SECR2033
Computer Organization and Architecture

Lecture slides prepared by “Computer Organization and Architecture”, 9/e, by William Stallings, 2013.

Module 2

Data Representation in Computer Systems

Objectives:

- To understand the fundamentals of **numerical data** representation and manipulation in digital computers.
- To master the skill of converting between various **radix systems**.
- To understand how **errors** can occur in computations because of overflow and truncation.
- To understand the fundamental concepts of **floating-point** representation.

Module 2

Data Representation in Computer Systems

2.1 Introduction

2.2 Fixed-Number (Integer) Representation

2.3 Fixed-Number (Integer) Arithmetic

2.4 Floating-Points Representation

2.5 Floating-Points Arithmetic

2.6 Summary

Module 2

Data Representation in Computer Systems

- Representation
- Components
- Normalized & Unnormalized
- Format: A Simple Model
- The IEEE-754 Floating-Point Standard

2.4 Floating-Point Representation

2

- The signed magnitude, one's complement, and two's complement representation that we have just presented deal with integer values only.
- Without modification, these formats are not useful in scientific or business applications that deal with real number values.
- Floating-point representation solves this problem.
- Known as *real number* in mathematics.

Representation

- Floating-point numbers allow an arbitrary number of decimal places to the right of the decimal point.
  - **Examples**:
  -

$$5 \times 25 = 125$$
$$0.5 \times 0.25 = 0.125$$

→ often expressed in scientific notation as:

$$125 = 1.25 \times 10^{2}$$
$$0.125 = 1.25 \times 10^{-1}$$

Sign

Significand/Fraction/Mantissa

Base/Radix

Exponent

Components

(Decimal) $+743.059 \times 10^{-63}$

(Binary) -101.001×2^{011}

Fraction and *exponent* can be +ve or –ve

Normalized & Unnormalized

- In generalized normalization (like in mathematics), a floating point number is said to be *normalized* if the number after the radix point is a non-zero value.
- *Unnormalized* floating number is when the number after the radix point is '0'.
- **Examples**:

Radix point

1.03×10^{-9} (*Unnormalized*)

0.10×10^{8} (*Normalized*)

45.07×10^{-10} (*Unnormalized*)

0.1234×10^{16} (*Normalized*)

Normalization Process

- *Normalization* is the process of deleting the zeroes until a non-zero value is detected.

- **Examples**:

$$0.00743 \times 10^4 = 0.743 \times 10^{4-2}$$
$$= 0.743 \times 10^2$$

$$45.09 \times 10^4 = 0.4509 \times 10^{4+2}$$
$$= 0.4509 \times 10^6$$

A rule of thumb:

- moving the radix point to the right → subtract exponent
- moving the radix point to the left → add exponent

Examples:

- Decimal:

$$743.09 \times 10^{61} = 0.74309 \times 10^{61+3}$$
$$= 0.74309 \times 10^{64}$$

Can represent the exponent as binary.

- Binary:

$$10.0111_2 \times 2^{-110011} = 0.100111_2 \times 2^{(-110011)+010}$$
$$= 0.100111_2 \times 2^{-110001}$$

$$0.00011011_2 \times 2^{9} = 0.11011_2 \times 2^{9-3}$$
$$= 0.100111_2 \times 2^{6}$$

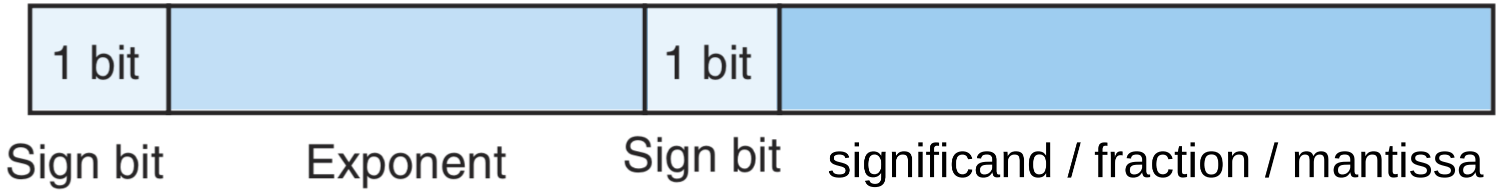
Can represent the exponent as decimal.

Format : A Simple Model

- In digital computers, floating-point numbers consist of three parts: a *sign bit*, an *exponent* part (representing the exponent on a power of 2), and a fractional part called a *significand.*
- The general form of *normalized* floating-point is:

$$\pm 0.Fraction \times Base^{\pm exponent}$$

- In binary form:



Linda Null and Julia Lobur (2003). *The Essentials of Computer Organization and Architecture.* United States: Jones and Bartlett Publishers. p.56.

The 2 signs bit are not good for design as it incurs extra cost.
→ *need new representation.*

- In binary form:

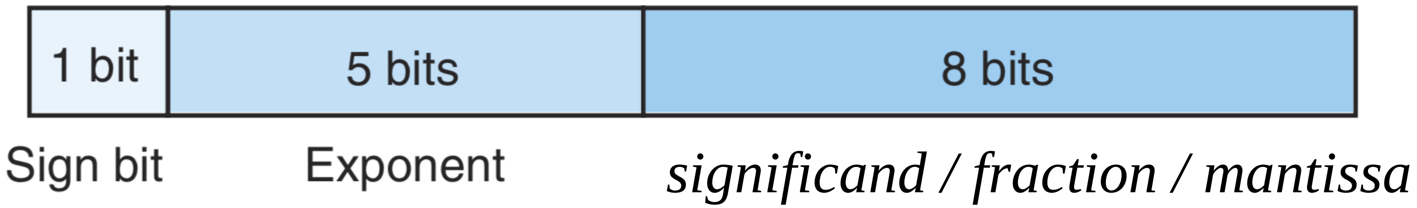
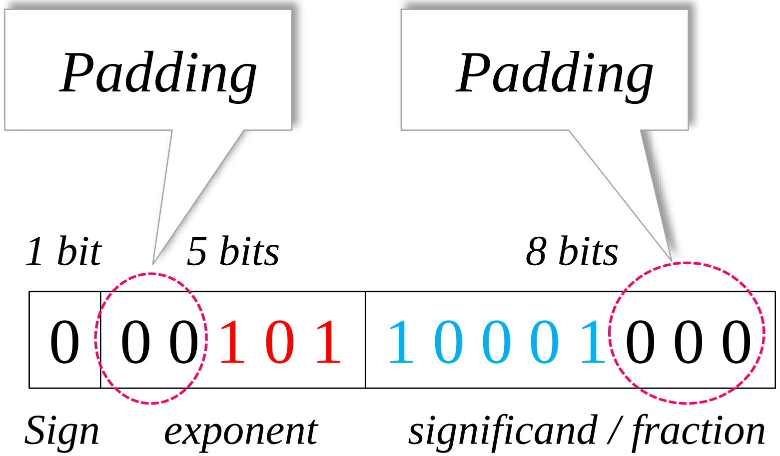


Figure: Example of 14-bit model for floating-point format.

- The next few examples and exercises will use this simple model before the discussion on the IEEE-754 floating-point standard.

Example 17:

Store the decimal number 17 in this model.



Solution:

(Unnormalized Binary):

$17 = 10001.0 \quad x\ 2^0$

(Normalized Binary):

$= \quad 1000.1 \quad x\ 2^1$

$= \quad 100.01 \quad x\ 2^2$

$= \quad 10.001 \quad x\ 2^3$

$= \quad 1.0001 \quad x\ 2^4$

$= \quad 0.10001 \quad x\ 2^5$

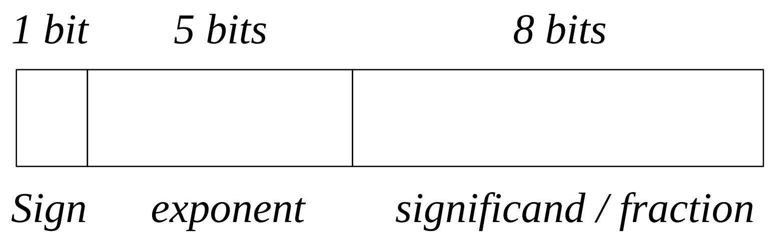
Rule of thumb:

- the *exponent* is always padded to the left (←).
- the *fraction* is always padded to the right (→).

Activity 6

Exercise 2.7:

Based on the following format, write the decimal number 65536 in the floating-point representation and store it in the floating-point format. Show your working.



Linda Null and Julia Lobur (2003). *The Essentials of Computer Organization and Architecture.* United States: Jones and Bartlett Publishers. p.57.

This is used unless the IEEE standard is mentioned

Biased Exponent

- One obvious problem with this model is that negative exponents cannot be represented.
- **Solution**: *biased* exponent

1 bit	*n bits*	
± sign	*biased exponent (E_b)*	*significand / fraction*

Where,

E_b = biased exponent

n = bits of exponent format

(i.e. the word format)

Biased value, $b = 2^{n-1}$

Normalized exponent, $e' = E_b - b$

Biased exponent, $E_b = e' + b$

- The bias value → a number near the middle of the range of possible values that we select to represent zero.

- Typically, the bias value (*b*) is equals (2^{n-1}), where n is the number of bits in the binary exponent.

  - *positive* value : Any number larger than (2^{n-1}) in the exponent field → bias value + exponent
  - *negative* value : Any number less than (2^{n-1}) in the exponent field → bias value - exponent

Linda Null and Julia Lobur (2003). *The Essentials of Computer Organization and Architecture.* United States: Jones and Bartlett Publishers. p.57.
William Stallings (2013). *Computer Organization and Architecture: Designing for Performance* (9th Edition). United States: Pearson Education Limited, p.328.

- The steps of conversion to floating-point:

1. Change to binary (if given decimal number).
2. Normalized the number.
3. Change the number to *biased exponent*.
4. Form the floating-point format (3 fields) .

Example 18:

Returning to *Example 17* that storing: $17 = 0.10001 \times 2^5$

0	0 0 1 0 1	1 0 0 0 1 0 0 0
Sign	*exponent*	*significand / fraction*

- The *bias* value, $b = 2^{5-1} = 2^4 = 16$
- The *biased exponent*, E_b is now = $e' + b$ = 5 + 16 = 21

0	1 0 1 0 1	1 0 0 0 1 0 0 0
Sign	*Biased exponent*	*significand / fraction*

Example 19:

Store the decimal number 0.25 in the floating-point representation.

Show your working.

Solution:

1.Convert into binary using repetitive multiplication:

$$0.25 \times 2 = 0.5$$
$$0.5 \times 2 = 1.0$$
$$0.25_{10} = 0.01_2$$

2.Normalized Binary:

$$0.25_{10} = 0.01 \times 2^{0}$$
$$= 0.1 \times 2^{-1}$$

3. Biased exponent

- The *bias* value, $b = 2^{5-1} = 16$
- The *biased exponent*, E_b is now 16 + (–1) = 15

4. Floating point format

1 bit	*5 bits*	*8 bits*
0	0 1 1 1 1	1 0 0 0 0 0 0 0
Sign	E_b	*significand / fraction*

Converting decimal fraction to binary (express 21.1_{10} into binary)

Start with integer part (repetitive division with 2)

$21.1_{10} = ?_2$

21/2 = 10 r=1

10/2 = 5 r=0

5/2 = 2 r=1

2/2 = 1 r=0

$21_{10} = 10101_2$

Stop when r=0

Then with fraction part (repetitive multiplication with 2)

$21.1_{10} = 10101.0001100_2$

0.1x2 = 0.2 -> 0

0.2x2 = 0.4 -> 0

0.4x2 = 0.8 -> 0

0.8x2 = 1.6 -> 1

0.6x2 = 1.2 -> 1

0.2x2 = 0.4 -> 0

0.4x2 = 0.8 -> 0

Fraction will never become 0 and sequence shows recurring pattern, so we stop

Exercise 2.8:

Convert the answer in previous example to a *normalized* 14-bit format with a bias of 16. Show your working.

$$0.1110111 \times 2^{2}$$

0	0 0 0 1 0	1 1 1 0 1 1 1 0
Sign	*exponent*	*significand / fraction*

- The *bias* value, $b = 2^{5-1} = 2^4 = 16$
- The *biased exponent*, E_b is now

$$E_b = e' + b = 2 + 16 = 18$$

$$18_{10} = 10010_2$$

1 bit	*5 bits*					*8 bits*							
0	1	0	0	1	0	1	1	1	0	1	1	1	0
S	E_b					*Fraction*							

Activity 7

2

Exercise 2.9:

Transform (–33.625_{10}) to floating point using the following format (radix 2). Show your working.

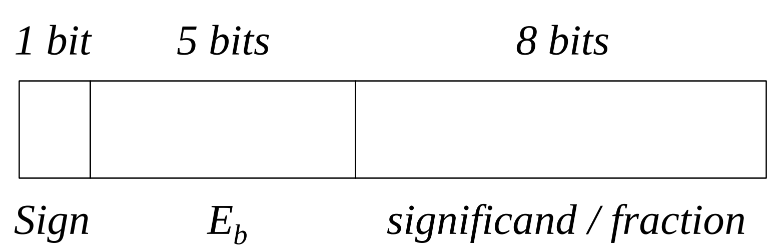
-33.625 = -3.3625 x10^{-1}

- The steps of conversion to floating-point:
  1. Change to binary (if given decimal number).
  2. Normalized the number.
  3. Change the number to *biased exponent*.
  4. Form the floating-point format (3 fields) .

1 bit	*5 bits*	*8 bits*
Sign	E_b	*significand / fraction*

Exercise 2.10:

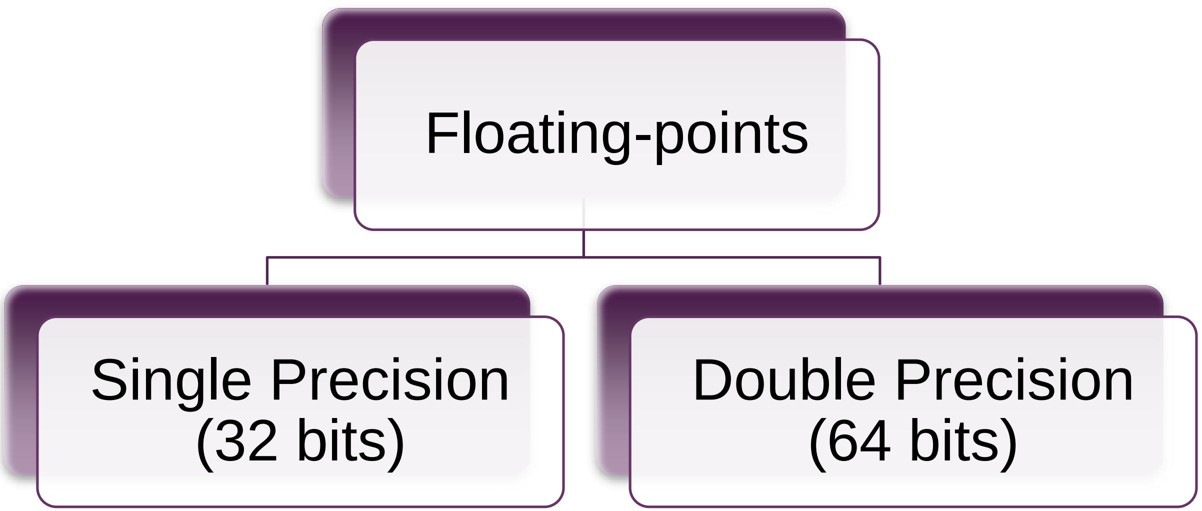
Transform (-0.03125_{10}) to floating point using the following format (radix 2). Show your working.



The IEEE-754 Floating-point Standard

(IEEE) Institute of Electrical and Electronic Engineers

- The floating-point model (non-standard) described before is for simplicity and conceptual understanding.
- This standard is officially known as IEEE-754 (1985):



Overview

- *Floating-point* → Computer arithmetic that represents numbers in which the binary point is not fixed:

$$1.xxxxxxxx_2 \times 2^{yyyy}$$

- A standard scientific notation for real number in normalized form.

- General form:

$$(-1)^S \times Fraction \times 2^{e'}$$

where $S = Sign$, $e' =$ Normalized exponent.

Patterson, D.A. and Hennessy, J.L. (2014). *Computer Organization and Design: The Hardware/Software Interface* (5th Edition). United States: Elsevier, p.197.

(a) Single Precision

- The representation of a MIPS 32-bit floating-point 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																						

- The sizes of *exponent* and *fraction* give MIPS computer arithmetic an extraordinary range:
  - Smallest fraction: $2.0_{10} \times 10^{-38}$
  - Largest number: $2.0_{10} \times 10^{38}$

- This is called *single-precision* = 1 word.

- Unfortunately, single precision floating-point is still possible for numbers to be too large:

  - *Overflow* → the *exponent* is too large.
  - *Underflow* → the negative *exponent* is too large.

- **Solution**:

  One way to reduce these problems, need another format that has larger exponent → *Double precision* floating-point.

(b) Double Precision

- The representation of a double precision floating-point number takes two MIPS words → 64 bits.

<table>
<tr><td>31</td><td>30</td><td>29</td><td>28</td><td>27</td><td>26</td><td>25</td><td>24</td><td>23</td><td>22</td><td>21</td><td>20</td><td>19</td><td>18</td><td>17</td><td>16</td><td>15</td><td>14</td><td>13</td><td>12</td><td>11</td><td>10</td><td>9</td><td>8</td><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td></tr>
<tr><td>s</td><td colspan="12">exponent</td><td colspan="19">fraction</td></tr>
<tr><td>1 bit</td><td colspan="12">11 bits</td><td colspan="19">20 bits</td></tr>
<tr><td colspan="32">fraction (continued)</td></tr>
<tr><td colspan="32">32 bits</td></tr>
</table>

  - Smallest fraction: $2.0_{10} \times 10^{-308}$
  - Largest number: $2.0_{10} \times 10^{308}$

Patterson, D.A. and Hennessy, J.L. (2014). *Computer Organization and Design: The Hardware/Software Interface* (5th Edition). United States: Elsevier, p.198.

The IEEE-754 Floating-point Standard

Representation

- This standard has greatly improved both the ease of porting floating-point programs and the quality of computer arithmetic.
- To pack even more bits into the *significand*, IEEE 754 makes the leading 1-bit of normalized binary numbers implicit.

- *Significand* : 1 + fraction
  - *Single precision* : 24 bits
  - *Double precision* : 53 bits

0 has no leading 1, it is given the reserved exponent value 0.

Patterson, D.A. and Hennessy, J.L. (2014). *Computer Organization and Design: The Hardware/Software Interface* (5th Edition). United States: Elsevier, p.198.

- The representation of normalized floating-point in IEEE standard:

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

Sign (S): plus (0) or minus (1)
Normalized exponent (e')

The IEEE-754 Floating-point Standard

Normalized & Unnormalized

$1.xxxxxxxx_2 \times 2^{yyyy}$: normalized form

- In IEEE standard normalization (used in computers), a floating point number is said to be *normalized* if there is only a single non-zero before the radix point.

- **Examples**:

$$1.03 \times 10^{-9} \qquad (Normalized)$$
$$0.10 \times 10^{8} \qquad (Unnormalized)$$
$$1.011 \times 2^{-011} \qquad (Normalized)$$
$$1.234 \times 10^{16} \qquad (Normalized)$$

Exercise 2.11:

Complete the table with the normalized binary number and its exponent respectively using single precision floating-point.

	Binary Values	Normalized as	*Exponent (e')*
(a)	1101.101		
(b)	0.00101		
(c)	1.0001		
(d)	10000011.0		

Biased Notation

- Placing the exponent before the fraction simplifies sorting of floating-point numbers using integer comparison instructions.
- However, using 2's complement in the exponent field makes a negative exponent look like a big number.

- **Example**: IEEE-754 Single Precision (32 bits)

$1.0_2 \times 2^{-1}$

Problem: e' = –1

1 bit	8 bits	23 bits
0	1 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
Sign	*Exponent*	*significand / fraction*

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

Sign (S): plus (0) or minus (1)

Biased Exponent (E_B)

Bit of Exponent (n)

Bias values, $B = (2^{n-1}) - 1$:

→ In *single precision* is 127

→ In *double precision* is 1023

- **Example**:

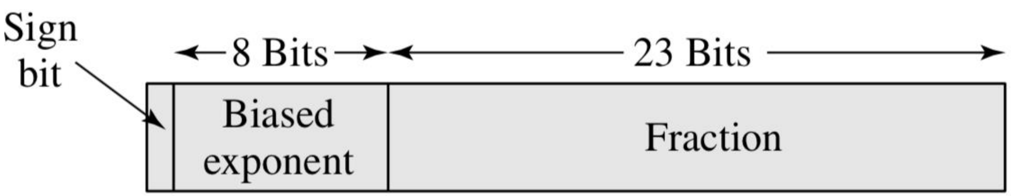
  $1.0_2 \times 2^{-1}$

$E_B = e' + \text{Bias}$

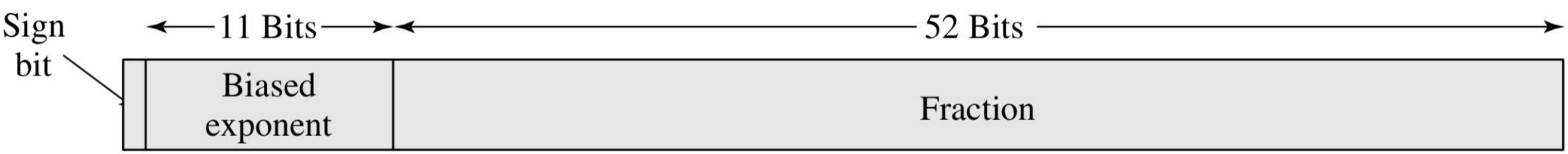
$= (-1) + 127$

$= 126_{10} = 01111\ 1110_2$

1 bit	8 bits	23 bits
0	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
Sign	*Biased Exponent*	*significand / fraction*



(a) Single format / Single-Precision / Single word



(b) Double format / Double-Precision / Double words

Figure: IEEE-754 floating-point format.

Activity 8

2

Exercise 2.12:

Complete the table with the *biased exponent* (E_B) and binary representation for each number using the type of floating-point respectively.

	Exponent (e')	*Biased Exponent (E_B)*			
		Single Precision		Double Precision	
		(Dec)	*(Bin)*	*(Dec)*	*(Bin)*
(a)	3				
(b)	– 3				
(c)	0				
(d)	7				

2

The IEEE-754 Floating-point Standard

Conversion of Decimal to Binary Floating-point

- To convert a decimal number to single or double precision floating point:
  - Step 1: Normalized.
  - Step 2: Determine sign bit.
  - Step 3: Determine *biased exponent*.
  - Step 4: Determine significand (fraction)

Example 20:

Convert 10.4_{10} to single precision floating-point.

- **Step 1**: Normalized from the bit value.

$10_{10} \rightarrow 1010_2$

$0.4 \times 2 = 0.8$
$0.8 \times 2 = 1.6$
$0.6 \times 2 = 1.2$
$0.2 \times 2 = 0.4$
.....

(Repetitive Multiplication)

$10.4_{10} = 1010.0110_2 \times 2^0$

$= 1.0100110_2 \times 2^{0+3}$

$= 1.0100110_2 \times 2^3$

- **Step 2**: Determine sign bit (S). $\rightarrow S = 0$

- **Step 3**: Determine the biased exponent, E_B →

$$E_B = e' + \text{bias} = 3 + 127 = 130_{10} = 1000\ 0010_2$$

- **Step 4**: Determine fraction.

Since all real no. in IEEE 754 is always in the form of $1.xxxxx_2\ 2^{yy}$ so we can drop the leading 1

❑ Drop the leading 1 of the fraction;

$$1.0100110_2 \times 2^{10000010} \rightarrow 0100110$$

❑ Expand (padding) to 23 bits;

$$01001100000000000000000 \qquad = 1.0100110_2 \times 2^3$$

❑ Complete the representation;

1 bit	*8 bits*	*23 bits*
0	1 0 0 0 0 0 1 0	0 1 0 0 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
Sign	*Biased Exponent*	*significand / fraction*

Activity 9 - (a) only

Exercise 2.14:

Convert the following number to single precision floating-point.

(a) (-33.625_{10})

(b) 0.03125_{10}

Activity 10

2

Exercise 2.15:

Complete the table with all *sign (S)*, *exponent (e')* and *fraction (F)* values if single precision floating-point applied.

	Binary Values	E_B (Decimal)	S	E_B (Binary)	*Fraction*
(a)	-1.11				
(b)	+1101.101				
(c)	-0.00101				
(d)	+100111.0				
(e)	+0.0000001101011				

Activity 11 – (a) only

Exercise 2.16:

Convert the following number to double precision floating-point.

(a) (-0.75_{10})

(b) 10.4_{10}

The IEEE-754 Floating-point Standard

Conversion of Binary Floating-point to Decimal

- To convert a single or double precision floating point to decimal number:

  - **Step 1**: Extract value of sign.
  - **Step 2:** Extract value of *biased exponent* & bias value.
  - **Step 3**: Extract value of fraction.
  - **Step 4**: Apply the basic equation.

Example 21:

What is the decimal number represented by this single precision float?

1 bit	*8 bits*	*23 bits*
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

Solution:

$E_B = e' + \text{Bias}$

- **Step 1**: Extract value of sign. → $S = 1$
- **Step 2**: Extract value of biased exponent, E_B → 10000001 = 129_{10}

  Bias value, B → $(2^{n-1} - 1) = 2^7 - 1 = 128 - 1 = 127_{10}$

$\rightarrow$ 01 x 2^0
$\rightarrow$ 0.1 x 2^{-1}
$\rightarrow$ 1.0 x 2^{-2}

S = 1
Fraction = 0.25
E_B = 129
B = 127

- **Step 3**: Extract value of fraction. $\rightarrow$ 1 x 2^{-2} $= \frac{1}{2^2} = \frac{1}{4} = 0.25_{10}$

- **Step 4**: Apply the basic equation.

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

$E_B = e' + \text{bias}$
$\rightarrow e' = E_B - B$

$= (-1)^1 \times (1 + 0.25) \times 2^{129-127}$

$= (-1) \times (1.25) \times 2^2$

$= (-1.25) \times 4$

$= -5.0_{10}$

Patterson, D.A. and Hennessy, J.L. (2014). *Computer Organization and Design: The Hardware/Software Interface* (5th Edition). United States: Elsevier, p.202.

Exercise 2.17:

What is the decimal number represented by this double precision float?

1 bit	*11 bits*	*52 bits*
0	0 1 1 1 1 1 1 1 1 1 0	0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 . . . 0

Module 2

Data Representation in Computer Systems

- Overview
- Addition
- Multiplication

2.5 Floating-Point Arithmetic

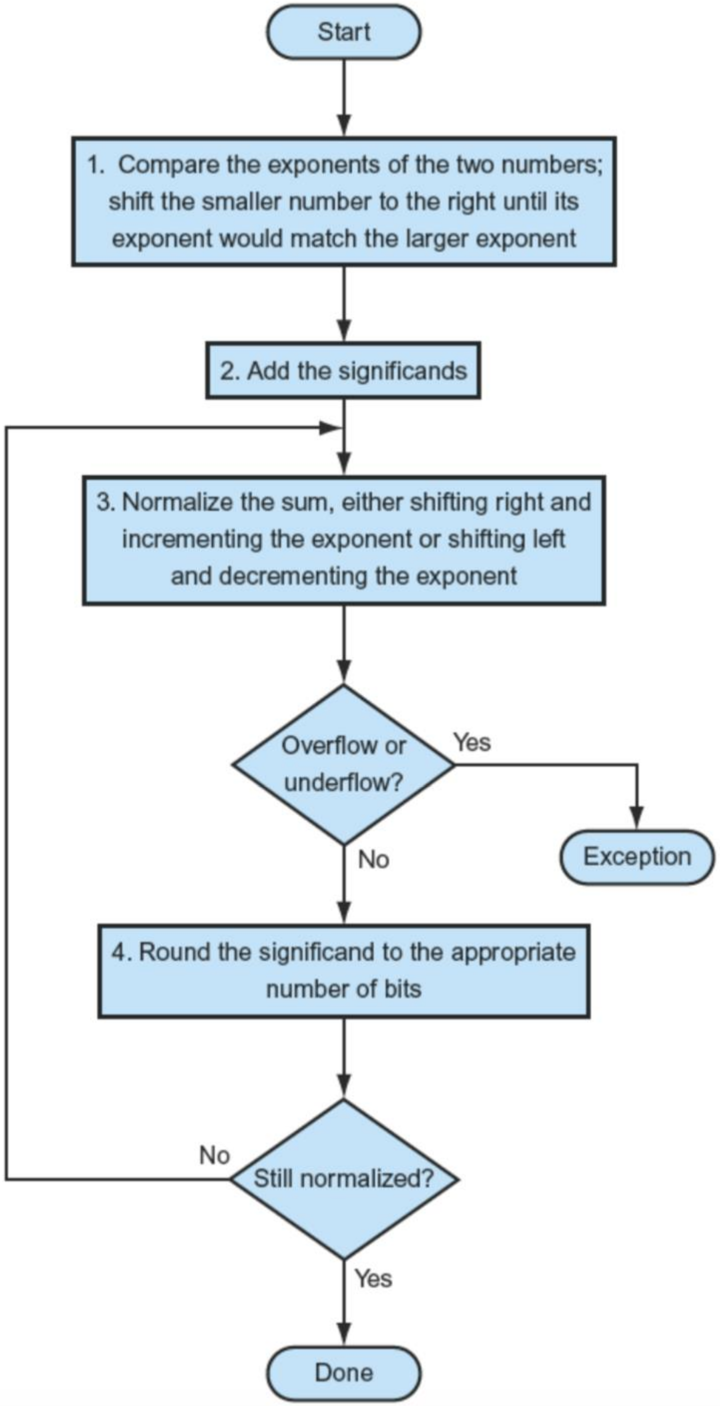
2

Overview

- For addition and subtraction, it is necessary to ensure that both operands have the same exponent value.
- This may require shifting the radix point on one of the operands to achieve alignment.
- Multiplication and division are more straightforward.

Addition

Flows



Biased Exponent ranges:

Single precision: -126 ~ 127

Double precision: - 1022 ~ 1023

Patterson, D.A. and Hennessy, J.L. (2014). *Computer Organization and Design: The Hardware/Software Interface (5th* Edition). United States: Elsevier, p.205.

2

- Assume 4 decimal digits for fraction and 2 decimal digits for exponent.
  - **Step 1**: Align the decimal point of the number that has the smaller *exponent*.
  - **Step 2**: Add the fraction.
  - **Step 3**: Normalize the sum.
  - **Step 4**: Round the fraction.
    (If the fraction does not fit in the space reserved for it, it has to be rounded off)
  - **Step 5**: Normalize it (if need be) .

Example 22:

Add these two decimal floating-point numbers. Assume that we can store only four decimal digits of the significand and two decimal digits of the exponent.

$(9.999_{10} \times 10^{1}) + (1.610_{10} \times 10^{-1}) = \_\_\_\_\_\__{10}$

Solution:

- **Step 1**: Align the decimal point of the number that has the smaller exponent.

$1.610_{10} \times 10^{-1}$

$= 1.610_{10} \times 10^{-1} \times 10^{2}$

$= 0.0161_{10} \times 10^{1}$

- **Step 2**: Add the fraction.

$$\begin{array}{r} 9.9990 \times 10^{1} \\ +\ 0.0161 \times 10^{1} \\ \hline 10.0151 \times 10^{1} \\ \hline \end{array}$$

- **Step 3**: Normalize the sum.

$$10.0151 \times 10^1 \rightarrow 1.00151 \times 10^2$$

- **Step 4:** Round the fraction (to 4 decimal digits for fraction).

$$1.00151 \times 10^2 \rightarrow 1.0015 \times 10^2$$

- **Step 5**: Normalize it (if need be).

No need as its normalized

Answer = 1.0015×10^2

Example 23:

Add these two binary floating-point numbers.

$$0.5_{10} + (-0.4375_{10}) = \_\_\_\_\_\__{2}$$

Solution: (Convert to binary)

$0.5 \times 2 = 1.0$

$0.4375 \times 2 = 0.875$
$0.875 \times 2 = 1.75$
$0.75 \times 2 = 1.5$
$0.5 \times 2 = 1.0$

$(0.1_2 \times 2^0) \times 2^{-1}$

$(-0.0111_2 \times 2^0) \times 2^{-2}$

$$(1.0_2 \times 2^{-1}) + (-1.11_2 \times 2^{-2})$$

- **Step 1**: Align the decimal point of the number that has the smaller exponent.

$$-1.11_2 \times 2^{-2}$$

$$= -1.11_2 \times 2^{-2} \times 2^{1}$$

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

- **Step 2**: Add the fraction.

~~1~~.~~00~~0 x 2^{-1}

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

$$0.001 \times 2^{-1}$$

- **Step 3**: Normalize the sum.

$$0.001 \times 2^{-1} \times 2^{-3} \rightarrow 1.0 \times 2^{-4}$$

- **Step 4:** Round the fraction (to 4 decimal digits for fraction).

$$1.0 \times 2^{-4} \rightarrow 1.0000 \times 2^{-4}$$

- **Step 5**: Normalize it (if need be).

No need as its normalized

Answer = 1.0000 x 2^{-4}

$$.(0 \times 2^{-1}) + (0 \times 2^{-2}) + (0 \times 2^{-3}) + (1 \times 2^{-4})$$

$$.(0.\frac{1}{2^1}) + (0.\frac{1}{2^2}) + (0.\frac{1}{2^3}) + (1.\frac{1}{2^4})$$

$$.(0.\frac{1}{2}) + (0.\frac{1}{4}) + (0.\frac{1}{8}) + (1.\frac{1}{16})$$

This sum in decimal is then:

$$1.0000 \times 2^{-4} = 0.0001000_2 = 0.0001_2$$

$$= \frac{1}{(2^4)_{10}} = \frac{1}{16_{10}} = 0.0625_{10}$$

Patterson, D.A. and Hennessy, J.L. (2014). *Computer Organization and Design: The Hardware/Software Interface (5th* Edition). United States: Elsevier, p.204.

Activity 12

2

Exercise 2.18:

Add these two binary floating-point numbers.

$$0.6015625_{10} + 0.78125_{10} = \_\_\_\_\_\__{2}$$

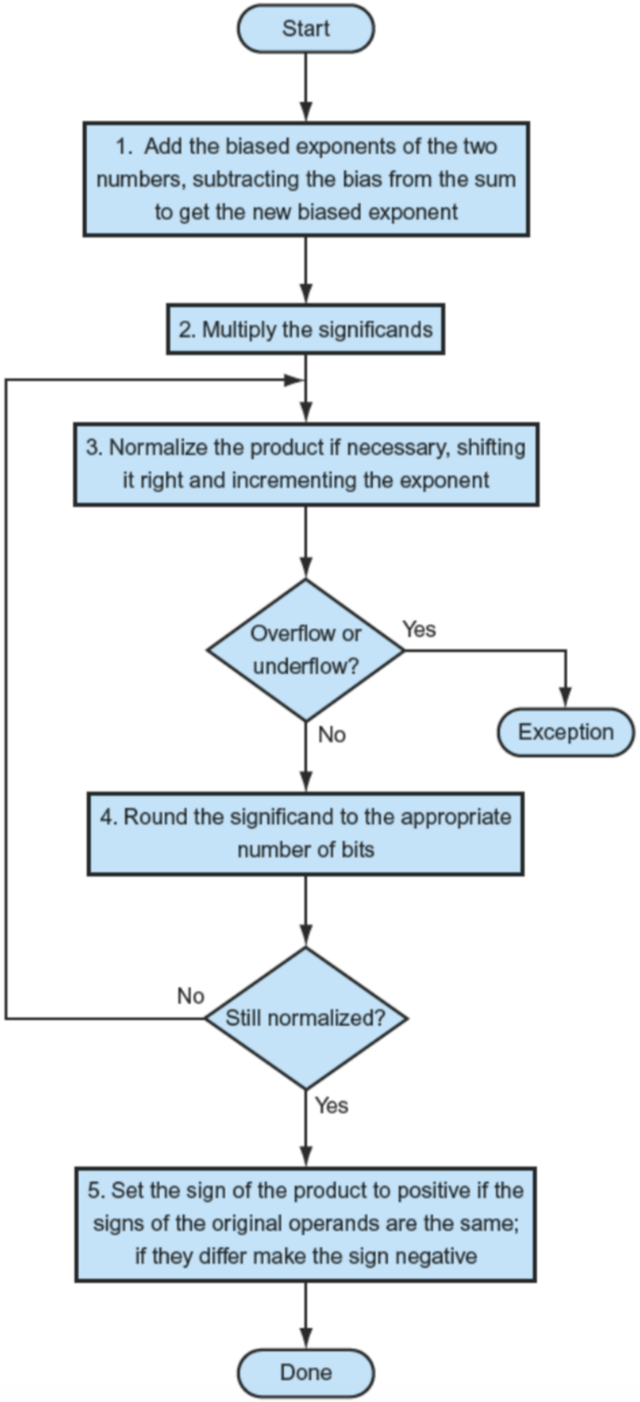
Exercise 2.19:

Add the following binary numbers as represented in a normalized single precision format.

0	1 1 0 0 1 0 0 0	1 1 1 1 0 1 0 1 0 0 ….. 0 0 0	+
0	1 0 0 1 1 0 1 0	0 1 0 1 0 0 1 1 0 0 ….. 0 0 0	

Multiplication

Flows



Patterson, D.A. and Hennessy, J.L. (2014). *Computer Organization and Design: The Hardware/Software Interface* (5th Edition). United States: Elsevier, p.209.

- Assume 4 decimal digits for fraction and 2 decimal digits for exponent.
  - **Step 1**: Add the *exponent* of the 2 numbers.
  - **Step 2**: Multiply the fraction.
  - **Step 3**: Normalize the product.
  - **Step 4**: Round the fraction.

    (If the fraction does not fit in the space reserved for it, it has to be rounded off)
  - **Step 5**: Normalize it (if need be) .
  - **Step 6**: Set the sign of the product.

Example 24:

Multiply these two decimal floating-point numbers.

Assume 4 decimal digits for *significand* and 2 decimal digits for *exponent*.

$$(1.110 \times 10^{10}) \times (9.200 \times 10^{-5}) = \_\_\_\_\__{10}$$

Solution:

11100000000000

0.000092

- **Step 1**: Add the exponent of the 2 numbers.

  10 + (–5) = 5

  If bias considered →

  5 + 127 = 132

- **Step 2**: Multiply the fraction.

```
      9.200
x     1.110
      0 000
     92 00
    920 0
   9200
   10212000
```

→ 10.212000

$= 10.2120 \times 10^{5}$

- **Step 3**: Normalize the product.

$$10.2120 \times 10^5 \times 10^1 \rightarrow 1.02120 \times 10^6$$

- **Step 4:** Round the fraction (to 4 decimal digits for fraction).

$$1.02120 \times 10^6 \rightarrow 1.0212 \times 10^6$$

- **Step 5**: Normalize it (if need be).

No need as its normalized

- **Step 6**: Set the sign of the product.

$$+1.0212 \times 10^6$$

2

Example 25:

Multiply these two binary floating-point numbers.

Assume 4 binary digits for significand and 2 binary digits for exponent.

$$(1.000 \times 2^{-1}) \times (-1.110 \times 2^{-2}) = \_\_\_\_\_\_{}_2$$

Solution:

- **Step 1**: Add the exponent of the 2 numbers.

$$(-1) + (-2) = -3$$

If bias considered →

$$(-3) + 127 = 124$$

- **Step 2**: Multiply the fraction.

```
    1.110
x   1.000
    0 000
   00 00
  000 0
 1110    .
1110000
```

→ 1.110000

$= 1.110000 \times 2^{-3}$

- **Step 3**: Normalize the product.

Already normalized

- **Step 4:** Round the fraction (to 4 decimal digits for fraction).

$1.110000 \times 2^{-3} \rightarrow 1.1100 \times 2^{-3}$

- **Step 5**: Normalize it (if need be).

No need as its normalized

- **Step 6**: Set the sign of the product.

-1.1100×2^{-3}

$$-111_2 \times 2^{-5} = -7_{10} \times \frac{1}{2^5}$$

$$= -(\frac{7}{32}) = -0.21975_{10}$$

Activity 13

Exercise 2.20:

Given two numbers 0.5_{10} and -0.4375_{10}.

(a) Multiply the numbers.

(b) Converting to decimal to check the results.

Show your workings.

2.6 Summary

- This module presented the essentials of data representation and numerical operations in digital computers.
- Student should master the techniques described for base conversion and memorize the smaller hexadecimal and binary numbers.
- This knowledge will be beneficial to student throughout remainder of this subject.
- Knowledge of hexadecimal coding will be useful if you are ever required to read a core (memory) dump after a system crash or if do any serious work in the field of data communications.

Linda Null and Julia Lobur (2003). *The Essentials of Computer Organization and Architecture.* United States: Jones and Bartlett Publishers. p.83.

Floating-points

Non-Standard
(*Example*: 14-bits)

Normalized:

$$0.12_{10} \times 10^3$$

$$0.1111_2 \times 2^7$$

1	5	8
S	*e'*	*Fraction*

$$\pm 0.Fraction \times Base^{e'}$$

Biased Notation:

$$\pm 0.Fraction \times Base^{E_b}$$

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

$b = 16$

$$E_b = e' + b$$

Standard (IEEE-754)

Single Precision (32-bits)

Double Precision (64-bits)

$$1.2_{10} \times 10^2$$

$$1.111_2 \times 2^6$$

Single Precision:

1	8	23
S	*e'*	*Fraction*

Double Precision:

1	11	52
S	*e'*	*Fraction*

$$(-1)^S \times (1 + Fraction) \times Base^{e'}$$

$$(-1)^S \times (1 + Fraction) \times Base^{E_B}$$

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

$B = 127$

$$E_B = e' + B$$

$B = 1023$

Review Questions

2.1 What are the three component parts of a floating-point number?

2.2 How many bits long is a double-precision number under the IEEE-754 floating-point standard?

2.3 Perform the following binary multiplications:

a) 1100×101

b) 10101×111

c) 11010×1100

2.4 Perform the following binary divisions:

a) $101101 \div 101$

b) $10000001 \div 101$

c) $1001010010 \div 1011$

2

2.5 Express the following numbers in IEEE 32-bit floating-point format:

(a) – 8

(b) – 7

(c) – 2.5

(d) 384

(e) 1/16

(f) – 1/4

2.6 The following numbers use the IEEE 32-bit floating-point format. What is the equivalent decimal value?

(a) 1 10000000 11000000000000000000000

(b) 0 01111111 00000000000000000000000

(c) 0 10000011 10100000000000000000000

2

2.7 Consider a floating-point format with 8 bits for the biased exponent and 23 bits for the significand. Show the bit pattern for the following numbers in this format:

(a) – 720 (b) 0.645

2.8 Show how the following floating-point calculations are performed (where significands are truncated to 4 decimal digits). Show the results in normalized form.

(a) $7.286 \times 10^{2} + 7.847 \times 10^{2}$

(b) $3.314 \times 10^{1} + 8.227 \times 10^{-2}$

(c) $(8.954 \times 10^{1}) \times (1.324 \times 10^{0})$

2.9. Assume we are using a floating-point representation uses a 14-bit format, 5 bits for the exponent with a bias of 16, a normalized mantissa of 8 bits, and a single sign bit for the number):

(a) Show how the computer would represent the numbers 100.0 and 0.25 using this floating-point format.

(b) Show how the computer would add the two floating-point numbers in part (a) by changing one of the numbers so they are both expressed using the same power of 2.

(c) Show how the computer would represent the sum in part (b) using the given floating-point representation. What decimal value for the sum is the computer actually storing? Explain.