RISC V Architecture

Prof. H R Vanamala
Department of Electronics and Communication Engg.

RISC V ARCHITECTURE

UNIT 4: Arithmetic for Computers

Prof. H R Vanamala
Department of Electronics and Communication Engineering

Representation for non-integer numbers : Including very small and very large numbers

Scientific notation:

Examples:

Normalized scientific notation:

$1.0_{ten} \times 10^{-9}$

-2.34×10^{56}

Not normalized:

$0.1_{ten} \times 10^{-8}$

$10.0_{ten} \times 10^{-10}$

$+0.002 \times 10^{-4}$

$+987.02 \times 10^{9}$

Floating Point

In binary:

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

$\pm 1.xxxxxxx_2 \times 2^{yyyy}$

Types: float and double in C

Three advantages:

- It simplifies exchange of data that includes floating-point numbers
- It simplifies the floating-point arithmetic algorithms to know that numbers will always be in this form
- It increases the accuracy of the numbers that can be stored in a word, since real digits to the right of the binary point replace the unnecessary leading 0s.

RISC-V floating-point number :

s is the sign of the floating-point number (1 meaning negative),
exponent is the value of the 8-bit exponent field (including the sign of the exponent), and fraction is the 23-bit number.
This representation is sign and magnitude, since the sign is a separate bit from the rest of the number.
F involves the value in the fraction field and E involves the value in the exponent Field.

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																						

In general, floating-point numbers are of the form

$$(-1)^S \times F \times 2^E$$

Floating Point

Fraction : The value generally between 0 and 1, placed in the fraction field. The fraction is also called the mantissa.

Exponent In the numerical representation system of floating-point arithmetic, the value that is placed in the exponent field.

Fixed word size: This tradeoff is between precision and range: increasing the size of the fraction enhances the precision of the fraction, while increasing the size of the exponent increases the range of numbers that can be represented.

Fractions almost as small as $2.0_{ten} \times 10^{-38}$ and numbers almost as large as $2.0_{ten} \times 10^{38}$ can be represented in a computer

overflow (floating point) A situation in which a positive exponent becomes too large to fit in the exponent field.

underflow (floating point) A situation in which a negative exponent becomes too large to fit in the exponent field.

Floating Point

Double precision: A floating-point value represented in a 64-bit double word.
Single precision: A floating-point value represented in a 32-bit word.

63	62	61	60	59	58	57	56	55	54	53	52	51	50	49	48	47	46	45	44	43	42	41	40	39	38	37	36	35	34	33	32
s	exponent											fraction																			
1 bit	11 bits											20 bits																			

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
fraction																															
32 bits																															

RISC-V double precision: as small as $2.0_{ten} \times 10^{-308}$ and almost as large as $2.0_{ten} \times 10^{308}$.
Although double precision does increase the exponent range, its primary advantage is its greater precision because of the much larger fraction.

Floating Point: Exceptions and Interrupts

Exception/Interrupt - unscheduled event that disrupts program execution; used to detect overflow, for example.

An exception that comes from outside of the processor. (Some architectures use the term interrupt for all exceptions.)

RISC-V computers do not raise an exception on overflow or underflow; instead, software can read the floating-point control and status register (fcsr) to check whether overflow or underflow has occurred.

Floating Point: Floating-Point Representation

Defined by IEEE Std 754-1985

Developed in response to divergence of representations

- Portability issues for scientific code

Now almost universally adopted

Two representations

- Single precision (32-bit)
- Double precision (64-bit)

Floating Point: Floating-Point Representation

To pack even more bits into the number, IEEE 754 makes the leading 1 bit of normalized binary numbers implicit.(Not RISC V)
The number is actually 24 bits long in single precision (implied 1 and a 23-bit fraction), and 53 bits long in double precision (1 + 52).
To be precise, we use the term significand to represent the 24- or 53-bit number that is 1 plus the fraction, and fraction when we mean the 23- or 52-bit number.
0 has no leading 1, it is given the reserved exponent value 0 so that the hardware won't attach a leading 1 to it.

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

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

Floating Point: IEEE 754 Floating-Point Standard

IEEE 754 encoding of floating-point numbers: A separate sign bit determines the Sign.

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)

Note: Check slide 17-18

Floating Point -Example

Example: $1.0_{two} \times 2^{-1}$ would be represented in a single precision as (Remember that the leading 1 is implicit in the significand)

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	0	0	0

The value $1.0_{two} \times 2^{+1}$
would look like the smaller binary 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
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	0	0	0

IEEE 754 uses a bias of 127 for single precision, so an exponent of −1 is represented by the bit pattern of the value

$-1 + 127_{ten}$, or $126_{ten} = 0111\ 1110_{two}$,

and +1 is represented by 1 + 127, or $128_{ten} = 1000\ 0000_{two}$.

The exponent bias for double precision is 1023.

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

The range of single precision numbers is then from as small as

$$\pm 1.00000000000000000000000_{two} \times 2^{-126}$$

to as large as

$$\pm 1.11111111111111111111111_{two} \times 2^{+127}.$$

Unit 4: Arithmetic for Computers

Floating Point

Represent –0.75 in single and double precision.

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

S = 1

Fraction = $1000\ldots00_2$

Exponent = –1 + Bias

Single: –1 + 127 = 126 = 01111110_2

Double: –1 + 1023 = 1022 = 01111111110_2

Single: 1011111101000…00

Double: 1011111111101000…00

The number -0.75_{ten} is also

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

It is also represented by the binary fraction

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

In scientific notation, the value is

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

and in normalized scientific notation, it is

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

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

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

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

The double precision representation is

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

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

What number is represented by the 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	0	0	0

What number is represented by the 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	0	0	0

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,

S = 1
Fraction = $01000\ldots00_2$
Fxponent = $10000001_2 = 129$

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

Floating Point- Floating-Point Addition – Denormal Numbers

Exponent = 000...0 $\Rightarrow$ hidden bit is 0

$$x = (-1)^S \times (0 + \text{Fraction}) \times 2^{-\text{Bias}}$$

- Smaller than normal numbers
  - allow for gradual underflow, with diminishing precision
- Denormal with fraction = 000...0

$$x = (-1)^S \times (0 + 0) \times 2^{-\text{Bias}} = \pm 0.0$$

Two representations of 0.0!

Floating Point- Floating-Point Addition - Infinities and NaNs

Exponent = 111...1, Fraction = 000...0

- ±Infinity
- Can be used in subsequent calculations, avoiding need for overflow check

Exponent = 111...1, Fraction ≠ 000...0

- Not-a-Number (NaN)
- Indicates illegal or undefined result
  - e.g., 0.0 / 0.0
- Can be used in subsequent calculations

Floating Point - Addition – Binary Numbers

Now consider a 4-digit binary example

$1.000_2 \times 2^{-1} + -1.110_2 \times 2^{-2}$ (0.5 + –0.4375)

Steps:

1. Align binary points

   Shift number with smaller exponent

   $1.000_2 \times 2^{-1} + -0.111_2 \times 2^{-1}$

2. Add significands

   $1.000_2 \times 2^{-1} + -0.111_2 \times 2^{-1} = 0.001_2 \times 2^{-1}$

3. Normalize result & check for over/underflow

   $1.000_2 \times 2^{-4}$, with no over/underflow

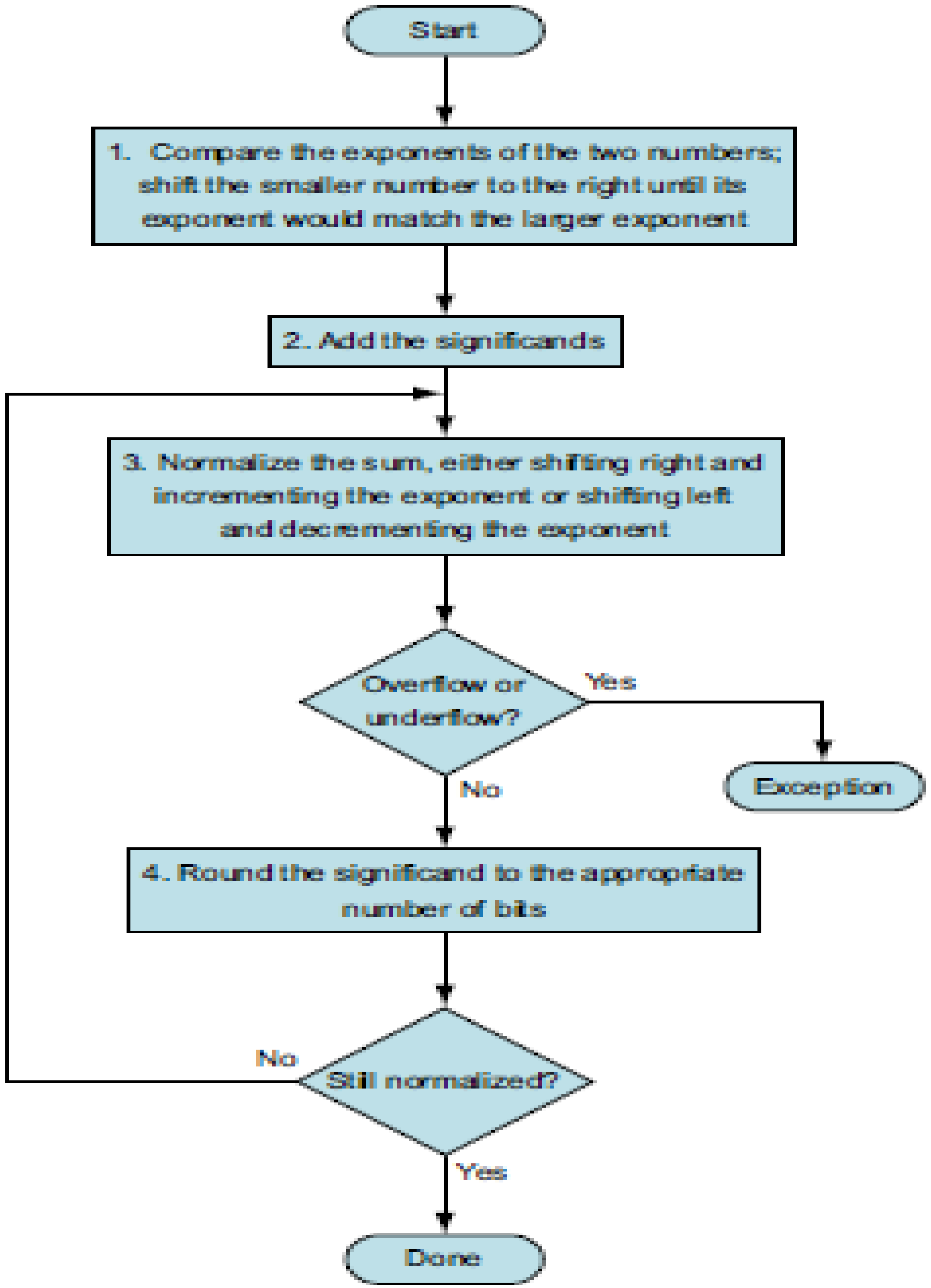
4. Round and renormalize if necessary

   $1.000_2 \times 2^{-4}$ (no change) = 0.0625

Unit 4: Arithmetic for Computers

Floating Point – Addition-Hardware

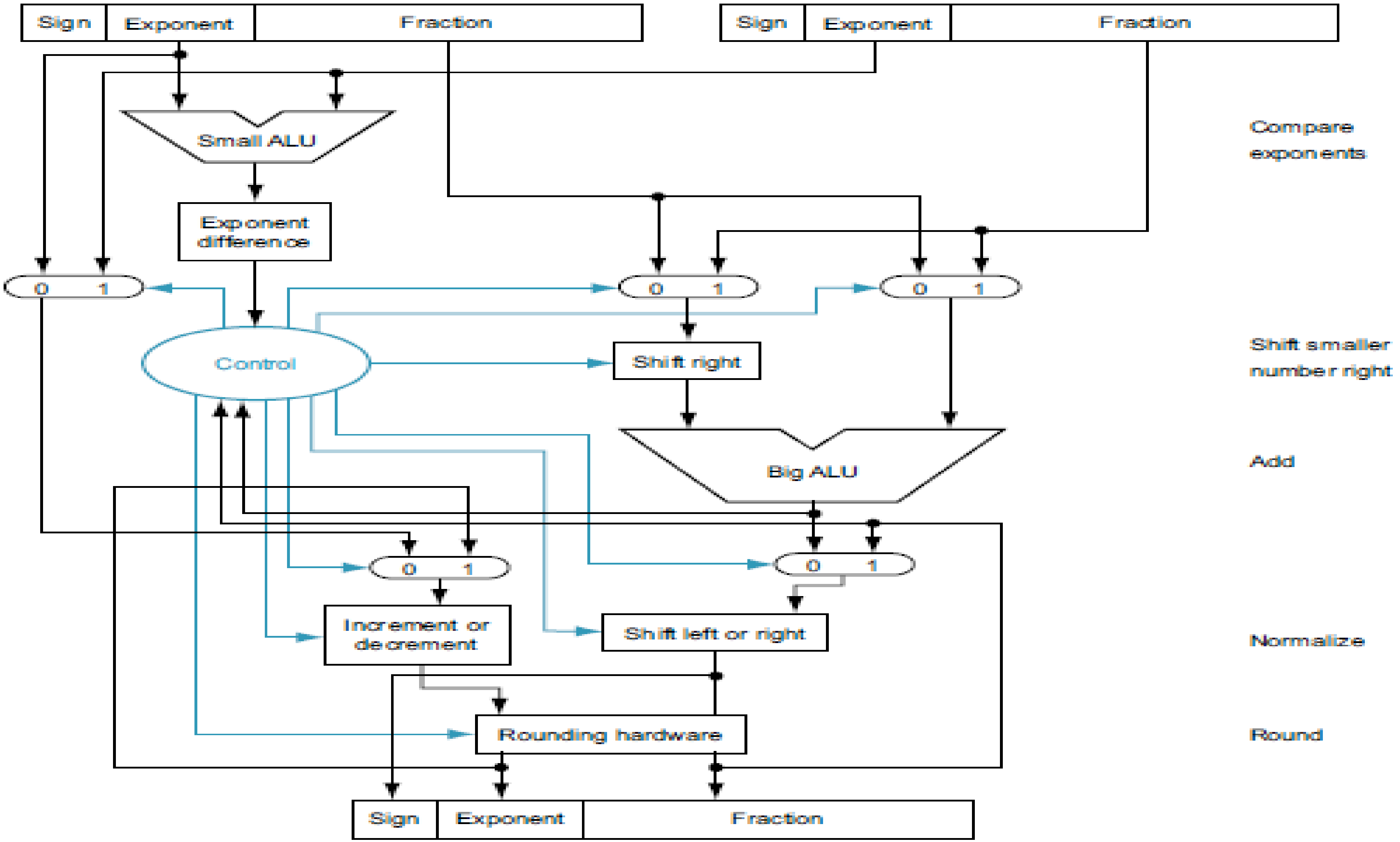
Floating-point addition: The normal path is to execute steps 3 and 4 once, but if rounding causes the sum to be unnormalized, we must repeat step 3.



Floating Point – Addition - Hardware

- Much more complex than 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

Floating Point – Addition –Hardware – Block diagram



Hennessy

Floating Point - Addition –Hardware – Block Diagram

The steps of the flow chart above correspond to each block, from top to bottom.

- First, the exponent of one operand is subtracted from the other using the small ALU to determine which is larger and by how much. This difference controls the three multiplexers; from left to right, they select the larger exponent, the significand of the smaller number, and the significand of the larger number.
- The smaller significand is shifted right, and then the significands are added together using the big ALU.
- The normalization step then shifts the sum left or right and increments or decrements the exponent.
- Rounding then creates the final result, which may require normalizing again to produce the actual final result.

THANK YOU

Vanamala H R
Department of Electronics and Communication