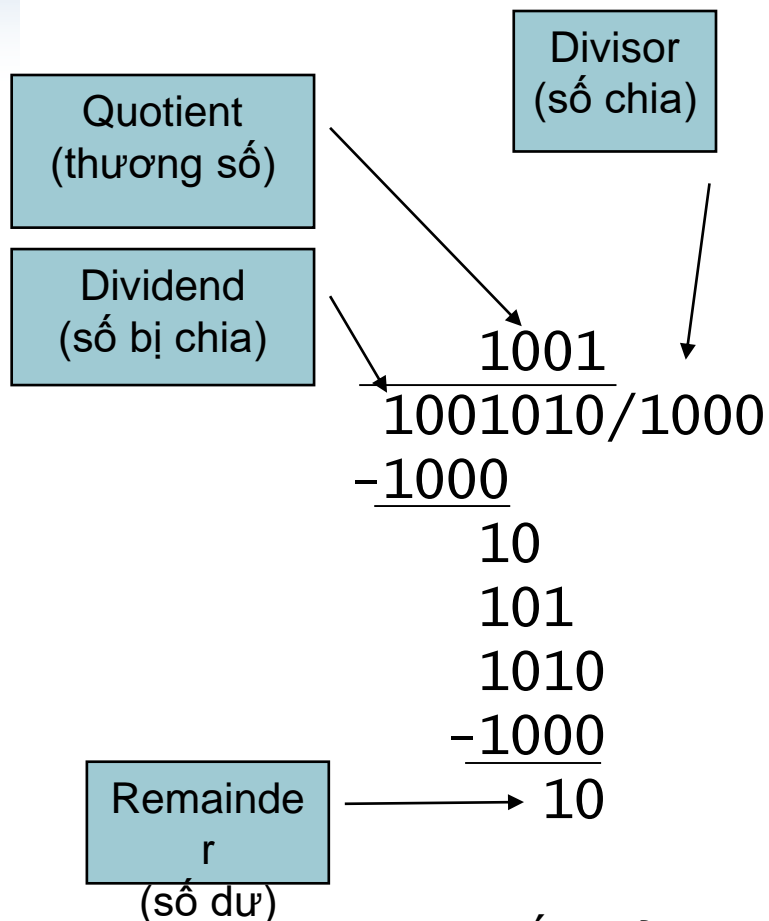
- Có thể thực hiện theo cơ chế ống
  - Nhiều tác vụ nhân thực hiện cùng lúc



# Lệnh nhân trong MIPS

- ❖ Kết quả sẽ là 64-bit, chứa trong 2 thanh ghi 32-bit
  - ✧ HI: chứa 32-bit cao
  - ✧ LO: chứa 32-bit thấp
- ❖ Lệnh nhân
  - ✧ `mult rs, rt` / `multu rs, rt`
    - 64-bit kết quả chứa trong HI/LO
  - ✧ `mflo rd` / `mfi rd`
    - Chuyển từ HI/LO vào rd
    - Có thể kiểm tra giá trị HI xem kết quả phép nhân có tràn?
  - ✧ `mul rd, rs, rt`
    - 32 bits thấp của kết quả phép nhân → rd

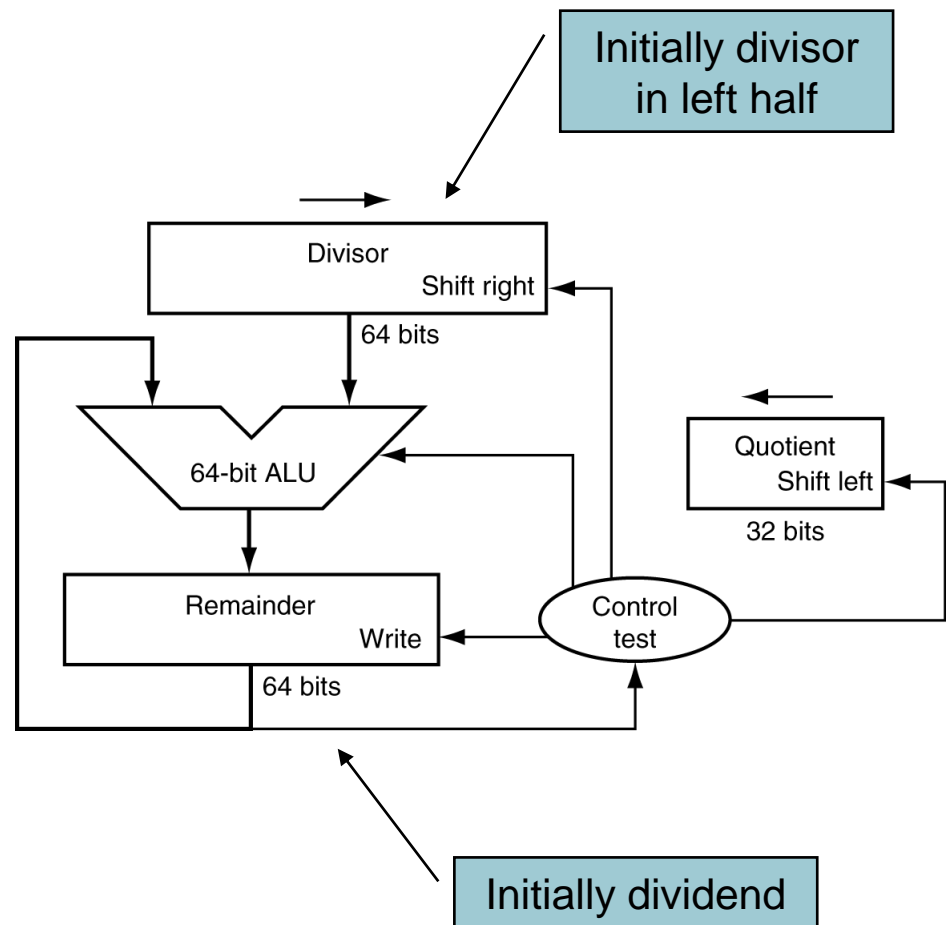
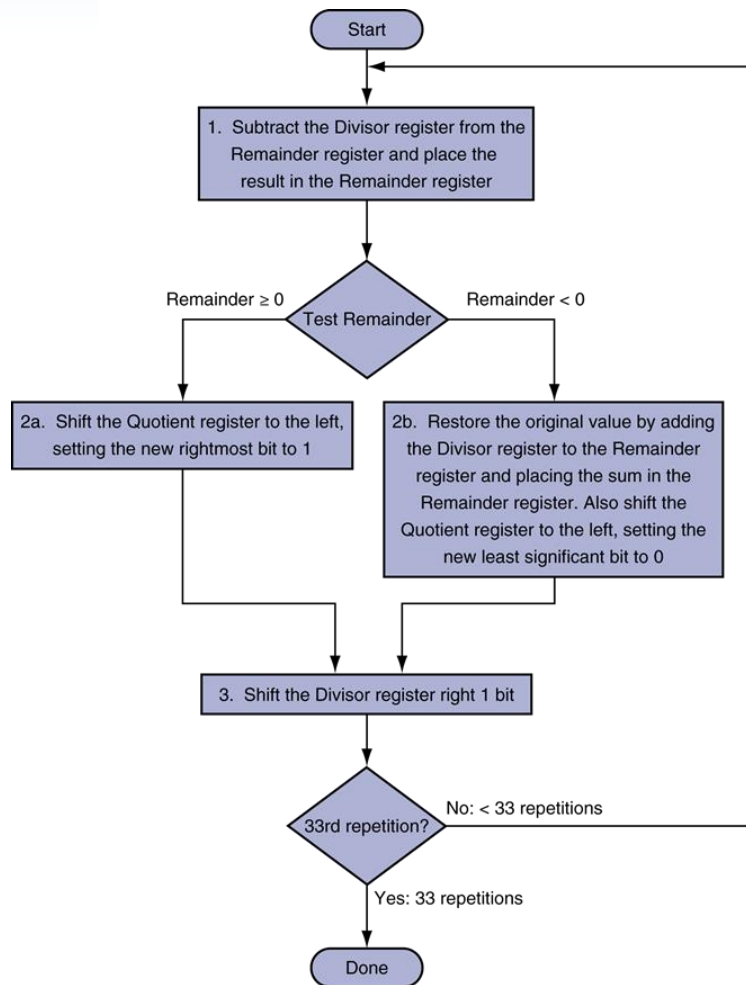
# Phép chia



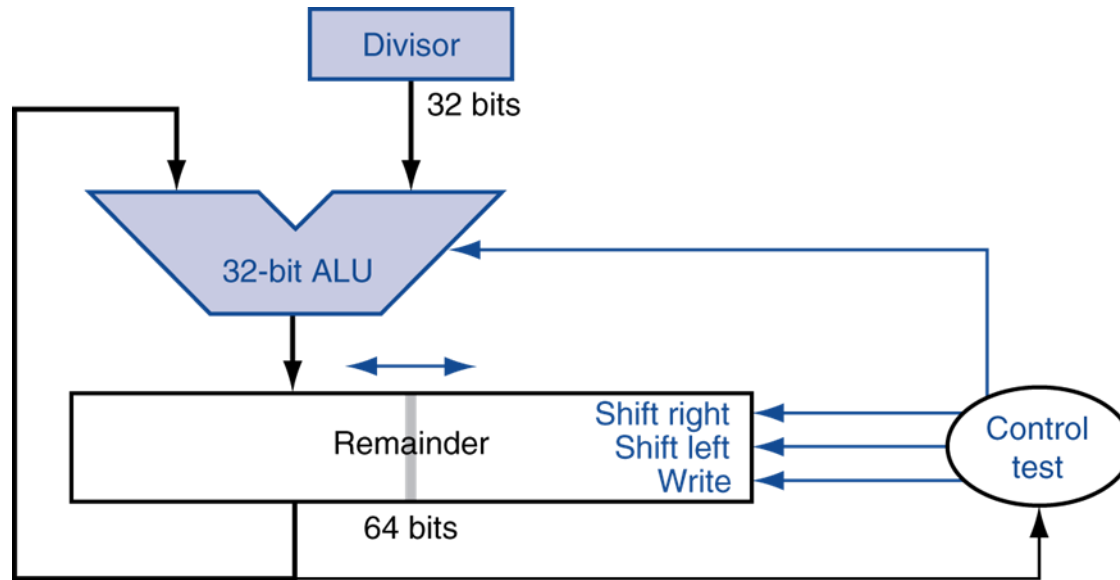
Toán hạng  $n$ -bit cho kết quả  $n$ -bit thương số và số dư

- ❖ Kiểm tra chia 0 → báo lỗi
- ❖ Long division approach
  - ✧ If divisor  $\leq$  dividend bits
    - 1 bit in quotient, subtract
  - ✧ Otherwise
    - 0 bit in quotient, bring down next dividend bit
- ❖ Restoring division
  - ✧ Do the subtract, and if remainder goes  $< 0$ , add divisor back
- ❖ Signed division
  - ✧ Divide using absolute values
  - ✧ Adjust sign of quotient and remainder as required

# Phần cứng thực hiện chia



# Bộ chia cải thiện



- Một chu kỳ cho mỗi phép trừ thành phần
- Tương tự rất nhiều với bộ nhân
  - Có thể dùng cùng một phần cứng cho cả 2

# Bộ chia nhanh

- ❖ Không thể thực hiện song song như trong bộ nhân
  - ✧ Dấu trong mỗi phép trừ thành phần là điều kiện
- ❖ Có thể tạo bộ chia nhanh (e.g. SRT division)

# Lệnh chia trong MIPS

- ❖ Thanh ghi HI/LO chứa kết quả phép chia
  - ✧ HI: 32-bit số dư (remainder)
  - ✧ LO: 32-bit (kết quả) quotient
- ❖ Lệnh trong MIP
  - ✧ `div rs, rt` / `divu rs, rt`
  - ✧ Không kiểm tra tràn hoặc lỗi /0
    - Nếu có yêu cầu, phần mềm phải tự thực hiện
  - ✧ Sử dụng lệnh `mfhi`, `mflo` để lấy kết quả

❖ Biểu diễn các số khác số nguyên (số thực)

✧ Bao gồm cả số rất nhỏ lẫn số rất lớn

❖ Giống như biểu diễn số trong khoa học

✧  $-2.34 \times 10^{56}$

✧  $+0.002 \times 10^{-4}$

✧  $+987.02 \times 10^9$

normalized

not normalized

❖ Kiểu nhị phân

✧  $\pm 1.xxxxxxx_2 \times 2^{yyyy}$

❖ Kiểu float và double trong ngôn ngữ C

# Chuẩn của hệ thống số chấm di động

- ❖ Định chuẩn bởi Tổ chức IEEE(754-1985)
- ❖ Được phát triển nhằm đáp ứng tiêu chuẩn trình bày thống nhất
  - ✧ Dễ sử dụng và chuyển đổi giữa các bộ mã trong khoa học
- ❖ Hiện nay trở thành thông dụng
- ❖ Tồn tại 2 cách biểu diễn
  - ✧ Chính xác đơn(32-bit)
  - ✧ Chính xác kép (64-bit)



# Dạng định chuẩn theo IEEE

single: 8 bits  
double: 11 bits

single: 23 bits  
double: 52 bits

S	Exponent	Fraction
---	----------	----------

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

- S: bit dấu (0  $\Rightarrow$  (+) , 1  $\Rightarrow$  (-))
- Normalize significand:  $1.0 \leq |\text{significand}| < 2.0$ 
  - Luôn có 1 bit trước dấu chấm, nên bit này thường ẩn
  - Significand is Fraction with the “1.” restored
- Exponent: excess representation: actual exponent + Bias
  - Ensures exponent is unsigned
  - Single: Bias = 127; Double: Bias = 1203

# Tầm giá trị với độ chính xác đơn

- ❖ Giá trị (Exponents) 00000000 và 11111111 : dự trữ
- ❖ Giá trị nhỏ nhất
  - ✧ Số mũ: 00000001  
 $\Rightarrow$  số mũ thực chất sẽ là  $= 1 - 127 = -126$
  - ✧ Fraction: 000...00  $\Rightarrow$  significand = 1.0
  - ✧  $\pm 1.0 \times 2^{-126} \approx \pm 1.2 \times 10^{-38}$
- ❖ Giá trị lớn nhất:
  - ✧ Số mũ: 11111110  
 $\Rightarrow$  số mũ thực tế sẽ là  $= 254 - 127 = +127$
  - ✧ Fraction: 111...11  $\Rightarrow$  significand  $\approx 2.0$
  - ✧  $\pm 2.0 \times 2^{+127} \approx \pm 3.4 \times 10^{+38}$

# Mức độ chính xác

## ❖ Mang tính tương đối

- ✧ Xác định bởi các bit fraction

- ✧ Đơn: khoảng  $2^{-23}$

- Tương đương với  $23 \times \log_{10} 2 \approx 23 \times 0.3 \approx 6$ :  
chính xác đến 6 số (hệ thập phân)

- ✧ Kép: khoảng  $2^{-52}$

- Tương đương với  $52 \times \log_{10} 2 \approx 52 \times 0.3 \approx 16$ :  
chính xác đến 16 số (hệ thập phân)

# Ví dụ: Dấu chấm di động

❖ Biểu diễn số thực thập phân:  $-0.75$

✧  $-0.75 = (-1)^1 \times 1.1_2 \times 2^{-1}$

✧  $S = 1$

✧ Fraction =  $1000\dots00_2$

✧ Exponent =  $-1 + \text{Bias}$

▪ Đơn:  $-1 + 127 = 126 = 01111110_2$

▪ Kép:  $-1 + 1023 = 1022 = 01111111110_2$

❖ Single:  $1011111101000\dots00$

❖ Double:  $1011111111101000\dots00$

# Ví dụ: (tt.)

- ❖ Cho biết số thực thập phân của một số biểu diễn bằng dấu chấm di động (đơn) sau:

11000000101000...00

✧  $S = 1$

✧ Fraction =  $01000...00_2$

✧ Exponent =  $10000001_2 = 129$

❖  $x = (-1)^1 \times (1 + 01_2) \times 2^{(129 - 127)}$   
 $= (-1) \times 1.25 \times 2^2$   
 $= -5.0$

# Số vô hạn (Infinities) và Số không hợp lệ (NaNs)

- ❖ Exponent = 111...1, Fraction = 000...0
  - ✧  $\pm$ Infinity
  - ✧ Dùng để kiểm tra kết quả của phép tính
- ❖ Exponent = 111...1, Fraction  $\neq$  000...0
  - ✧ Not-a-Number (NaN)
  - ✧ Số không hợp lệ
    - Ví dụ: chia cho zero:  $0.0 / 0.0$
  - ✧ Dùng để kiểm tra kết quả của phép tính

❖ Giả sử có phép cộng 2 số thập phân (4 ký số)

✧  $9.999 \times 10^1 + 1.610 \times 10^{-1}$

❖ 1. Điều chỉnh dấu chấm

✧ Dời số mũ của số nhỏ hơn cho đồng số mũ

✧  $9.999 \times 10^1 + 0.016 \times 10^1$

❖ 2. Cộng hệ số

✧  $9.999 \times 10^1 + 0.016 \times 10^1 = 10.015 \times 10^1$

❖ 3. Chuẩn hóa kết quả & kiểm tra ngưỡng

✧  $1.0015 \times 10^2$

❖ 4. Làm tròn và điều chỉnh nếu cần thiết

✧  $1.002 \times 10^2$

# Cộng nhị phân

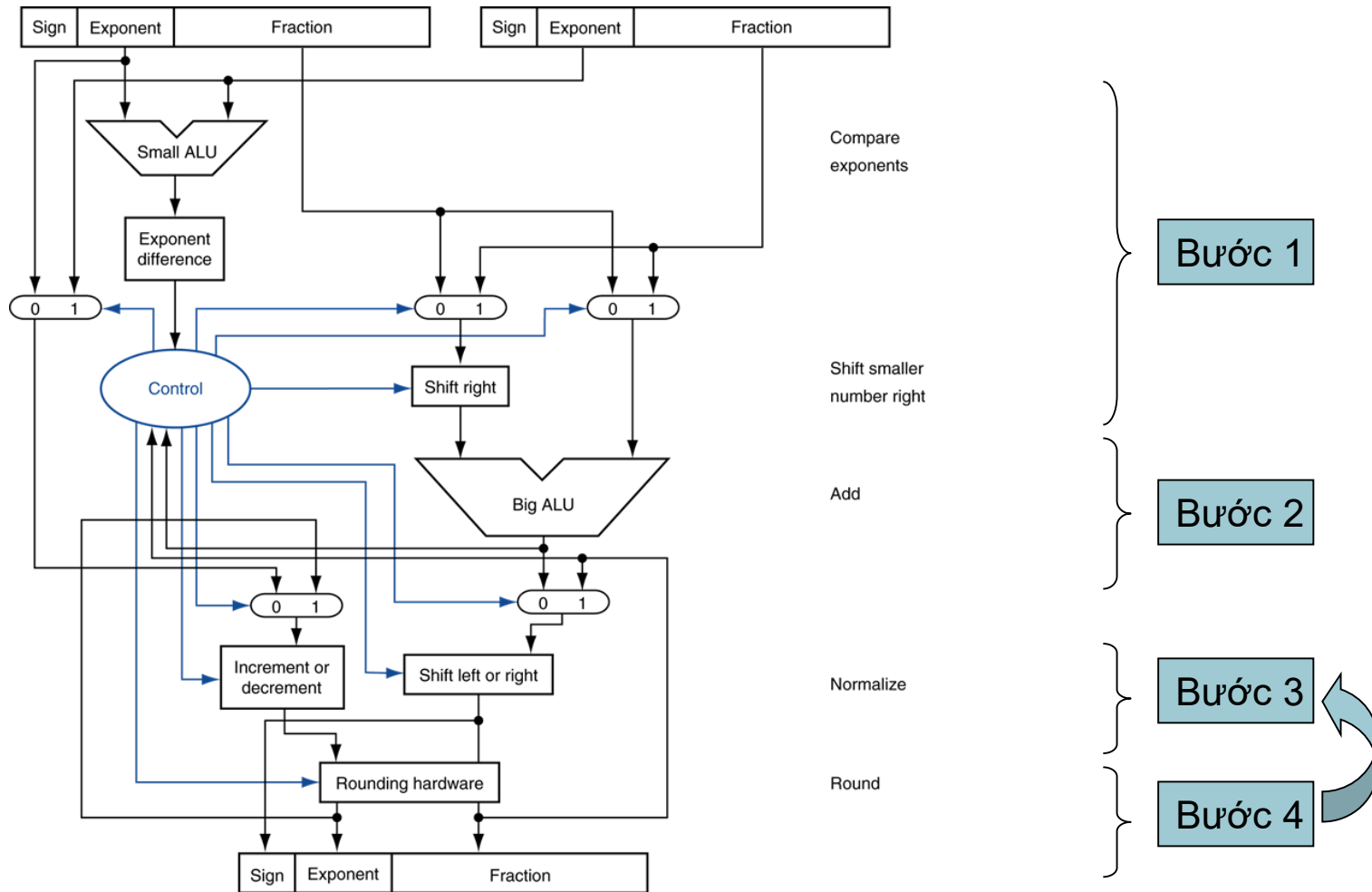
- ❖ Giả sử cộng 2 số nhị phân (4 ký số):
  - ✧  $1.000_2 \times 2^{-1} + -1.110_2 \times 2^{-2}$  ( $0.5 + -0.4375$ )
- ❖ 1. Điều chỉnh dấu chấm
  - ✧ Dời số mũ của số nhỏ hơn cho đồng số mũ
  - ✧  $1.000_2 \times 2^{-1} + -0.111_2 \times 2^{-1}$
- ❖ 2. Cộng hệ số
  - ✧  $1.000_2 \times 2^{-1} + -0.111_2 \times 2^{-1} = 0.001_2 \times 2^{-1}$
- ❖ 3. Chuẩn hóa kết quả & kiểm tra ngưỡng
  - ✧  $1.000_2 \times 2^{-4}$ , (nằm trong ngưỡng cho phép)
- ❖ 4. Làm tròn và điều chỉnh nếu cần thiết
  - ✧  $1.000_2 \times 2^{-4}$  (không cần điều chỉnh) = 0.0625



# Phần cứng bộ cộng (FP)

- ❖ Phức tạp hơn rất nhiều so với bộ cộng số nguyên
- ❖ Nếu thực hiện trong 1 chu kỳ đồng hồ → Chu kỳ quá dài
  - ✧ Dài hơn nhiều so với các phép cộng số nguyên
  - ✧ Kéo dài thời gian xung đồng hồ → ảnh hưởng đến các lệnh khác
- ❖ Bộ cộng (FP) thường kéo dài nhiều chu kỳ
  - ✧ Có thể cải thiện bằng cơ chế ống

# Phần cứng bộ cộng (FP)



# Phép nhân thập phân

❖ Giả sử nhân 2 số thập phân (4 ký số)

✧  $1.110 \times 10^{10} \times 9.200 \times 10^{-5}$

❖ 1. Cộng số mũ

✧ Nếu dùng số mũ biased, trừ biased vào tổng

✧ Số mũ mới là  $= 10 + -5 = 5$

❖ 2. Nhân hệ số

✧  $1.110 \times 9.200 = 10.212 \Rightarrow 10.212 \times 10^5$

❖ 3. Chuẩn hóa kết quả & kiểm tra ngưỡng

✧  $1.0212 \times 10^6$

❖ 4. Làm tròn và điều chỉnh nếu cần thiết

❖ 5. Xác định dấu của kết quả

✧  $+1.021 \times 10^6$

# Phép nhân nhị phân (FP)

- ❖ Giả sử nhân 2 số thập phân (4 ký số)
  - ✧  $1.000_2 \times 2^{-1} \times -1.110_2 \times 2^{-2} (0.5 \times -0.4375)$
- ❖ 1. Cộng số mũ
  - ✧ Unbiased:  $-1 + -2 = -3$
  - ✧ Biased:  $(-1 + 127) + (-2 + 127) = -3 + 254 - 127 = -3 + 127$
- ❖ 2. Nhân hệ số
  - ✧  $1.000_2 \times 1.110_2 = 1.110_2 \Rightarrow 1.110_2 \times 2^{-3}$
- ❖ 3. Chuẩn hóa kết quả & kiểm tra ngưỡng
  - ✧  $1.110_2 \times 2^{-3}$  (không đổi: nằm trong ngưỡng cho phép)
- ❖ 4. Làm tròn và điều chỉnh nếu cần thiết
  - ✧  $1.110_2 \times 2^{-3}$  (no change)
- ❖ 5. Xác định dấu:  $(+) \times (-) \Rightarrow (-)$ 
  - ✧  $-1.110_2 \times 2^{-3} = -0.21875$

# Phần cứng Bộ số học (FP)

- ❖ Bộ nhân (FP) và Bộ cộng (FP) có độ phức tạp như nhau
  - ✧ Chỉ khác nhau cho phép tính hệ số
- ❖ Phần cứng Bộ số học thường thực hiện các tác vụ sau:
  - ✧ Cộng, Trừ, Nhân, Chia, Căn, Nghịch đảo
  - ✧ Chuyển đổi FP  $\leftrightarrow$  integer
- ❖ Các tác vụ này thường kéo dài trong nhiều chu kỳ xung đồng hồ
  - ✧ Cải thiện bằng cơ chế đường ống

# Lệnh FP trong MIPS

- ❖ Phần cứng bộ FP là một coprocessor
  - ✧ Mở rộng kiến trúc tập lệnh
- ❖ Có các thanh ghi FP riêng
  - ✧ 32 thanh ghi (đơn): \$f0, \$f1, ... \$f31
  - ✧ Chính xác kép bằng cách ghép: \$f0/\$f1, \$f2/\$f3, ..
    - Phiên bản 2 của MIPS ISA hỗ trợ 32 × 64-bit FP reg's
- ❖ Các lệnh FP chỉ thực hiện trên các thanh ghi FP
  - ✧ Chương trình thường không thực hiện các phép số nguyên trên dữ liệu FP hoặc ngược lại
  - ✧ Thanh ghi riêng không làm phức tạp thêm code
- ❖ Các lệnh FP load và store
  - ✧ lwc1, ldc1, swc1, sdc1
    - Ví dụ: ldc1 \$f8, 32(\$sp)

# Lệnh FP trong MIPS

## ❖ Phép tính số học (đơn)

- ✧ `add.s`, `sub.s`, `mul.s`, `div.s`
  - Ví dụ: `add.s $f0, $f1, $f6`

## ❖ Phép tính số học (kép)

- ✧ `add.d`, `sub.d`, `mul.d`, `div.d`
  - Ví dụ: `mul.d $f4, $f4, $f6`

## ❖ Lệnh so sánh (đơn/kép)

- ✧ `c.xx.s`, `c.xx.d` (`xx` is `eq`, `lt`, `le`, ...)
- ✧ Gán hoặc xóa bit điều kiện code
  - e.g. `c.lt.s $f3, $f4`

## ❖ Rẽ nhánh theo điều kiện

- ✧ `bc1t`, `bc1f`
  - Ví dụ: `bc1t TargetLabel`

# Ví dụ: Chuyển °F sang °C

## ❖ C code:

```
float f2c (float fahr) {  
    return ((5.0/9.0)*(fahr - 32.0));  
}
```

✧ fahr chứa trong \$f12, kết quả trong \$f0, hằng số trong bộ nhớ toàn cục

## ❖ Biên dịch thành MIPS code:

```
f2c: lwc1    $f16, const5($gp)  
     lwc2    $f18, const9($gp)  
     div.s   $f16, $f16, $f18  
     lwc1    $f18, const32($gp)  
     sub.s   $f18, $f12, $f18  
     mul.s   $f0, $f16, $f18  
     jr      $ra
```



# Ví dụ: Nhân Ma trận

❖  $X = X + Y \times Z$

✧ Tất cả đều là ma trận  $32 \times 32$ , các phần tử của ma trận 64-bit (chính xác kép)

❖ C code:

```
void mm (double x[][],  
         double y[][], double z[][]) {  
    int i, j, k;  
    for (i = 0; i != 32; i = i + 1)  
        for (j = 0; j != 32; j = j + 1)  
            for (k = 0; k != 32; k = k + 1)  
                x[i][j] = x[i][j]  
                    + y[i][k] * z[k][j];  
}
```

✧ Địa chỉ của x, y, z chứa trong \$a0, \$a1, \$a2, và i, j, k trong \$s0, \$s1, \$s2

# Ví dụ: Nhân Ma trận (tt.)

## ■ MIPS code:

```
li    $t1, 32          # $t1 = 32 (row size/loop end)
li    $s0, 0           # i = 0; initialize 1st for loop
L1: li    $s1, 0        # j = 0; restart 2nd for loop
L2: li    $s2, 0        # k = 0; restart 3rd for loop
sll   $t2, $s0, 5       # $t2 = i * 32 (size of row of x)
addu  $t2, $t2, $s1     # $t2 = i * size(row) + j
sll   $t2, $t2, 3       # $t2 = byte offset of [i][j]
addu  $t2, $a0, $t2     # $t2 = byte address of x[i][j]
l.d   $f4, 0($t2)      # $f4 = 8 bytes of x[i][j]
L3: sll   $t0, $s2, 5    # $t0 = k * 32 (size of row of z)
addu  $t0, $t0, $s1     # $t0 = k * size(row) + j
sll   $t0, $t0, 3       # $t0 = byte offset of [k][j]
addu  $t0, $a2, $t0     # $t0 = byte address of z[k][j]
l.d   $f16, 0($t0)     # $f16 = 8 bytes of z[k][j]
...
```

# Ví dụ: Nhân Ma trận (tt.)

```
...  
sll    $t0, $s0, 5           # $t0 = i*32 (size of row of y)  
addu   $t0, $t0, $s2         # $t0 = i*size(row) + k  
sll    $t0, $t0, 3           # $t0 = byte offset of [i][k]  
addu   $t0, $a1, $t0         # $t0 = byte address of y[i][k]  
l.d    $f18, 0($t0)          # $f18 = 8 bytes of y[i][k]  
mul.d  $f16, $f18, $f16      # $f16 = y[i][k] * z[k][j]  
add.d  $f4, $f4, $f16        # f4=x[i][j] + y[i][k]*z[k][j]  
addiu  $s2, $s2, 1           # $k k + 1  
bne    $s2, $t1, L3          # if (k != 32) go to L3  
s.d    $f4, 0($t2)           # x[i][j] = $f4  
addiu  $s1, $s1, 1           # $j = j + 1  
bne    $s1, $t1, L2          # if (j != 32) go to L2  
addiu  $s0, $s0, 1           # $i = i + 1  
bne    $s0, $t1, L1          # if (i != 32) go to L1
```

# Kết luận

- ❖ ISAs hỗ trợ phép số học
  - ✧ Số nguyên có dấu và không dấu
  - ✧ Floating-point approximation to reals
- ❖ Bounded range and precision
  - ✧ Operations can overflow and underflow
- ❖ MIPS ISA
  - ✧ Core instructions: 54 most frequently used
    - 100% of SPECINT, 97% of SPECFP
  - ✧ Other instructions: less frequent