CS / EE 320
Computer Organization and Assembly Language
Spring 2025
Lecture 11

Shahid Masud

Topics: Introduction to Floating Point Numbers, Basic Operations

Topics

- Introduction to Floating Point Numbers
- Conversion from Decimal to Floating Point
- IEEE 754 Floating Point Standard Representation
- Concept of Normalization
- Concept of Biased Exponent
- Different Precision Specification in IEEE 754
- Range and Accuracy Calculations in Floating Point Standard
- Special Representations NaN, Inf, etc.

Quiz Next Week

Common Physical Constants

Mostly Floating-Point Numbers

PHYSICAL CONSTANT

Speed of light	c	3×10^{8} m/s
Planck constant	h	6.63×10^{-34} J s
	hc	1242 eV-nm
Gravitation constant	G	6.67×10^{-11} m^3 kg^{-1} s^{-2}
Boltzmann constant	k	1.38×10^{-23} J/K
Molar gas constant	R	8.314 J/(mol K)
Avogadro's number	N_A	6.023×10^{23} mol^{-1}
Charge of electron	e	1.602×10^{-19} C
Permeability of vacuum	μ_0	$4\pi \times 10^{-7}$ N/A^2
Permitivity of vacuum	ϵ_0	8.85×10^{-12} F/m
Coulomb constant	$\frac{1}{4\pi\epsilon_0}$	9×10^{9} N m^2/C^2
Faraday constant	F	96485 C/mol
Mass of electron	m_e	9.1×10^{-31} kg
Mass of proton	m_p	1.6726×10^{-27} kg
Mass of neutron	m_n	1.6749×10^{-27} kg
Atomic mass unit	u	1.66×10^{-27} kg
Atomic mass unit	u	931.49 MeV/c^2
Stefan-Boltzmann constant	σ	5.67×10^{-8} W/(m^2 K^4)
Rydberg constant	R_∞	1.097×10^{7} m^{-1}
Bohr magneton	μ_B	9.27×10^{-24} J/T
Bohr radius	a_0	0.529×10^{-10} m
Standard atmosphere	atm	1.01325×10^{5} Pa
Wien displacement constant	b	2.9×10^{-3} m K

Floating Point

- Representation for non-integral numbers
  - Including very small and very large numbers
- Like scientific notation
  - $\mathbf{-2}.34 \times \mathbf{10}^{56}$
  - $\mathbf{+0}.002 \times \mathbf{10}^{-4}$
  - $\mathbf{+987}.02 \times \mathbf{10}^{9}$

Normalized?

Not Normalized?

- In binary
  - $\pm 1.xxxxxxx_2 \times 2^{yyyy}$
- Types `float` and `double` in C for different accuracy

Floating Point

Three fields in Floating Point Numbers

Sign bit	**Exponent**	**Significand or Mantissa**

- +/- .significand x $2^{exponent}$
- Point is actually fixed somewhere between sign bit and body of mantissa
- Exponent indicates place value (point position)

Floating Point Representation

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

S = Sign

F = Fraction (fixed point number)

usually called **Mantissa** or **Significand**

E = Exponent (positive or negative integer)

- How to divide a word into S, F and E?
- How to represent S, F and E?

IEEE 754 Standard Specific

- Standard for floating-point storage
- 32 and 64 bit standards
- 8 and 11 bit exponent respectively
- Extended formats (both mantissa and exponent) for intermediate results
- Representation: sign, exponent, faction
  - 0: 0, 0, 0
  - -0: 1, 0, 0
  - Special:
    - Plus infinity: 0, all 1s, 0
    - Minus infinity: 1, all 1s, 0
    - NaN; 0 or 1, all 1s, =! 0. etc.

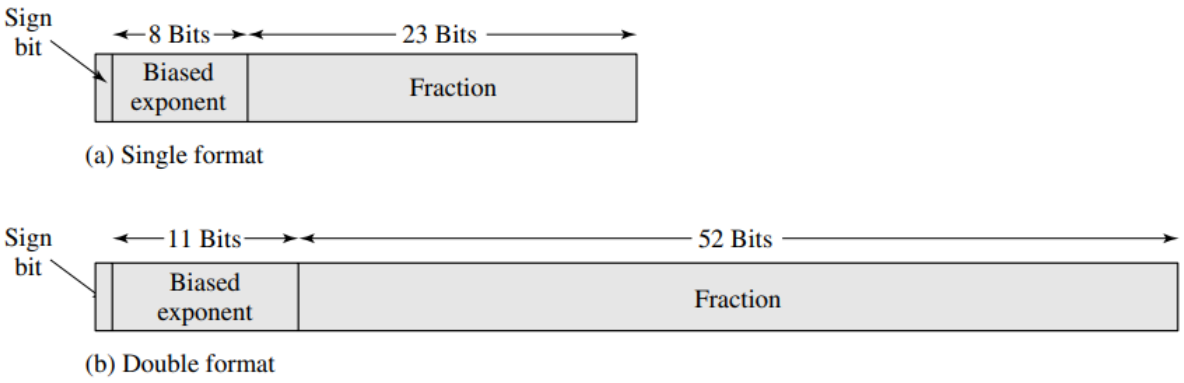


Figure 9.21 IEEE 754 Formats

Floating Point Standard

- Defined by **IEEE Std 754-1985**
- Developed in response to divergence of representations
  - Portability issues for scientific code
- Now almost universally adopted
- Three representations in latest IEEE 754 standard:
  - **Half precision** (16-bit)
  - **Single precision** (32-bit)
  - **Double precision** (64-bit)

IEEE Floating-Point Format

	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: sign bit (0 $\Rightarrow$ non-negative, 1 $\Rightarrow$ negative)
- 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 = 1203

IEEE 754 Precision types

Name	Significand Bits	Exponent Bits	Exponent Bias
Half Precision	11	5	+15
Single Precision	24	8	+127
Double Precision	53	11	+1023
Quadruple Precision	113	15	+16383

Double precision	0100000000001001001000011111101101010100010001000010110100011000
Single precision	01000000010010010000111111011011
Half precision	0100001001001000

Floating-Point Example

- What number is represented by the single-precision float

  11000000101000…00

  - S = 1
  - Fraction = $01000…00_2$
  - Fxponent = 10000001_2 = 129
- $x = (–1)^1 \times (1 + 01_2) \times 2^{(129 – 127)}$

  $= (–1) \times 1.25 \times 2^2$

  $= –5.0$

Normalization

- FP numbers are usually normalized
- i.e. Exponent is adjusted so that leading bit (MSB) of mantissa is 1
- Since it is always 1 there is no need to store it
- (Scientific notation where numbers are normalized to give a single digit before the decimal point e.g. 3.123×10^3)

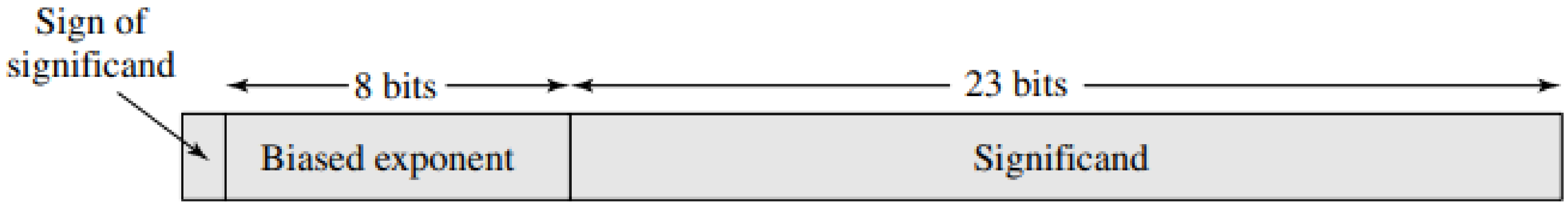
Floating-Point Example

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

Some Examples of Converting to FP

- Convert given Decimal number to Floating Point Representation
- Convert to Decimal Representation from Floating Point

Some Floating-Point Conversions



(a) Format

$$
\begin{aligned}
1.1010001 \times 2^{10100} &= 0\ 10010011\ 10100010000000000000000 = 1.6328125 \times 2^{20} \\
-1.1010001 \times 2^{10100} &= 1\ 10010011\ 10100010000000000000000 = -1.6328125 \times 2^{20} \\
1.1010001 \times 2^{-10100} &= 0\ 01101011\ 10100010000000000000000 = 1.6328125 \times 2^{-20} \\
-1.1010001 \times 2^{-10100} &= 1\ 01101011\ 10100010000000000000000 = -1.6328125 \times 2^{-20}
\end{aligned}
$$

(b) Examples

Figure 9.18 Typical 32-Bit Floating-Point Format

Examples of Floating Point Representation

$-(13.45)_{10}$
$= (1101.01\ 1100\ 1100\ 1100\ \ldots\ldots..)^2$; this is un-normalized
$= (1.10101\ 1100\ 1100\ 1100\ 1100\ 1) \times 2^3$; normalized
Fraction part is 10101 1100 1100 1100 1
Biased Exponent is 3+127 = 130
Sign = 1

5.0345
= 101 . 0000 1000 1101 0100 1111 110; this is un-normalized
$= 1.\ 01\ 0000\ 1000\ 1101\ 0100\ 1111\ 110 \times 2^2$; normalized
Biased Exponent = 2+127 = 129 = $(1000\ 0001)_2$
Fraction = 01 0000 1000 1101 0100 1111 110
Sign = 0

Single-Precision **Range**

- Exponents 00000000 and 11111111 **reserved**
- Smallest value
  - Exponent: 00000001
    $\Rightarrow$ actual exponent = 1 – 127 = –126
  - Fraction: 000…00 $\Rightarrow$ significand = 1.0
  - $\pm 1.0 \times 2^{-126} \approx \pm 1.2 \times 10^{-38}$
- Largest value
  - exponent: 11111110
    $\Rightarrow$ actual exponent = 254 – 127 = +127
  - Fraction: 111…11 $\Rightarrow$ significand ≈ 2.0
  - $\pm 2.0 \times 2^{+127} \approx \pm 3.4 \times 10^{+38}$

Double-Precision **Range**

- **Exponents 0000…00 and 1111…11 reserved**
- Smallest value
  - Exponent: 00000000001
    $\Rightarrow$ actual exponent = 1 – 1023 = –1022
  - Fraction: 000…00 $\Rightarrow$ significand = 1.0
  - $\pm 1.0 \times 2^{-1022} \approx \pm 2.2 \times 10^{-308}$
- Largest value
  - Exponent: 11111111110
    $\Rightarrow$ actual exponent = 2046 – 1023 = +1023
  - Fraction: 111…11 $\Rightarrow$ significand ≈ 2.0
  - $\pm 2.0 \times 2^{+1023} \approx \pm 1.8 \times 10^{+308}$

Floating-Point Precision

- Relative precision
  - all fraction bits are significant
  - Single: approx 2^{-23}
    - Equivalent to $23 \times \log_{10}2 \approx 23 \times 0.3 \approx 6$ decimal digits of precision
  - Double: approx 2^{-52}
    - Equivalent to $52 \times \log_{10}2 \approx 52 \times 0.3 \approx 16$ decimal digits of precision

Expressible Numbers

- Negative numbers between $-(2 - 2^{-23}) \times 2^{128}$ and -2^{-127}
- Positive numbers between 2^{-127} and $(2 - 2^{-23}) \times 2^{128}$

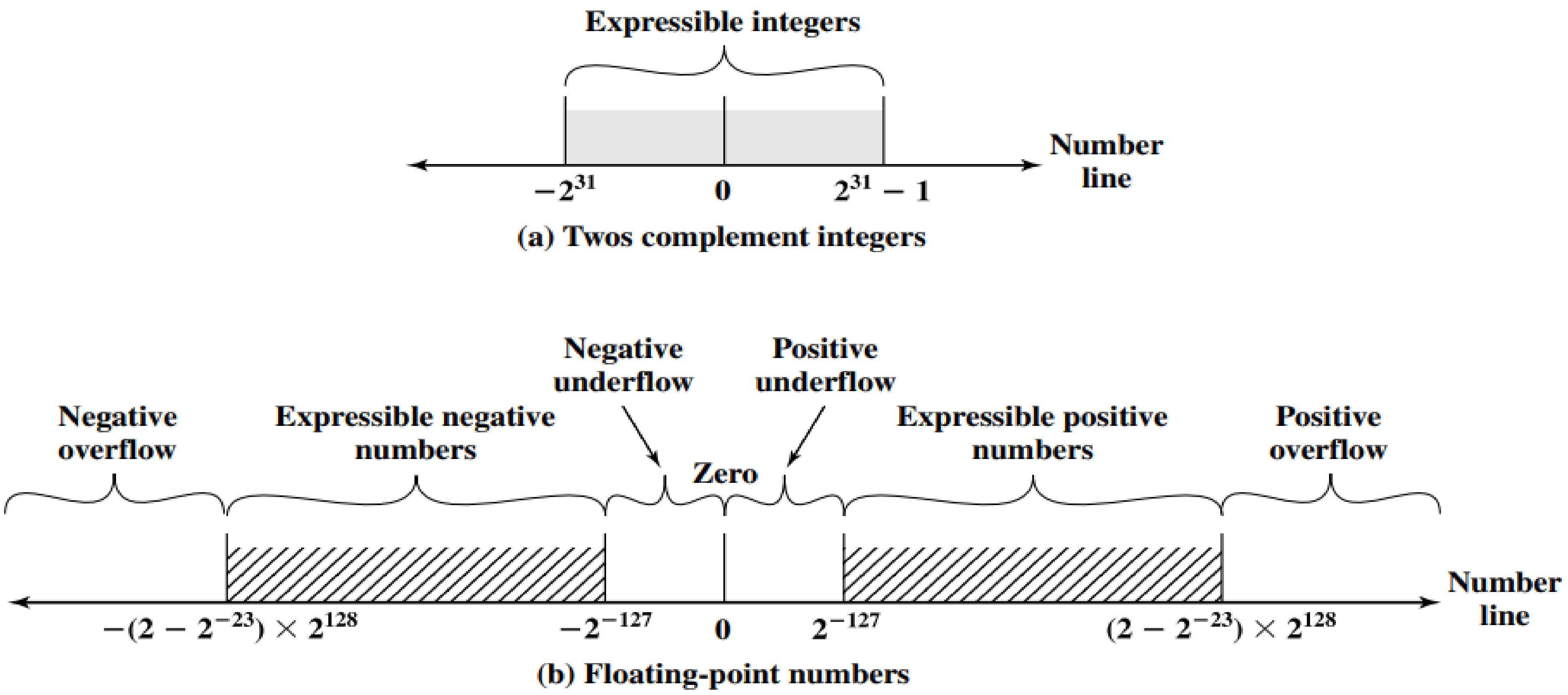


Figure 9.19 Expressible Numbers in Typical 32-Bit Formats

More about range of Floating-Point numbers

Five regions on the number line are not included in these ranges:

- Negative numbers less than $-(2 - 2^{-23}) \times 2^{128}$, called **negative overflow**
- Negative numbers greater than 2^{-127}, called **negative underflow**
- Zero
- Positive numbers less than 2^{-127}, called **positive underflow**
- Positive numbers greater than $(2 - 2^{-23}) \times 2^{128}$, called **positive overflow**

Overflow and Underflow in Floating-Point

largest positive/negative number (SP) =

$\pm(2 - 2^{-23}) \times 2^{127} \cong \pm 2 \times 10^{38}$

smallest positive/negative number (SP) =

$\pm 1 \times 2^{-126} \cong \pm 2 \times 10^{-38}$

Largest positive/negative number (DP) =

$\pm(2 - 2^{-52}) \times 2^{1023} \cong \pm 2 \times 10^{308}$

Smallest positive/negative number (DP) =

$\pm 1 \times 2^{-1022} \cong \pm 2 \times 10^{-308}$

Uneven Distribution in Floating Point

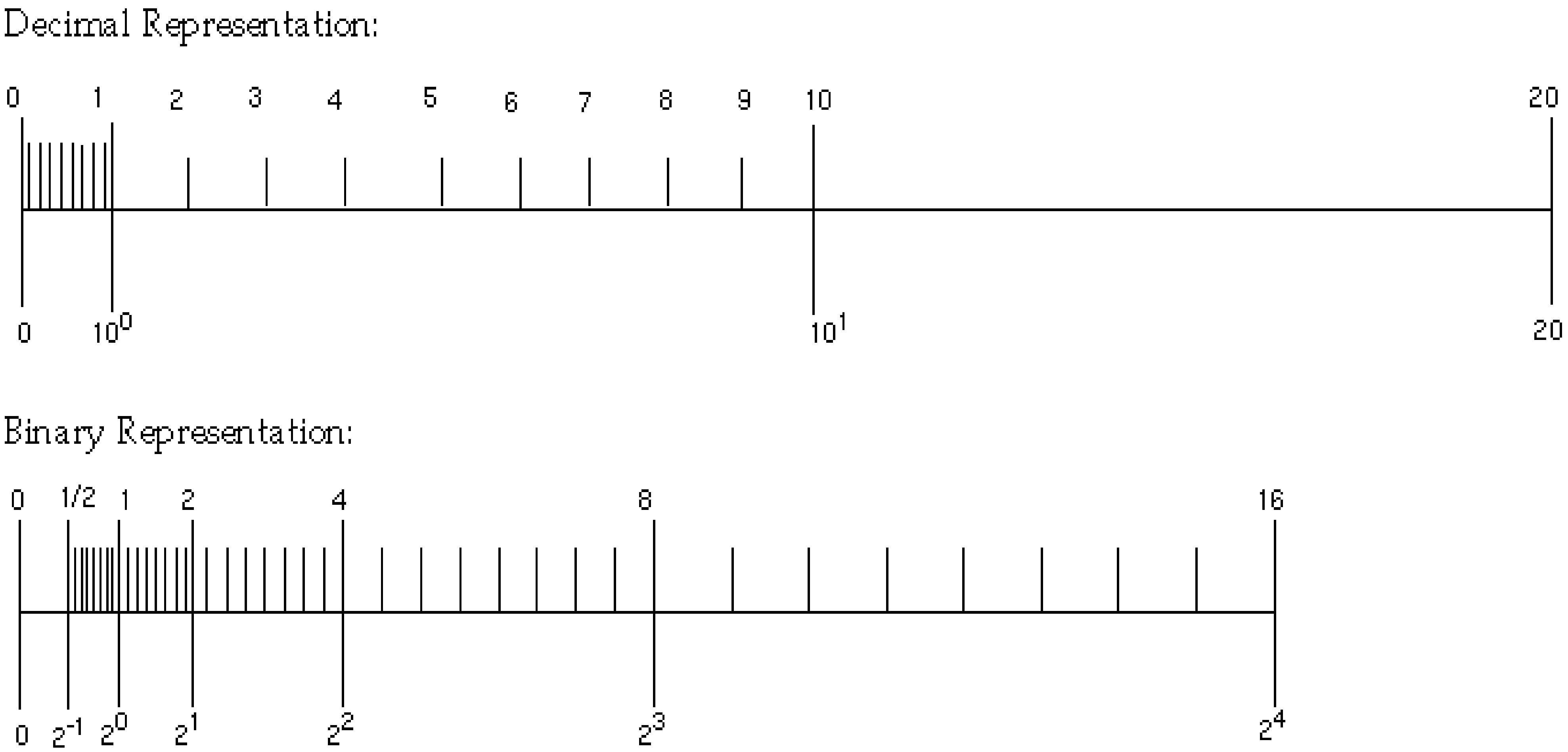
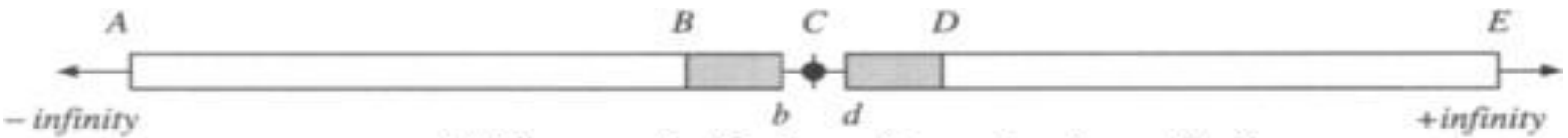


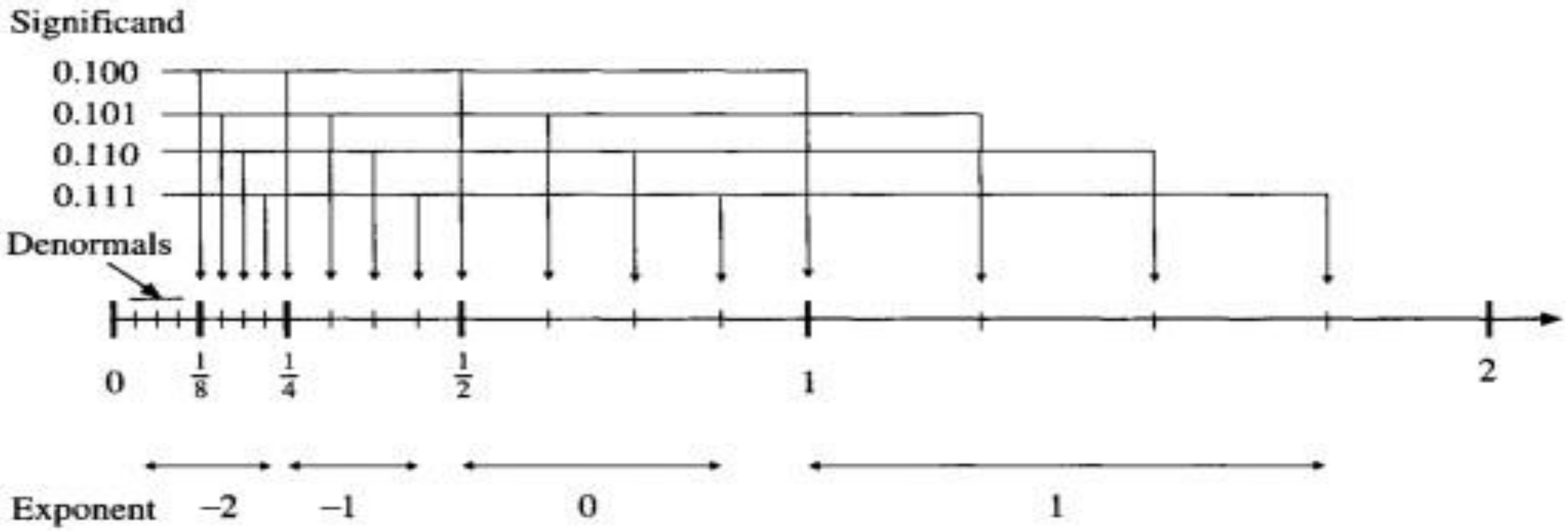
FIGURE 2-5 Comparison of a Set of Numbers Defined by Digital and Binary Representation

Floating Point – **Denormals** and Distribution



[A, B] — negative floating-point numbers (normalized)
[D, E] — positive floating-point numbers (normalized)
(B, b] & [d, D) — denormals
C — zero
> E — positive overflow
< A — negative overflow
(B, C) — negative underflow (normalized)
(C, D) — positive underflow (normalized)

(a)



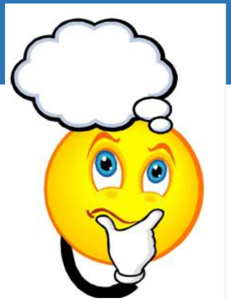
(b)

Special Representation – Subnormal

Zero: Zero is a special value denoted with an exponent field of 0 and a mantissa of 0.
Denormalized: If the exponent is all zeros, but the mantissa is not then the value is a *denormalized* number
Infinity: The values +infinity and -infinity are denoted with an exponent of all ones and a mantissa of all zeros. The sign bit distinguishes between negative infinity and positive infinity. Being able to denote infinity as a specific value is useful because it allows operations to continue past overflow situations. Operations with infinite values are well defined in IEEE.
Indeterminate: The value *indeterminate* is represented by an exponent of all ones, a mantissa with a leading one followed by all zeros, and a sign bit of one. This value is used to represent results that are indeterminate, such as (infinity - infinity), or (0 x infinity).
Not A Number: The value NaN (*Not a Number*) is used to represent a value that is an error of some form. This is represented with an exponent field of all ones and a zero sign bit or a mantissa that it not 1 followed by zeros.

Type	Exp	Fraction	Sign
Positive Zero	0	0	0
Negative Zero	0	0	1
Denormalised numbers	0	non zero	any
Normalised numbers	$1..2^e - 2$	any	any
Infinities	$2^e - 1$	0	any
NaN	$2^e - 1$	non zero	any

IEEE 754 Floating-Point Representation



Single Precision IEEE 754

Sign	Exponent	Fraction
1 bit	**8 bits**	**23 bits**

G R S

Guard Bits

32 bits

For Accuracy

Double Precision IEEE 754

Sign	Exponent	Fraction
1 bit	**11 bits**	**52 bits**

G R S

Guard Bits

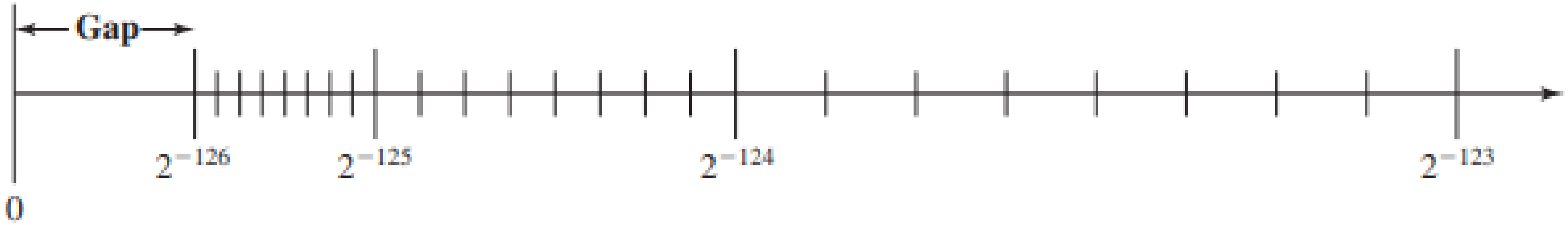
64 bits

Floating-Point Arithmetic – Basics

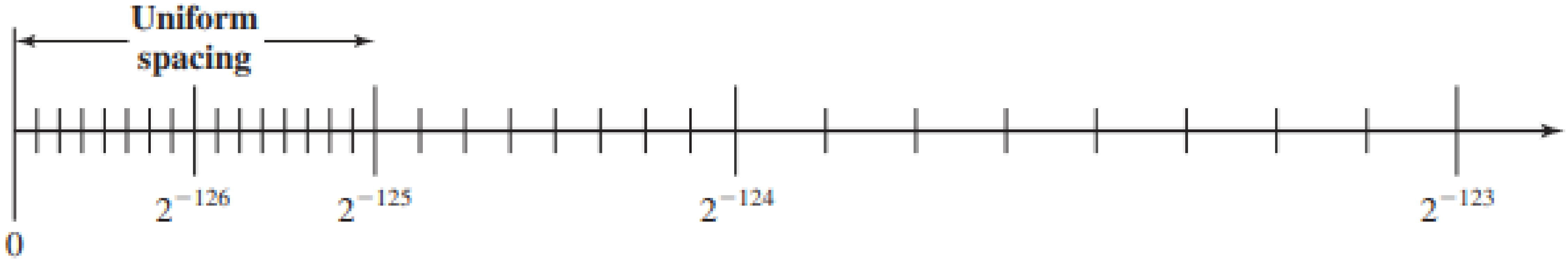
$N=(-1)^S \times (1+F) \times 2^E$

- A signed-magnitude system for the fractional part and a biased notation for the exponent
- Three subfields
  - Sign S
  - Fraction F (or Significand or Mantissa)
  - Exponent E
- Sign bit is 0 for positive numbers, 1 for negative numbers
- Fractions always start from **1**.xxxx, hence the integer **1** is not written (register has xxxx)
- Exponent is biased by +127 (add 127 to whatever is in register bits)
- Normalize: Express numbers is the standard format by shifting of bits and adding / subtracting from Exponent register

Subnormal or Denormal Numbers



(a) 32-bit format without subnormal numbers



(b) 32-bit format with subnormal numbers

Rounding Policy

ROUNDING Another detail that affects the precision of the result is the rounding policy. The result of any operation on the significands is generally stored in a longer register. When the result is put back into the floating-point format, the extra bits must be eliminated in such a way as to produce a result that is close to the exact result. This process is called **rounding**.

A number of techniques have been explored for performing rounding. In fact, the IEEE standard lists four alternative approaches:

- **Round to nearest:** The result is rounded to the nearest representable number.
- **Round toward $+\infty$:** The result is rounded up toward plus infinity.
- **Round toward $-\infty$:** The result is rounded down toward negative infinity.
- **Round toward 0:** The result is rounded toward zero.

Let us consider each of these policies in turn. **Round to nearest** is the default rounding mode listed in the standard and is defined as follows: The representable value nearest to the infinitely precise result shall be delivered.

Accurate Arithmetic

- IEEE Std 754 specifies additional rounding control
  - Extra bits of precision (**guard**, **round**, **sticky**)
  - Choice of rounding modes
  - Allows programmer to fine-tune numerical behavior of a computation
- Not all FP units implement all options
  - Most programming languages and FP libraries just use defaults
- Trade-off between hardware complexity, performance, and market requirements

Accuracy in Floating Point

- Precision is lost when some bits are shifted to right of the rightmost bit or are thrown
- Three extra bits are used internally -

  G (guard), R (round) and S (sticky)
  - G and R are simply the next two bits after LSB
  - S = 1 iff any bit to right of R is non-zero

1.	11010110101100010110110	GRS

Rounding Scheme in Floating Point

- if G=1 & R=1, add 1 to LSB
- if G=0 & R=0 or 1, no change
- if G=1 & R=0, look at S
  - if S=1, add 1 to LSB
  - if S=0, round to the nearest "even"

    i.e., add 1 to LSB if LSB = 1

Another view of Rounding Scheme

G Guard	R Round	S Sticky	Rounding Applied
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

S = OR (All bits in S and to the right of S)

Round to Even:
If S = 0, do nothing
If S = 1, add +1

The Use of Guard Bits

$$\begin{aligned} x &= 1.000\ldots\ldots00 \times 2^{1} \\ \underline{-y} &= \underline{0.111\ldots\ldots11} \times 2^{1} \\ z &= 0.000\ldots\ldots01 \times 2^{1} \\ &= 1.000\ldots\ldots00 \times 2^{-22} \end{aligned}$$

(a) Binary example, without guard bits

$$\begin{aligned} x &= .100000 \times 16^{1} \\ \underline{-y} &= \underline{.0FFFFF} \times 16^{1} \\ z &= .000001 \times 16^{1} \\ &= .100000 \times 16^{-4} \end{aligned}$$

(c) Hexadecimal example, without guard bits

$$\begin{aligned} x &= 1.000\ldots\ldots00\ 0000 \times 2^{1} \\ \underline{-y} &= \underline{0.111\ldots\ldots11\ 1000} \times 2^{1} \\ z &= 0.000\ldots\ldots00\ 1000 \times 2^{1} \\ &= 1.000\ldots\ldots00\ 0000 \times 2^{-23} \end{aligned}$$

(b) Binary example, with guard bits

$$\begin{aligned} x &= .100000\ 00 \times 16^{1} \\ \underline{-y} &= \underline{.0FFFFF\ F0} \times 16^{1} \\ z &= .000000\ 10 \times 16^{1} \\ &= .100000\ 00 \times 16^{-5} \end{aligned}$$

(d) Hexadecimal example, with guard bits

Floating-Point Arithmetic Operations

Floating-Point Numbers	Arithmetic Operations
$X = X_S \times B^{X_E}$ $Y = Y_S \times B^{Y_E}$	$\left.\begin{array}{l} X + Y = (X_S \times B^{X_E - Y_E} + Y_S) \times B^{Y_E} \\ X - Y = (X_S \times B^{X_E - Y_E} - Y_S) \times B^{Y_E} \end{array}\right\} X_E \le Y_E$ $X \times Y = (X_S \times Y_S) \times B^{X_E + Y_E}$ $\dfrac{X}{Y} = \left(\dfrac{X_S}{Y_S}\right) \times B^{X_E - Y_E}$

Floating-Point Addition – Decimal example

- Consider a 4-digit decimal example
  - $9.999 \times 10^{1} + 1.610 \times 10^{-1}$
- 1. Align decimal points
  - Shift number with smaller exponent
  - $9.999 \times 10^{1} + 0.016 \times 10^{1}$
- 2. Add significands
  - $9.999 \times 10^{1} + 0.016 \times 10^{1} = 10.015 \times 10^{1}$
- 3. Normalize result & check for over/underflow
  - 1.0015×10^{2}
- 4. Round and renormalize if necessary
  - 1.002×10^{2}

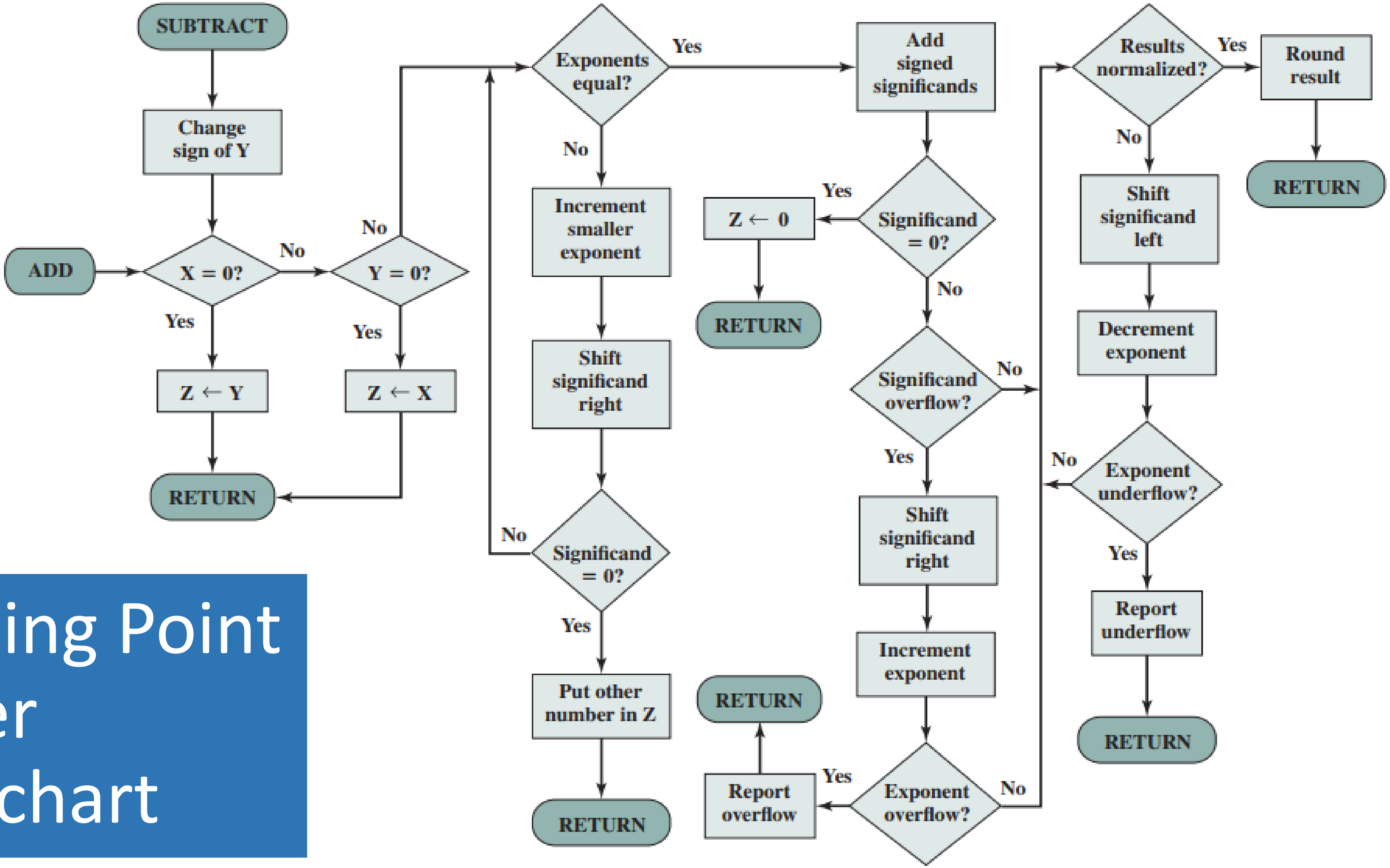
Floating-Point Addition – Binary example

- Consider a 4-digit binary example
  - $1.000_2 \times 2^{-1}$ **+** $-1.110_2 \times 2^{-2}$ (0.5 + –0.4375)
- 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

FP Arithmetic Add / Subtract

- Check for zeros
- Align significands (adjusting exponents)
- Add or subtract significands
- Normalize result

Floating Point Adder Flowchart



Floating-Point Addition and Subtraction (Z ← X ± Y)

Readings

- Chapter 3, P&H Textbook