Floating Point

0.4×10^{2}

1.5×10^{-22}

- Representation for non-integral numbers
- Including very small and very large numbers
- **Scientific notation:** A single digit to the left of the decimal point. A number in scientific notation that has no leading 0s is called a normalized number.
  - Normalized: -3.81×10^{22}
  - Not normalized: 0.006×10^{-5}, 105.75×10^{4}

0.4×10^{2} (No)

1.5×10^{-22} (Yes)

- Just as in scientific notation, numbers are represented as a single nonzero digit to the left of the binary point (**floating point**). In binary, the form is:

$$\pm 1.\underbrace{xxxx}_{2} \times 2^{\underbrace{yyyy}}$$

Floating Point Representation

- IEEE 754 Floating Point Standard
  - Single precision floating point (32-bit)
  - Double precision floating point (64-bit)

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																						

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	11 bits											20 bits																			
fraction (continued)																															
32 bits																															

Floating Point Representation

$B = 2^{n-1} - 1$

- In general, floating-point numbers are of the form
$$(-1)^S \times (1 + Fraction) \times 2^{(Exponent-Bias)}$$
- S: sign bit (0: non-negative, 1: negative)
- Normalized significand: $1.0 \leq |significand| < 2.0$
  - 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
  - Exponent is unsigned
  - Single: Bias = **127**; Double: Bias = **1023**

Floating Point Representation

- In general, floating-point numbers are of the form

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

- Example: -0.75_{10}

$-0.75_{10} = -(0.11)_2 = -(1.1)_2 \times 2^{-1}$ ← decimal representation

binary

S= 1

Fraction = $\boxed{10000\ldots000}_2$ (23 bit in single precision and 52 bit in double precision)

Exponent = -1 + Bias

Single: –1 + 127 = 126 = $0111\ 1110_2$ (8 bit)

Double: –1 + 1023 = 1022 = $011\ 1111\ 1110_2$ (11 bit)

Actual exponent

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

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

exponent

Floating Point Representation

- In general, floating-point numbers are of the form

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

- Example: -0.75_{10}

$-0.75_{10} = -\,0.11 = -1.1_2 \times 2^{-1}$

S= 1

Fraction = $10000\ldots000_2$ (23 bit in single precision and 52 bit in double precision)

Exponent = –1 + Bias

Single: –1 + 127 = 126 = $0111\ 1110_2$ (8 bit)

Double: –1 + 1023 = 1022 = $011\ 1111\ 1110_2$ (11 bit)

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

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

Floating Point Representation

bias = $2^{8-1} = 128 - 1$

$= \boxed{127}$

8 bit ALU can add this highest!!

- In general, floating-point numbers are of the form
$$(-1)^S \times (1 + Fraction) \times 2^{(Exponent-Bias)}$$
- $(1 + Fraction)$ is known as significand
- Why is Bias here?
  - For the benefit of comparison

0 to 255 (Biased)

8bit দিয়ে → -128 to +127

- Bias 127 (for single precision)
  - Lowest (0) and Highest (255) values of the exponent are reserved
  - The range of usable exponent values is [1,254]. So, the range of (exponent-127) is [-126 .. 127].

Bias exponent = Real exp + 127

Single Precision Range

- Exponents 0000 0000 and 1111 1111 reserved
- Smallest value
  - Exponent: 0000 0001, actual exponent = 1 – 127 = –126
  - Fraction: 000…00 (23bits), significand = 1.0 (i.e., 1 +Fraction)
  - $\pm 1.0 \times 2^{-126} \approx \pm 1.2 \times 10^{-38}$

simply by calculator

- Largest value
  - exponent: 1111 1110, actual exponent = 254 – 127 = +127
  - Fraction: 111…11 (23bits), significand ≈ 2.0 (i.e., 1+Fraction)
  - $\pm 2.0 \times 2^{+127} \approx \pm 3.4 \times 10^{+38}$

by calculator

Double Precision Range

- Exponents 000 0000 0000 and 111 1111 1111 reserved
- Smallest value
  - Exponent: 000 0000 0001, actual exponent =1 – 1023 = –1022
  - Fraction: 000…00 (52bits), significand = 1.0   (i.e., 1 +Fraction)
  - $\pm 1.0 \times 2^{-1022} \approx \pm\ 2.2 \times 10^{-308}$
- Largest value
  - exponent: 111 1111 1110, actual exponent = 2046 – 1023 = +1023
  - Fraction: 111…11 (52bits), significand ≈ 2.0   (i.e., 1+Fraction)
  - $\pm 2.0 \times 2^{+1023} \approx \pm\ 1.8 \times 10^{+308}$

EEE 754 encoding of floating-point numbers

$\boxed{0 \text{ to } 255}$

Single precision		Double precision		Object represented	
Exponent	Fraction	Exponent	Fraction		
0	0	0	0	0	0.0×2^0
0	Nonzero	0	Nonzero	± denormalized number	0.0×2^a
1–254	Anything	1–2046	Anything	± floating-point number	
255	0	2047	0	± infinity	255×2^0
255	Nonzero	2047	Nonzero	NaN (Not a Number)	255×2^a

Denormal Number

- Exponent = 000...0, hidden bit is 0

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

- Smaller than normal numbers
  - allow for gradual underflow, with diminishing precision
  - For example, the smallest single precision denormalized number is

$$0.0000\ 0000\ 0000\ 0000\ 0000\ 001_2 \times 2^{-126} \text{ or } 1.0_2 \times 2^{-149}$$

whereas the smallest positive single precision normalized number was

$$1.0000\ 0000\ 0000\ 0000\ 0000\ 000_2 \times 2^{-126}$$

Floating Point Addition

- Example: (Binary Floating-Point Addition) Add the number 0.5 and -04375 in binary.
- Consider a 4-digit binary example
  - $1.000_2 \times 2^{-1} + (-1.110_2 \times 2^{-2})$

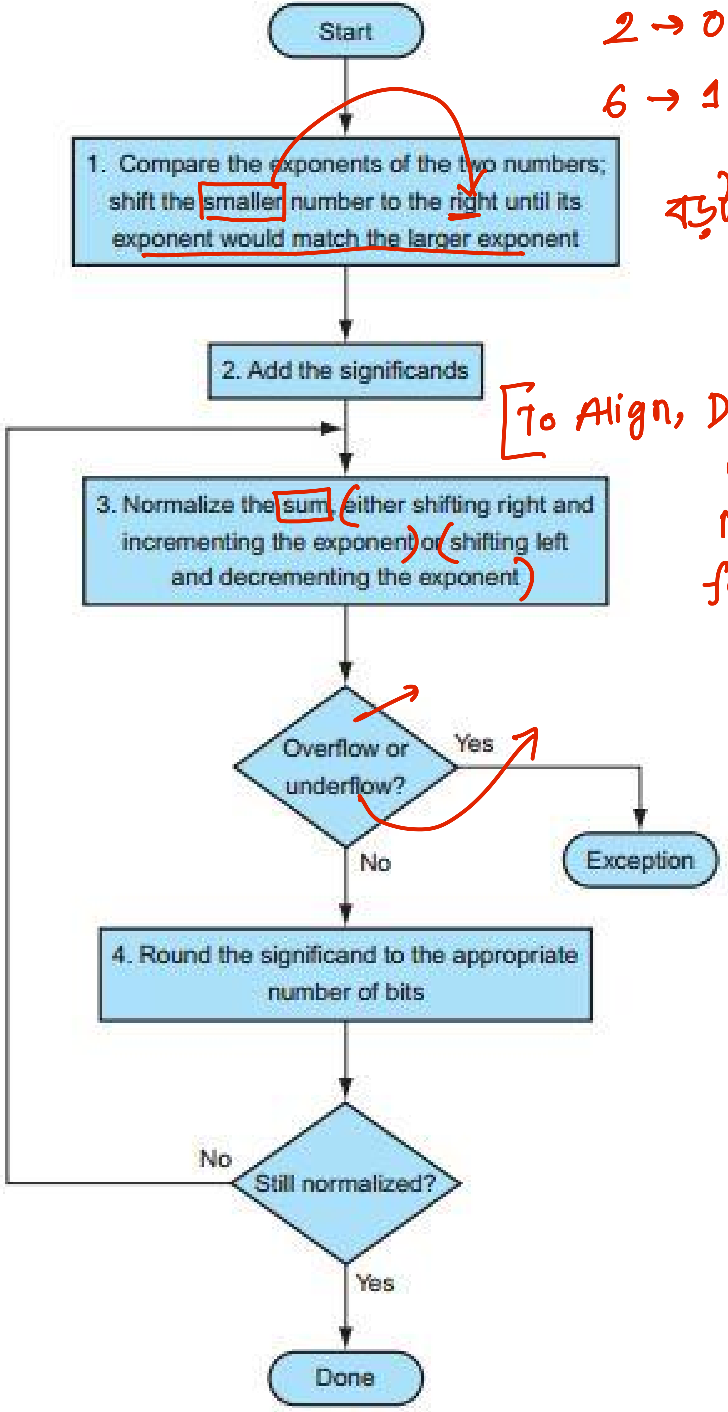
Normalized

Normalized

1. Align binary points
   - Shift the smaller number to right until its exponent would match the larger exponent
   - $1.000_2 \times 2^{-1} + (-0.111_2 \times 2^{-1})$

Not normalized; No problem!

2. Add significands
   - $\boxed{+}1.\boxed{000}_2 \times 2^{-1} + (\boxed{-}0.\boxed{111}_2 \times 2^{-1}) = 0.001_2 \times 2^{-1}$
3. Normalize result & check for over/underflow
   - $1.000_2 \times 2^{-4}$, with no overflow / underflow
4. Round and renormalize if necessary
   - $\mathbf{1.000_2 \times 2^{-4}}$(no change) = **0.0625**

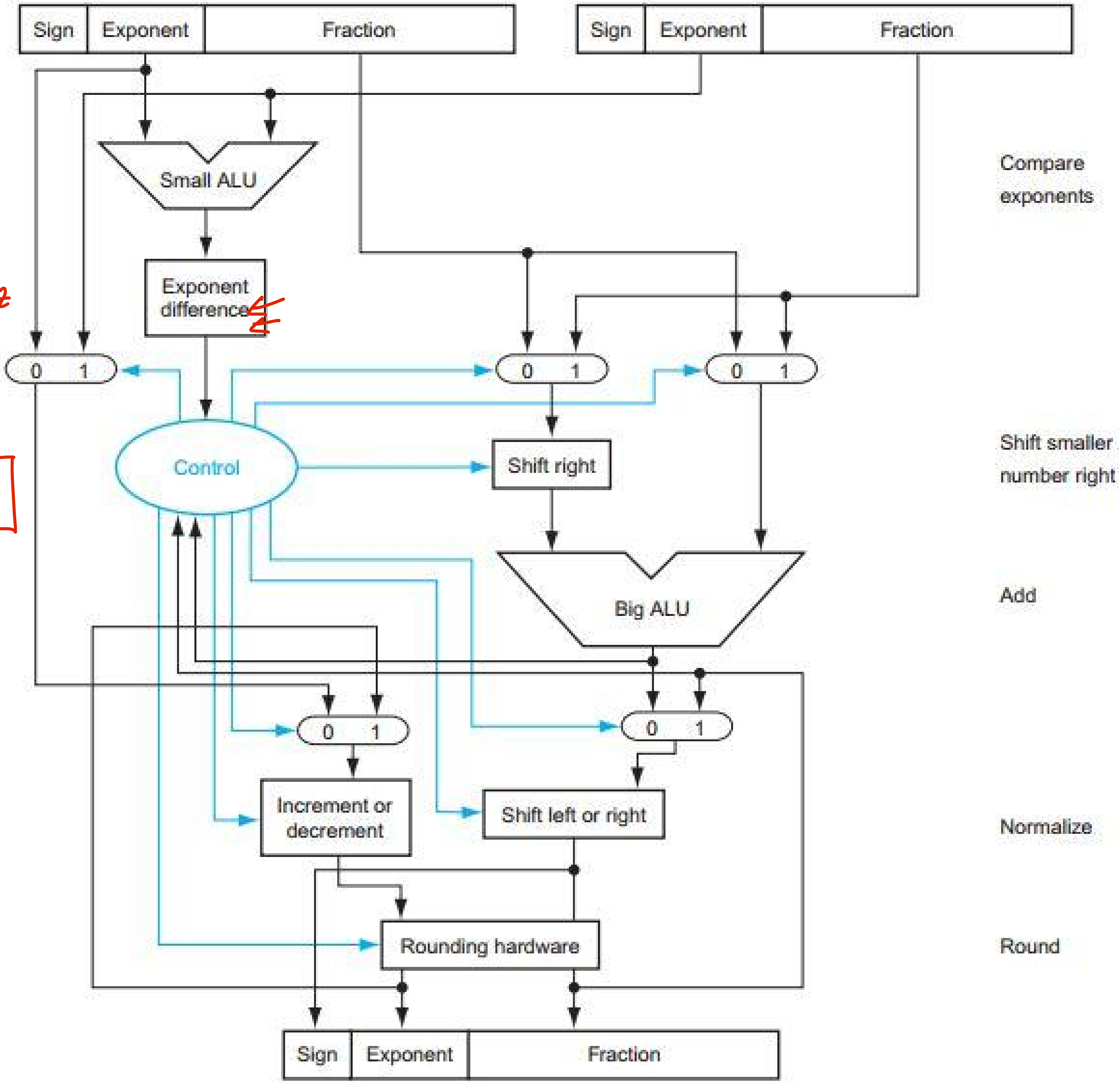


2 → 010 →

6 → 110 →

বড়টাকে Right shift then prob where??

To Align, Don't left bcz তাহলে তো Normalized format এ থেকে যাবে!



Rounding Bits

- Rounding using Guard and round digits, and sticky bit

G	R	S	Action
0	0	0	Truncate
0	0	1	Truncate
0	1	0	Truncate
0	1	1	Truncate
1	0	0	Round to Even
1	0	1	Round Up
1	1	0	Round Up
1	1	1	Round Up

round to closest even number

$$1.100GRS$$

$$1.10001 = 1.53125$$
$$1.10010 = 1.56250$$
$$1.10011 = 1.59375$$