Computer Architecture and Organization

INSTRUCTOR: YAN-TSUNG PENG

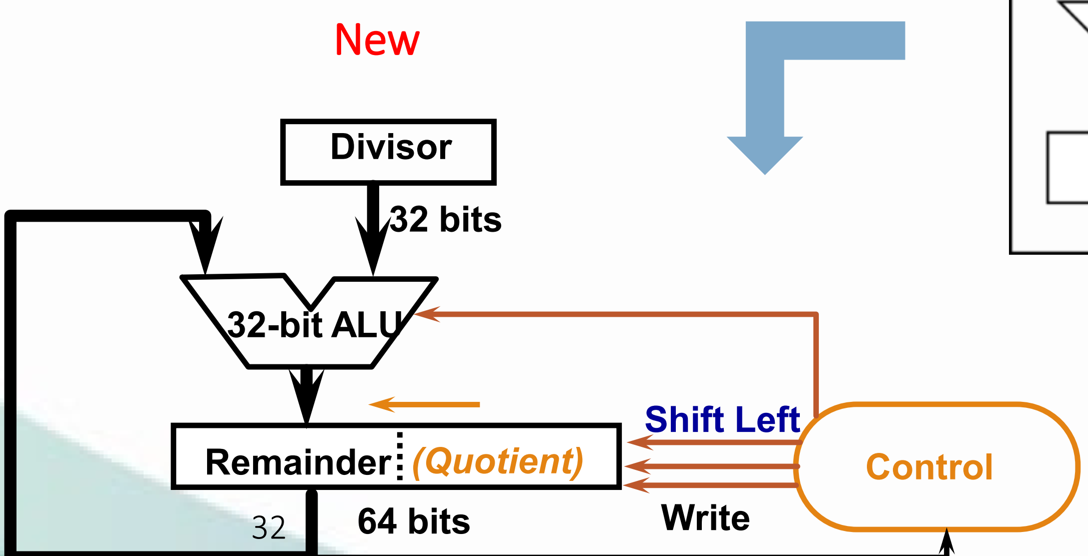
DEPT. OF COMPUTER SCIENCE, NCCU

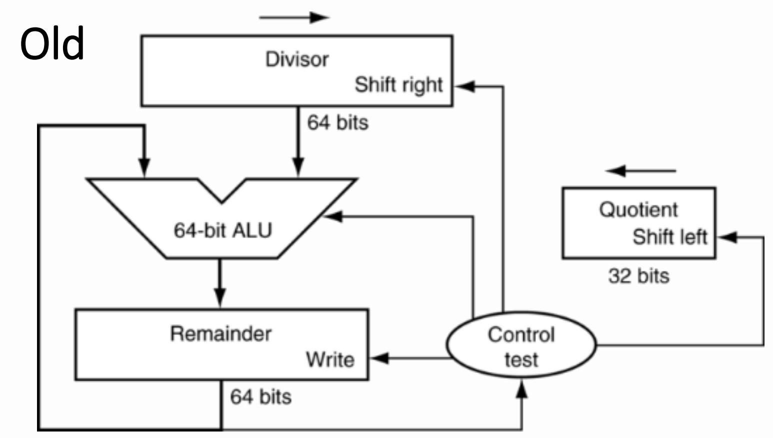
Improved Version of Division Hardware

- Half of the bits in divisor register are always 0
  => 1/2 of 64-bit adder is wasted
  => 1/2 of divisor is wasted
  - Instead of shifting divisor to right, shift remainder to left
- Eliminate Quotient register by combining with Remainder register as shifted left

Improved Divide Hardware

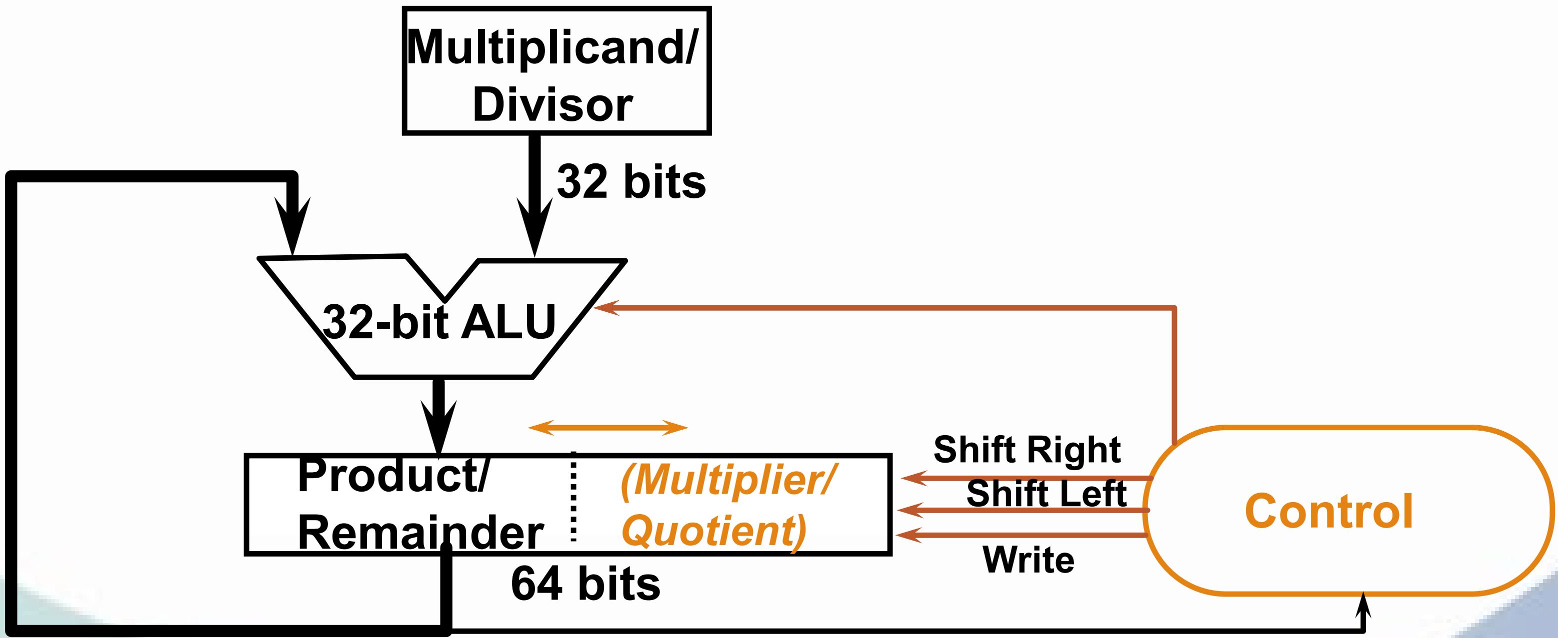
- 32-bit Divisor register, 32-bit ALU, 64-bit Remainder register





Multiply/Divide Hardware

- 32-bit Multiplicand/Divisor register, 32 -bit ALU, 64-bit Product/Remainder register, (0-bit Multiplier/Quotient register



Signed Division

- Dividend (Dd) = Quotient (Q) × Divisor (Dv) + Reminder (R)
- The sign of the dividend (Dd) must be the same as that of the reminder (R)
- If the sign of the dividend is not the same as that of the divisor, the quotient (Q) must be negative
- Assuming all are positive, and change the signs in the end

+7 ÷ 2 = 3, remainder = +1

+7 ÷ -2 = -3, remainder = +1

-7 ÷ +2 = -3, remainder = -1

-7 ÷ -2 = +3, remainder = -1

Dd	Q	Dv	R
+	+	+	+
+	−	−	+
−	+	−	−
−	−	+	−

Arithmetic for Computers: Floating Point

Floating Point

- What can be represented in N bits?

Unsigned	0	to	$2^n - 1$
2's Complement	-2^{n-1}	to	$2^{n-1} - 1$

But, what about ...

- very large numbers? 9,349,398,989,787,762,244,859,087,678
- very small number? 0.0000000000000000000000045691
- rational number $\frac{2}{3}$
- irrationals $\sqrt{2}$

Scientific Notation: Binary

Observation: all the black digits are fixed

Significand *Fraction* ***exponent***

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

"binary point" ***radix (base)***

- Normalized form: no leading 0s
  (exactly one digit to left of the binary point)

- To represent $1/2^9$
  - Normalized: 1.0×2^{-9}
  - Not normalized: 0.1×2^{-8}, 10.0×2^{-10}

Floating Point Representation

- Normal format: $1.xxx_{two} \times 2^{yyy}$
- Two format
  - 32 bits for single-precision
  - 64 bits for double-precision
- A simple single-precision representation:

precision exponent

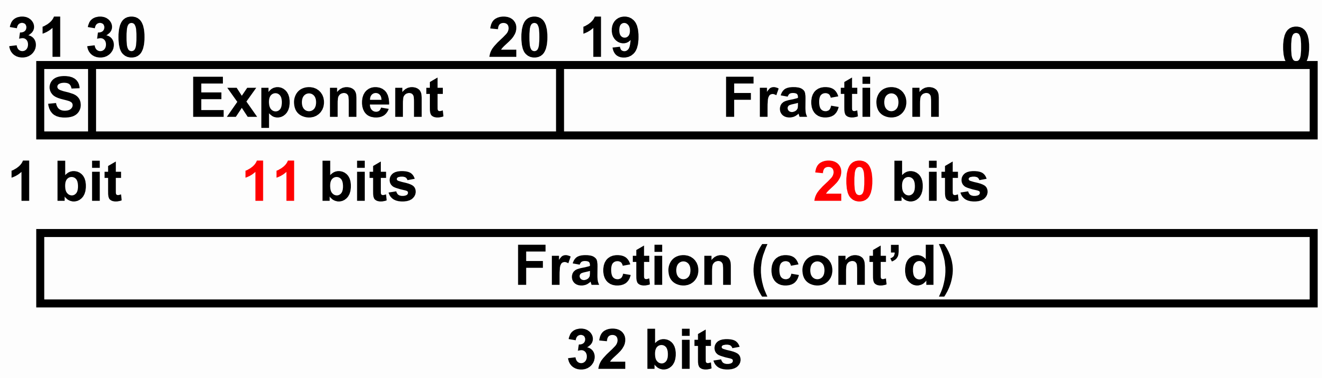
S=1 -
S=0 +

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

31	30 … 23	22 … 0
S	Exponent	Fraction
1 bit	8 bits	23 bits

Floating Point Representation

- Double-precision Representation



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

Floating Point Standard

- Defined by IEEE 754 Standard
- Sign bit:
  - 0:positive;
  - 1:negative.
- Fraction:
  - Leading 1 implicit to save bits for normalized numbers
  - Single precision (1 + 23 bits) vs. Double precision (1 + 52 bits)
  - For normalized numbers: 0 < Fraction < 1
  - For zero, there is no leading 1
- Developed in response to divergence of representations
  - Portability issues for scientific code
- Now almost universally adopted

IEEE 754 Standard

- Exponent:
  - Need to represent positive and negative exponents
  - 2’s complement does not work

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

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	.	.	.

All positive

If we use integer comparison for these two words (directly), we will conclude that 1/2 > 2!!!

Solution - Biased Notation

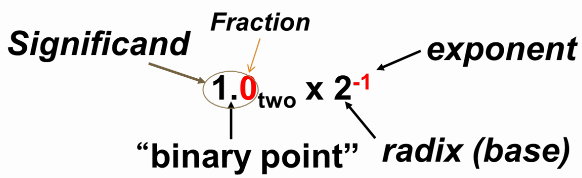
- **Ex: Biased 7**

0000	-7
0001	-6
0010	-5
0011	-4
0100	-3
0101	-2
0110	-1
0111	0
1000	1
1001	2
1010	3
1011	4
1100	5
1101	6
1110	7
1111	8

$$(-1)^S \times (1 + Fraction) \times 2^{Exponent-7}$$

(IEEE 754 uses Biased 127)

IEEE 754 Standard



	single: 8 bits double: 11 bits	single: 23 bits double: 52 bits
S	Exponent	Fraction

- S: sign bit (0 ⇒ non-negative, 1 ⇒ negative)

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

- Normalize significand: 1.0 ≤ |significand| < 2.0
  - Always has a leading pre-binary-point 1 bit, so no need to represent it explicitly (hidden bit)
  - Significand is Fraction with the "1" restored
- Exponent: excess representation: actual exponent + Bias
  - Ensures exponent is unsigned
  - Single: Bias = 127; Double: Bias = 1023

Decode: IEEE754→FP →exp-127
Encode: FP→IEEE754 →exp+127

Example: IEEE754 to FP

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
s	exponent								fraction																						
1 bit	8 bits								23 bits																						

0	**0110 1000**	**101 0101 0100 0011 0100 0010**

- Sign: 0 => positive
- Exponent:
  - $0110\ 1000_{two} = 104_{ten}$
  - Bias adjustment: 104 - 127 = -23
- Fraction:

  $2^{-1}+2^{-3}+2^{-5}+2^{-7}+2^{-9}+2^{-14}+2^{-15}+2^{-17}+2^{-22}$
  $= 0.666115$

Decode: IEEE754→FP →exp-127

Represents: $1.666115_{ten} \times 2^{-23} \approx 1.986 \times 10^{-7}$

Binarize Floating Point

- Example

1. 0.75

	0.75 x 2 = 1.5	0.1xxx
Drop the integer	0.5 x 2 = 1	0.11

2. 0.625

0.625 x 2 = 1.25	0.1xxx
0.25 x 2 = 0.5	0.10xx
0.5 x 2 = 1	0.101

Example: FP to IEEE754

- Number = - 0.75
- Sign: negative => 1
- Exponent:
  - Bias adjustment: -1 +127 = 126
  - $126_{ten} = 0111\ 1110_{two}$

$$0.75 = \frac{3}{4} = 3 \times 2^{-2}{}_{ten} = 11_{two} \times 2^{-2} = 1.1_{two} \times 2^{-1}$$

$$-0.75 = -1.1_{two} \times 2^{-1} = (-1) \times (1 + .1) \times 2^{-1}{}_{two}$$

Encode: FP→IEEE754 →exp+127

S	Exponent	Fraction
1	0111 1110	100 0000 0000 0000 0000 0000

Example: FP to IEEE754

- Number = 1/3
  - $1/3 = 0.3333..._{ten} = 0.0101010101..._{two} \times 2^{0} = 1.0101010101..._{two} \times 2^{-2}$
- Sign: positive => 0
- Exponent:
  - Bias adjustment: -2 +127 = 125
  - $125_{ten} = 0111\ 1101_{two}$

Encode: FP→IEEE754 →exp+127

S	Exponent	Fraction
0	0111 1101	0101 0101 0101 0101 0101 010

Single Precision (SP) vs. Double Precision (DP)

	Single Precision	Double Precision
Reserved	Exp 00000000 and 11111111	Exp 0000…00 and 1111…11
Smallest value	Exp: $00000001 \rightarrow 1 - 127 = -126$ Fraction: 000…00→significand = 1.0 $\pm 1.0 \times 2^{-126} \approx \pm 1.2 \times 10^{-38}$	Exp: $00000000001 \rightarrow 1 - 1023 = -1022$ Fraction: 000…00→significand = 1.0 $\pm 1.0 \times 2^{-1022} \approx \pm 2.2 \times 10^{-308}$
Largest value	Exp: $11111110 \rightarrow 254 - 127 = 127$ Fraction: 111…11→significand ≈ 2.0 $\pm 2.0 \times 2^{127} \approx \pm 3.4 \times 10^{38}$	Exp: $11111111110 \rightarrow 1023$ Fraction: 111…11→significand ≈ 2.0 $\pm 2.0 \times 2^{1023} \approx \pm 1.8 \times 10^{308}$

Special Numbers

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)

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
s	exponent								fraction																						
1 bit	8 bits								23 bits																						

Special Numbers

- Zero
  - Exponent = 00000000 for SP; 00000000000 for DP
  - Fraction bits are all zeros.
  - Sign bit
    - `+0: 0 00000000 00000000000000000000000`
    - `-0: 1 00000000 00000000000000000000000`
- +/- Denormalized numbers (denorms)
  - Exponent bits are all zeros (but representing **-126** fixed)
  - Fraction is not zero
  - Number degradation in significance (gradual underflow)
  - The smallest SP normalized number: 1.0×2^{-126}
  - The smallest SP denormalized number:

    0.0000 0000 0000 0000 0000 001two $\times \mathbf{2^{-126}} = 1.0_{\text{two}} \times 2^{-149}$

Normal vs Denormalized Numbers

- Denormalized numbers (denorms)
  - Exponent of all 0 bits
  - it represents an exponent of −126 in single precision (not −127), −1022 in double precision (not −1023)
- Normal numbers
  - The smallest exponent is 1

Positive and Negative Infinity

- For floating points, a number dividing by zero would be represented as +/- infinity, not overflow
- +/- infinity are still meaningful for comparisons
  - e.g., $\frac{X}{0} > Y$
- +/- infinity in IEEE 754
  - **Largest exponent value is used for infinity**
  - Fraction bits are all zeroes
  - Sign bit for +/-

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)

NaN (Not a Number)

- Undefined or unrepresentable numbers, such as, $\sqrt{-1}$ or $\frac{0}{0}$.
  - Exponent bits are all 1's
  - Fraction is non-zero
- Purposes
  - Debugging?
  - Compute any numbers with NaN is NaN

Floating-Point (FP) Addition

Basic addition algorithm:

(1) Align binary point (X+Y)

- Align the smaller number to make its exponent the same as the larger one.

$$1.000_2 \times 2^{-1} + -1.110_2 \times \boxed{2^{-2}} \xrightarrow{align} 1.000_2 \times 2^{-1} + -0.111_2 \times \boxed{2^{-1}}$$

(2) Add significands

$$1.000 - 0.111 = 0.001$$

(3) Normalization & check for over/underflow

$$0.001 \xrightarrow{normalization} 1.000 \times 2^{-4}$$

(4) Round the significands and renormalize if necessary

= 1.000×2^{-4} (no change)

Ex: $1.111\boxed{11} \rightarrow \overbrace{10.0000}$

If you only have 4-bit fraction

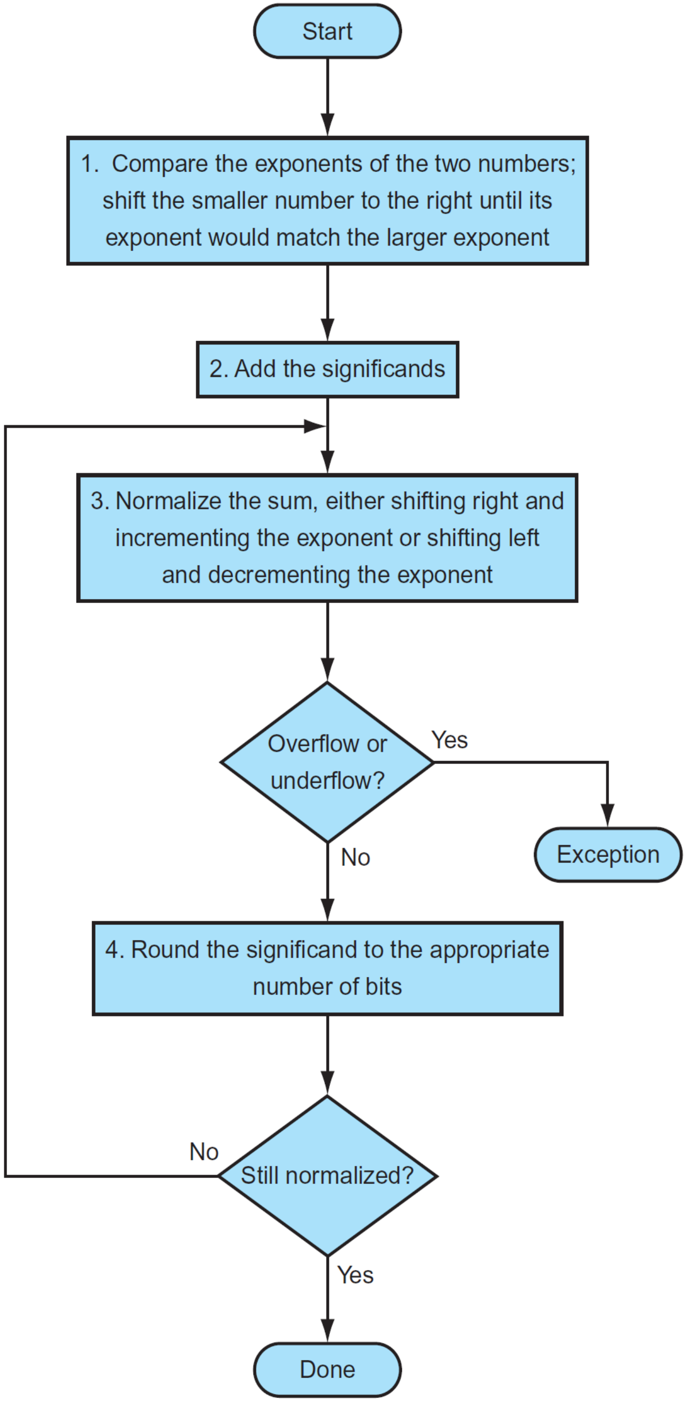
FP Adder Hardware

- Much more complex than an integer adder
- Doing it in one clock cycle would take too long
  - Much longer than integer operations
  - Slower clock would penalize all instructions
- FP adder usually takes several cycles
  - Can be pipelined

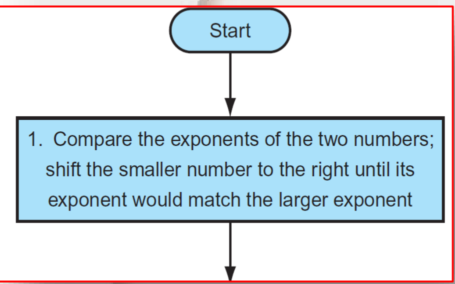
Steps for FP Add

1. Compare the exponents of the two numbers; shift the small number to the right until its exponent would match **the larger exponent**
2. Add the significands
3. Normalized the sum, either shifting right and incrementing the exponent or shifting left and decrementing the exponent
4. Check if overflow or underflow? (If yes, raise exception)
5. Round the significand to the appropriate number of bits
6. Check if still normalized (If no, go to step 3)

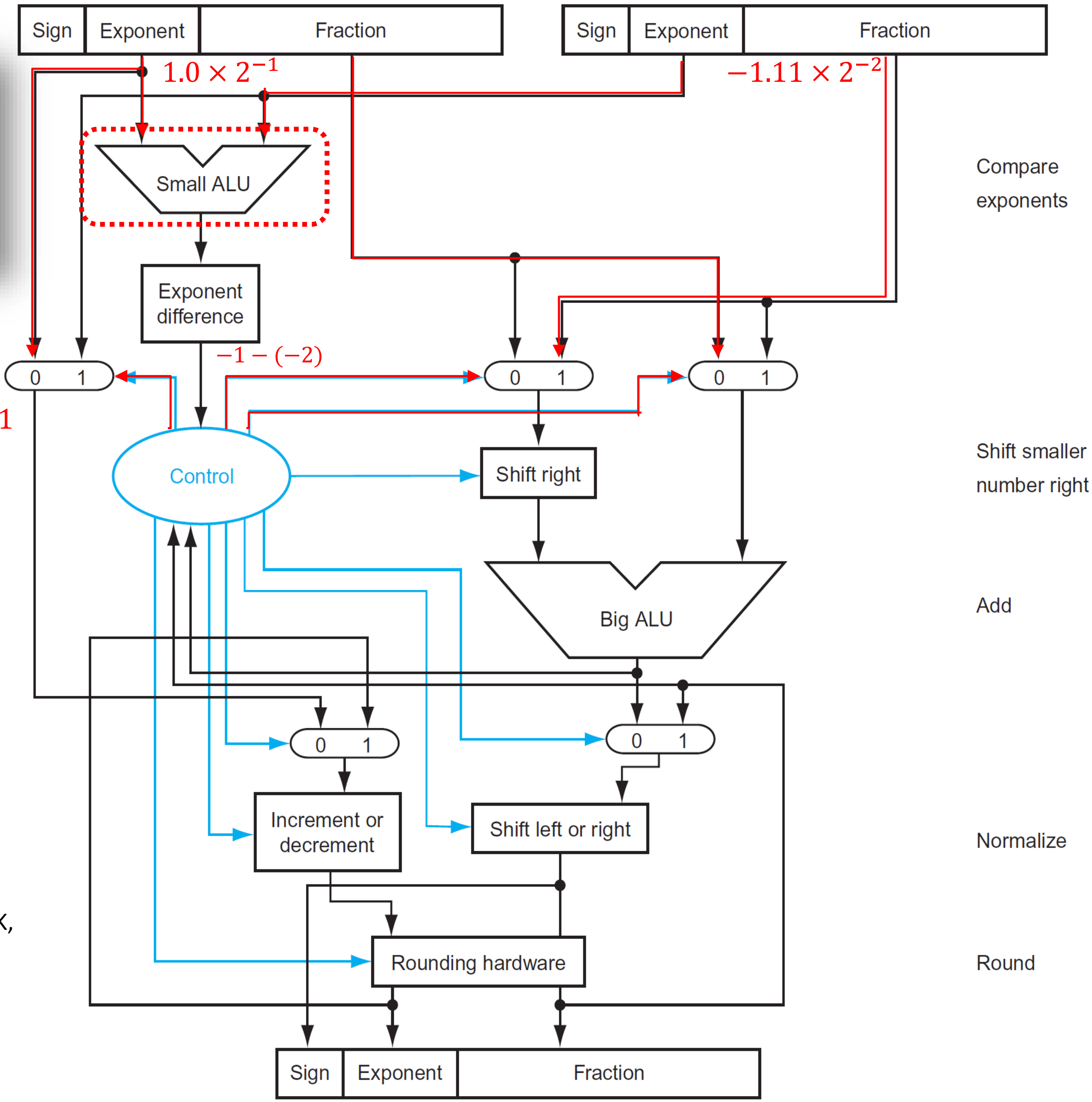
Credit: Fig. 3.14 in Computer Organization and Design 5^{th}, MK, Patterson and Hennessy.



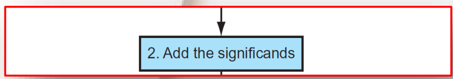
Credit: Fig. 3.14 in Computer Organization and Design 5th , MK, Patterson and Hennessy.



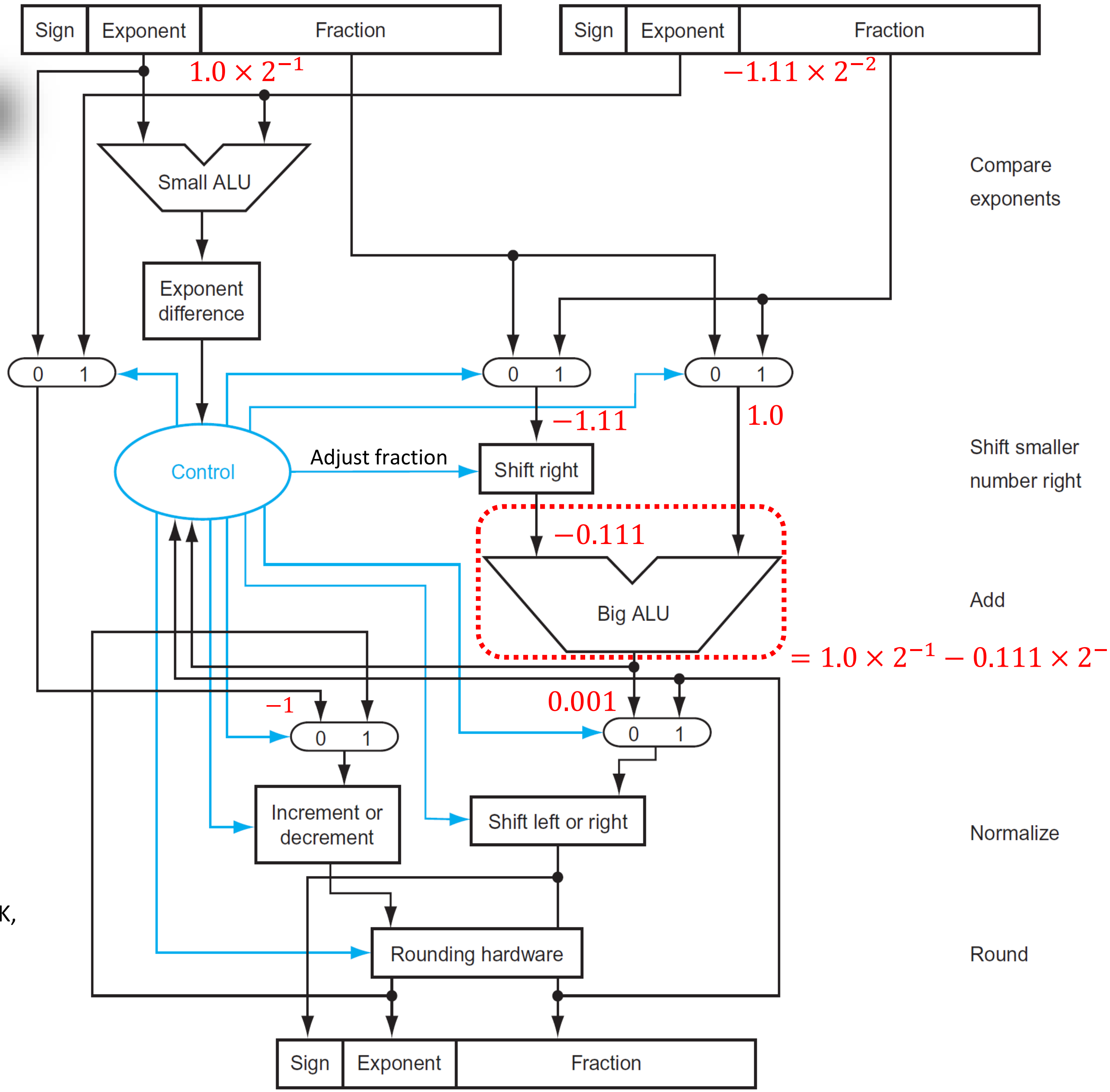
Larger exp = −1



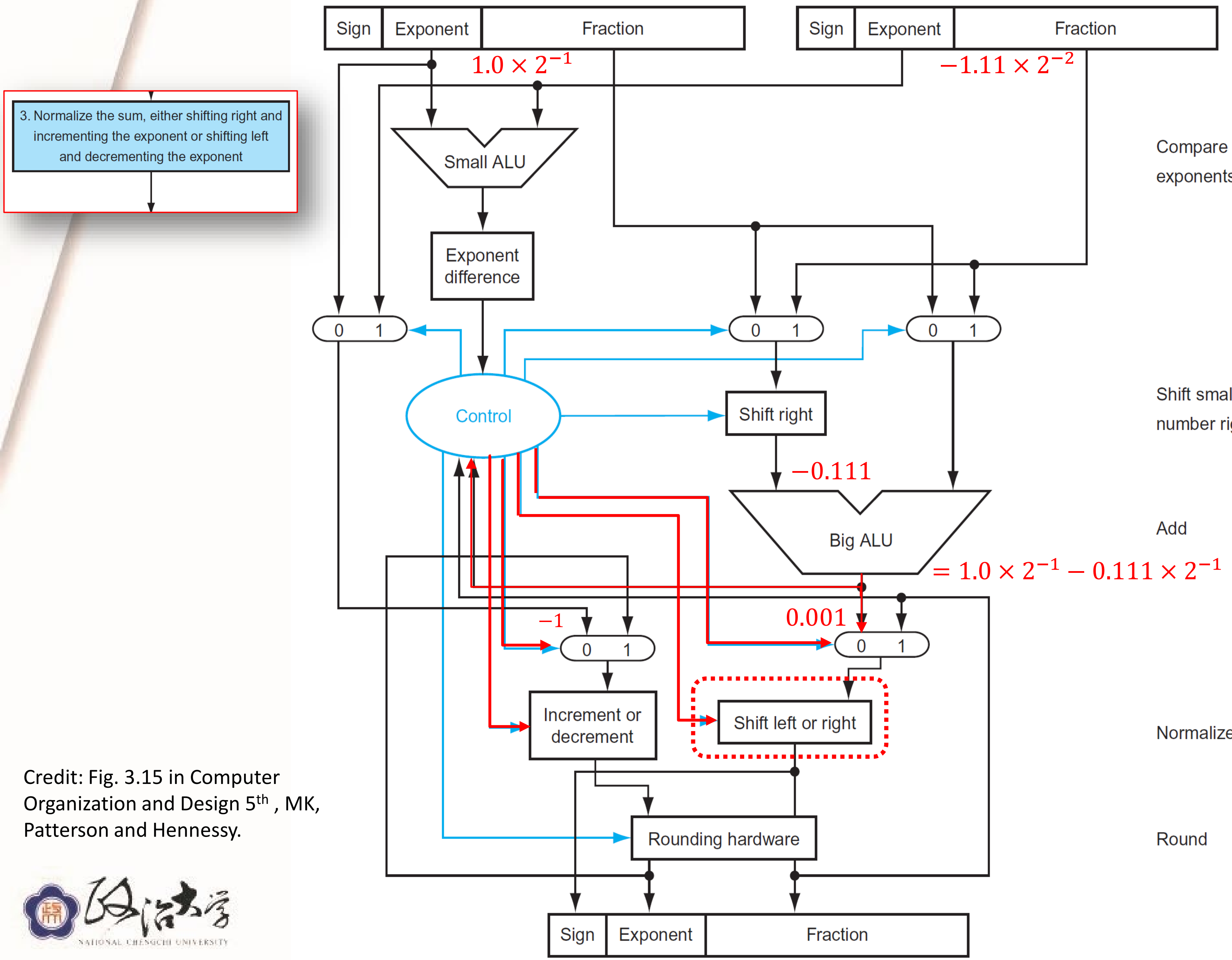
Credit: Fig. 3.15 in Computer Organization and Design 5th , MK, Patterson and Hennessy.



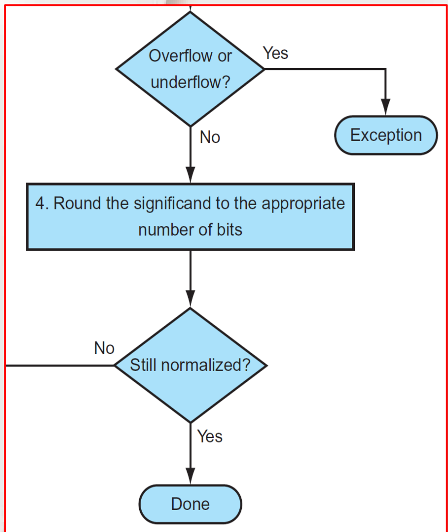
Add Significands
(not just Fractions)

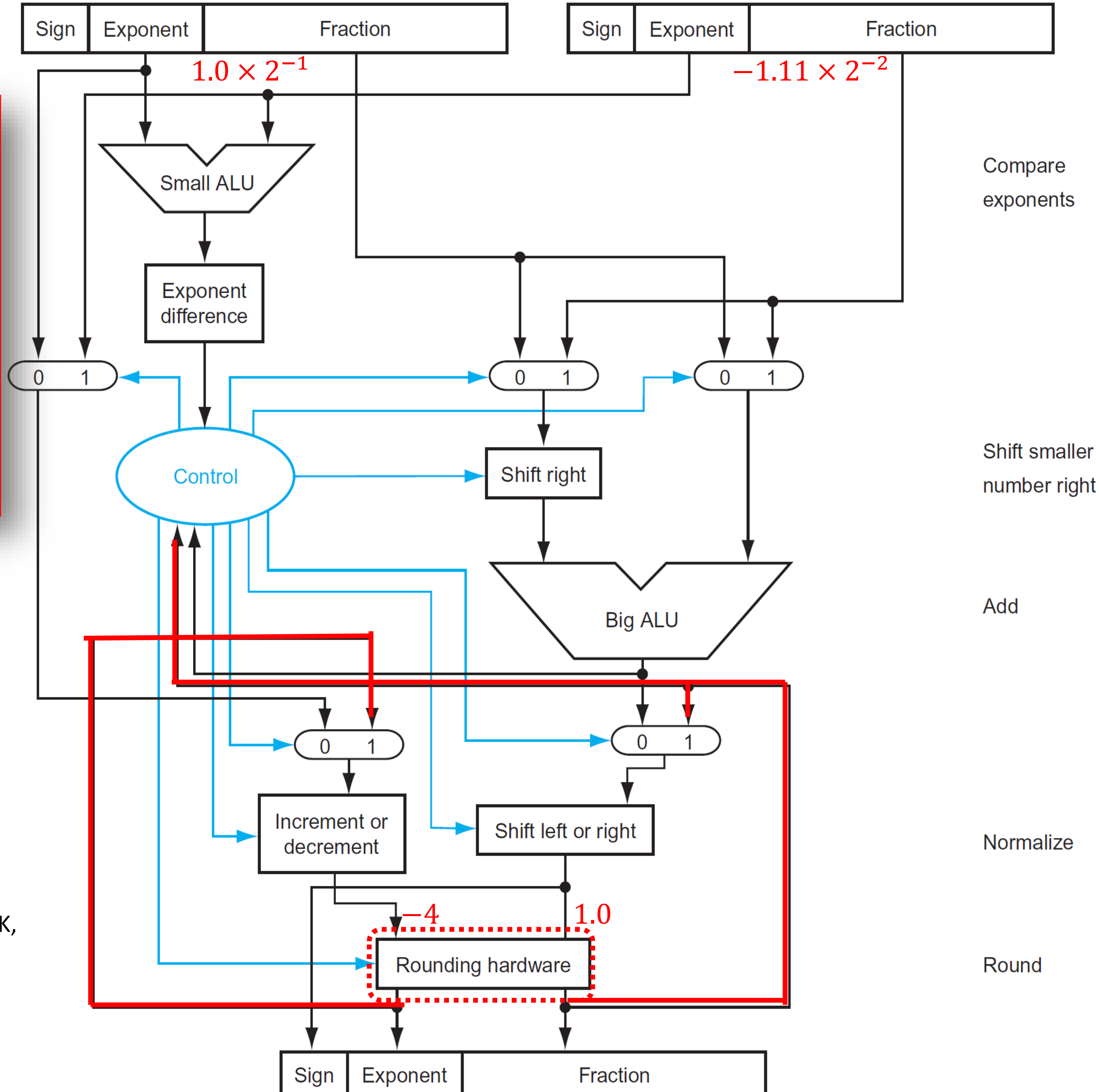


Credit: Fig. 3.15 in Computer Organization and Design 5th , MK, Patterson and Hennessy.



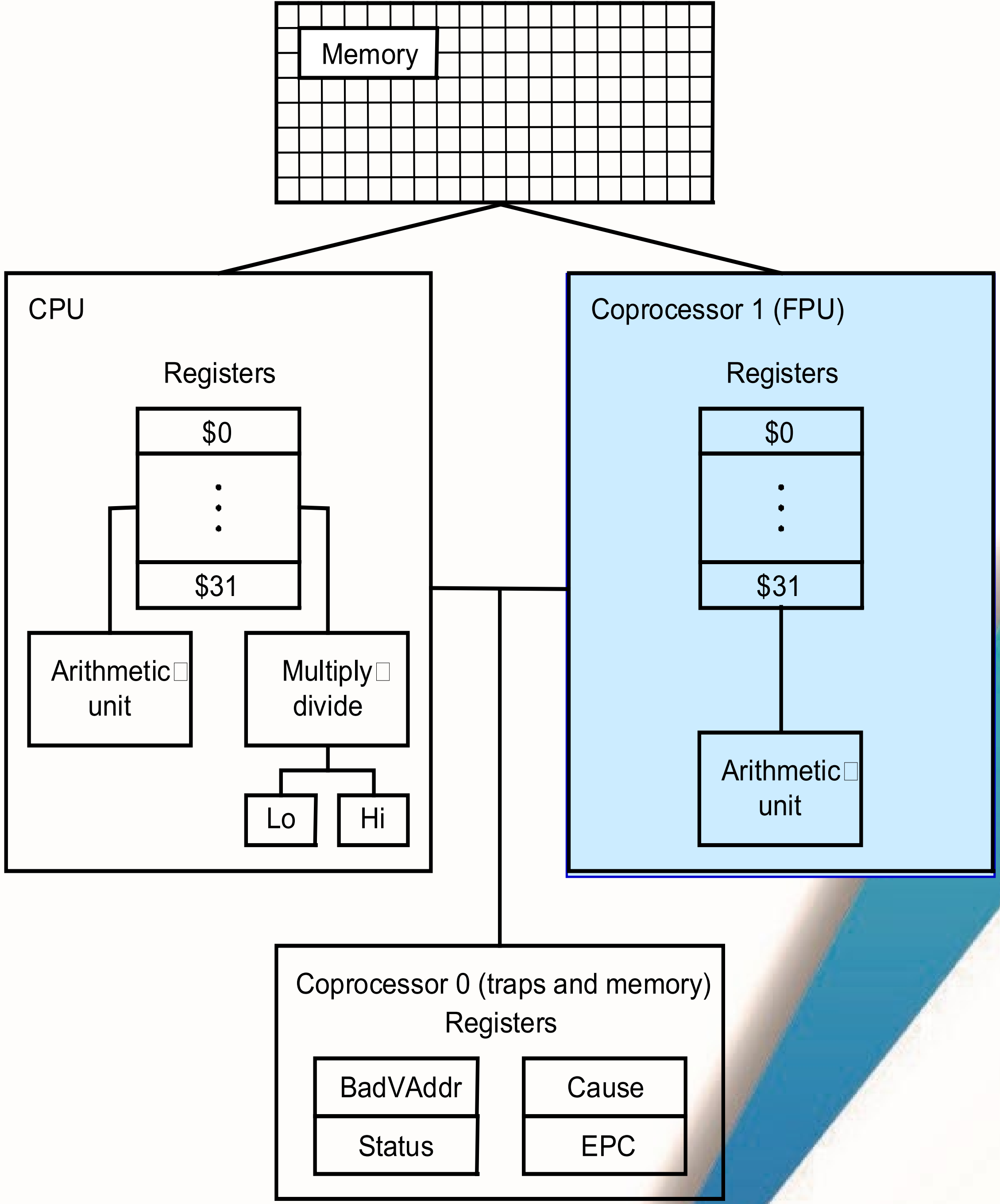
Credit: Fig. 3.15 in Computer Organization and Design 5th , MK, Patterson and Hennessy.





Credit: Fig. 3.15 in Computer Organization and Design 5^{th} , MK, Patterson and Hennessy.

MIPS R2000 Organization



MIPS Floating Point

- Separate floating point instructions:
  - Single precision: `add.s`,`sub.s`,`mul.s`,`div.s`
  - Double precision: `add.d`,`sub.d`,`mul.d`,`div.d`
- FP part of the processor:
  - contains 32 32-bit registers: `$f0`, `$f1`, …
  - most registers specified in .s and .d instruction refer to this set
  - Double precision: by convention, even/odd pair contain one DP FP number: `$f0/$f1`,`$f2/$f3`
  - separate load and store: `lwc1` and `swc1`
    - `lwc1 $f0, 0($t0)`
    - `swc1 $f0, 0($t0)`
  - Instructions to move data between main processor and coprocessors:
    - `mfc0`,`mtc0`,`mfc1`,`mtc1`, etc.

lwcz: lw means load word, c co-processor (z= 0 traps processor, z=1 floating point unit)
lwc1 : lw to floating point co-processor

Pitfall: Associativity

- Floating Point add, subtract are not associative
- Ex:
  - $A = -1.5 \times 10^{38}, B = 1.5 \times 10^{38}, C = 1.0$
  - In floating point representation, $(A + B) + C \neq A + (B + C)$
    - $(A + B) + C = 1.0$
    - $A + (B + C) = 0.0$

Consider $B + C$
(1) Align binary point
(2) Add significands
(3) …

What happened ?

Conclusions

- Bit Interpretation is contingent on how an instruction works with bits.
- Computer represents numbers with finite range and precision
- Number range and precision is limited
  - Need overflow and underflow
- FP Accuracy matters for scientific code