AACS3064
Computer Systems Architecture

Chapter 2: Representing Numerical Data

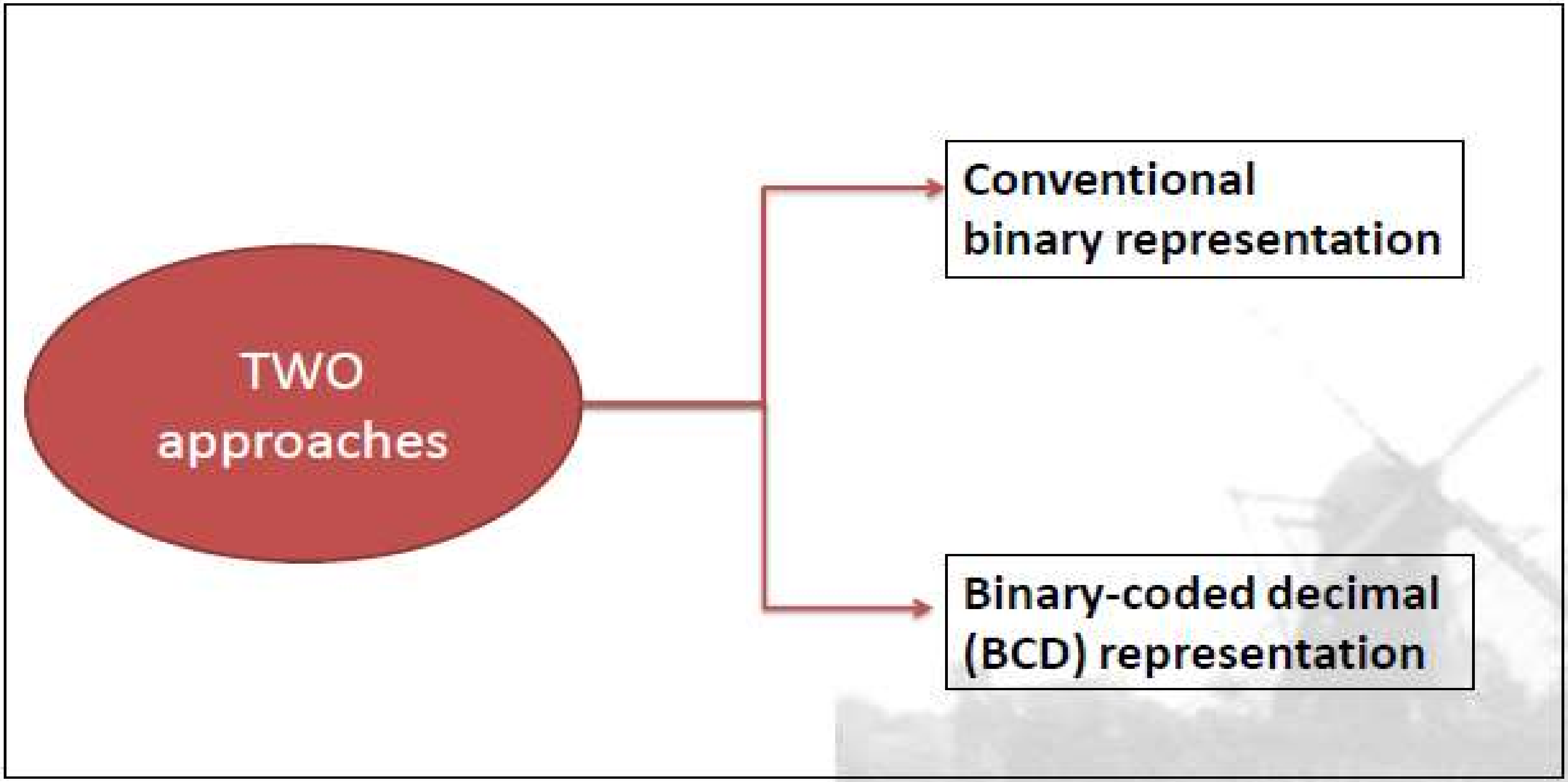
Chapter Overview

1) Unsigned Binary & BCD Representation
2) Signed Integers Representation
   - Sign-and-magnitude Representation
   - 1's Binary Complementary Representation
   - 2's Complement
   - Overflow and Carry Conditions
3) Floating point number
   - Exponential Notation
   - Floating Point Format
   - Normalizing and Formatting
   - Floating Point Calculations

1 Unsigned Binary & BCD Representation

1. Unsigned Binary & BCD

Unsigned Binary & BCD



1. Unsigned Binary & BCD (Continued)

Conventional binary representation

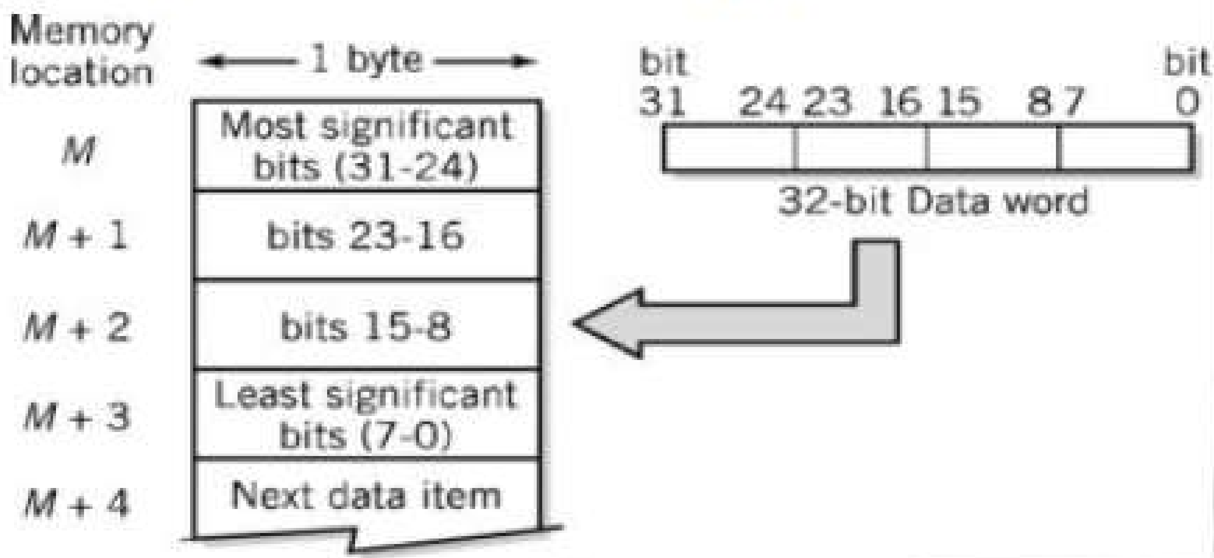
- Simply recognize that there is a direct binary equivalent for any decimal integers.
- Store any whole number as its binary representation.
- The range of integers that can store is determined by the number of bits available.
- Use multiple storage locations to expand the range of integers to be handled.

1. Unsigned Binary & BCD (Continued)

Conventional binary representation

Example : Multiple storage locations

4 consecutive 1-byte storage locations are used to provide 32 bits of range.



Disadvantages :

Increase the difficulty of calculation & manipulation.

1. Unsigned Binary & BCD (Continued)

Binary-coded decimal (BCD)

- The number is stored as a digit-by-digit binary representation of the original decimal integer.
- Each decimal digit is individually converted to binary. 4 bits per digit.

Example : 68_{10} -----> $0110\ 1000_2$

$0110_2 = 6_{10}$

$1000_2 = 8_{10}$

1. Unsigned Binary & BCD (Continued)

Value Range : Binary VS. BCD

- ➢ BCD range of values < conventional binary representation

No. of Bits	BCD Range		Binary Range	
4	0-9	1 digit	0-15	1+ digit
8	0-99	2 digits	0-255	2+ digits
12	0-999	3 digits	0-4,095	3+ digits
16	0-9,999	4 digits	0-65,535	4+ digits

Disadvantages :

- Calculations are more difficult.
- Any product / sum of any BCD integers that > 9 must be reconverted to BCD each time to perform the carries from digit to digit.

1. Unsigned Binary & BCD (Continued)

Binary-coded decimal (BCD)

E.g.

				Convert partial Sums to BCD ↓
76	→	$0111\ 0110_{bcd}$		
X 7	→	0111_{bcd}		
42	→	101010_{bin}	→	$0100\ 0010_{bcd}$
49	→	110001_{bin}	→	$+\ 0100\ 1001_{bcd}$
432	→			0100 1101 0010
13		Convert 13 back to BCD	→	+ 0001 0011
532	→			0101 0011 0010
				= 532

1. Signed Integers Representation

2. Signed Integer Representation

Sign-and-Magnitude Representation

- Use leftmost bit for sign.
  - 1 indicates a negative.
  - 0 indicates a positive.

- <u>Example using 8 bits</u> :
  - Unsigned: 1111 1111 (+255)
  - Signed: 0111 1111 (+127)
    1111 1111 (-127)

- <u>Disadvantage:</u>

  Calculations are difficult.

4 + 2	4 - 2	2 - 4	12 - 4
6	2	- 2	8

2. Signed Integer Representation (Continued)

1's Complement

- <u>Complementary representation</u>
  - The sign of the number is a natural result of the method
  - Does not need to be handled separately.

- <u>Inversion</u> : change 1's to 0's and 0's to 1's
  - Numbers beginning with 0 are positive
  - Numbers beginning with 1 are negative
  - 2 values for zero

<u>E.g.</u>
0111 1111 (127)
1000 0000 (-127)

2. Signed Integer Representation (Continued)

1's Complement

- <u>One's Complement Representation :</u>

1000 0000	1111 1111	0000 0000	0111 1111
-127_{10}	-0_{10}	0_{10}	127_{10}

- <u>*E.g.*</u>
  - One's complement of 4

    0000 0000 0000 0100 (4) → 1111 1111 1111 1011 (-4)

  - One's complement of 165

    0000 0000 1010 0101 (165) → 1111 1111 0101 1010 (-165)

2. Signed Integer Representation (Continued)

1's Complement

- **Arithmetic Example :**

```
    0010 1101 =  45          0010 1101 =  45
+   0011 1010 =  58      +   1100 0101 = -58
-------------------      -------------------
    0110 0111 = 103          1111 0010 = -13
```

2. Signed Integer Representation (Continued)

1's Complement

➢ <u>Arithmetic Example :</u>

```
                      0110 1010 =  106
                  +   1111 1101 =  - 2
                 ---------------------
                    1 0110 0111
(end-around carry)           +1
                 ---------------------
                      0110 1000 = 104
```

2. Signed Integer Representation (Continued)

1's Complement

- **Arithmetic Example :**

```
  0110 1010 =  106
- 0101 1010 =   90
-------------------
```

```
  0110 1010 =  106
+ 1010 0101 =  -90
-------------------
 1 0000 1111
         +1      (end-around carry)
-------------------
  0001 0000 =   16
```

2. Signed Integer Representation (Continued)

1's Complement

➢ <u>Arithmetic Example : (Overflow)</u>

	0100 0000 =	64
+	0100 0001 =	65
	1000 0001 =	-126

The correct positive result, 129. exceeds the range for 8 bits.

2. Signed Integer Representation (Continued)

2's Complement

- Find the 1's complement and adding 1 to the result.

- Two's Complement Representation :

1000 0000	1111 1111	0000 0000	0111 1111
-128_{10}	-1_{10}	0_{10}	127_{10}

E.g.
0111 1111 (127)
1000 0001 (-127)

2. Signed Integer Representation (Continued)

2's Complement

- ***E.g.***
  - Two's complement of 4

    One's complement of 4 → 1111 1111 1111 1011

    \+ 1

    Two's complement of 4 → 1111 1111 1111 1100 (-4)

2. Signed Integer Representation (Continued)

1's Complement vs. 2's Complement

- 1's complement
  - Addition requires extra end-around carry
  - Algorithm must test for and convert -0
- 2's complement
  - Additional add operation required for sign change

2. Signed Integer Representation (Continued)

Perform subtraction using 2's Complement

Example **: 30 - 10**

STEP 1 : Change 10 to its binary format

10 → 0000 1010

STEP 2 : Apply 2's complement rule to obtain its negative representation. (Reserved bit and Add 1)

		0000 1010
Reversed		1111 0101
Add 1	+	1
		1111 0110

STEP 3 : Perform addition

	00011110 (30)
+	11110110 (-10)
	(1) 00010100 (20)

2. Signed Integer Representation (Continued)

Overflow Flag (OF)

- Occurs when the result of an arithmetic operation does not fit into the fixed number of bits available for the result.
- Occur only when both operands have the same sign.
- Detected by the fact that the sign of the result is opposite of both operands.
- **<u>E.g. :</u>** slide 17

2. Signed Integer Representation (Continued)

Carry Flag (CF)

- Occurs when the result of an arithmetic operation exceeds the fixed number of bits allocated, without regard to sign.
- For normal, single precision 2's complement addition and subtraction the carry bit is ignored.
- Detected when extra '1' bit generated.
- **E.g. :** slide 15, 16

2. Signed Integer Representation (Continued)

Example : Addition of 4-bit 2's complement

Example :

(+4) + (+2)

0100	No overflow,
0010	No carry
0110 (+6)	The result is correct

(-4) + (-2)

1100	No overflow
1110	Carry
11010 (-6)	The result is correct

2. Signed Integer Representation (Continued)

Example : Addition of 4-bit 2's complement

Example :

(+4) + (+6)	
0100	overflow,
0110	No carry
1010 (-6)	The result is incorrect
(-4) + (-6)	
1100	Overflow
1010	Carry
10110 (+6)	Ignoring the carry, The result is incorrect

3. Floating Point Number

3. Floating Point numbers

Floating Point Numbers

- Real number

- Used in the computer when the number :
  - Is outside the integer range of the computer (too large or too small)
  - Contains a decimal fraction

3. Floating Point numbers (Continued)

Exponential Notation

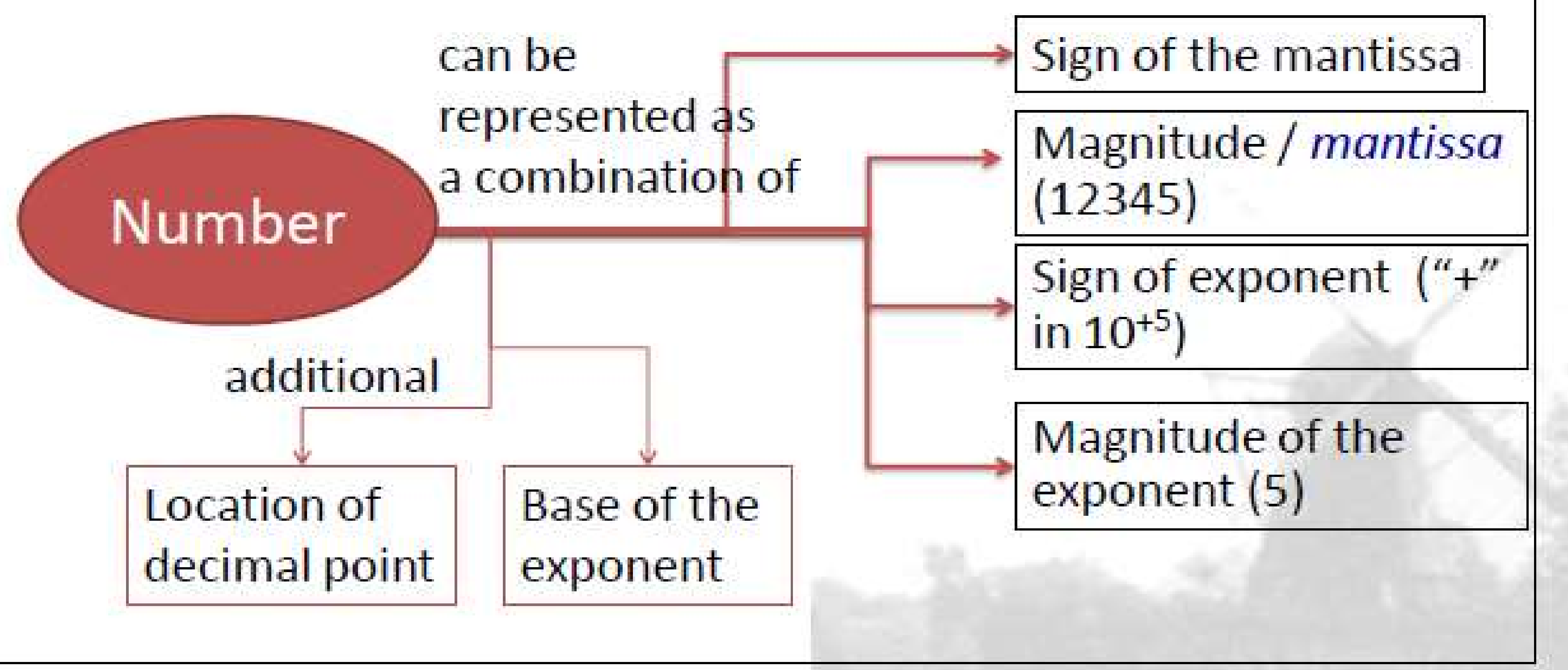
- Also called ***scientific notation***

12345	12345×10^{0}
0.12345×10^{5}	123450000×10^{-4}
0.0012345×10^{7}	

3. Floating Point numbers (Continued)

Exponential Notation

- 4 specifications required for a number



3. Floating Point numbers (Continued)

Floating Point Representation

Possible representation of number -0.0000003579 is :

Sign of the mantissa

Sign of the exponent

$$-\,0.35790 \times 10^{-6}$$

Location of decimal point

Mantissa

Base

Exponent

3. Floating Point numbers (Continued)

Format Specification

- Floating point numbers will be stored and manipulated in the computer using a "standard", predefined format, usually in 8 bits

Sign of the mantissa → **SEEMMMMM**

2-digit Exponent

5 digits of mantissa

- ***NOTE :*** the base of the exponent and the location of the binary point are standardize as part of the format, so do not have to stored at all.

3. Floating Point numbers (Continued)

Format

- Mantissa: sign digit in sign-magnitude format
  - Assume decimal point located at beginning of mantissa
- ***Excess-N notation***:
  - N is the chosen middle value
  - Example : Excess-50 representation

Representation	0	49	50	99
Exponent being represented	-50	-1	0	49

– Increasing value +

Allow a magnitude range of

$0.00001 \times 10^{-50} < \text{Number} < 0.99999 \times 10^{49}$

3. Floating Point numbers (Continued)

Conversion Examples

- **Format used :**
  - SEEMMMMM, Excess-50
  - 0 represent '+' sign, and 5 represent '-' sign
  - The base is 10.
  - The implied decimal point is at the beginning of the mantissa.

05324567	=	0.24567×10^{3}	=	246.57
54810000	=	-0.10000×10^{-2}	=	-0.0010000
5555555	=	-0.55555×10^{5}	=	-55555
04925000	=	0.25000×10^{-1}	=	0.025000

3. Floating Point numbers (Continued)

Normalization & Formatting

- <u>Normalization</u>
  Shift numbers left by increasing the exponent until leading zeros are eliminated.

  $.MMMMM \times 10^{EE}$

- <u>Steps</u> to converting decimal number into standard format.

  1. Provide number with exponent (0 if not yet specified).
  2. Increase/decrease exponent to shift decimal point to proper position.
  3. Decrease exponent to eliminate leading zeros on mantissa.
  4. Correct precision by adding 0's or discarding / rounding least significant digits

3. Floating Point numbers (Continued)

Normalization & Formatting

Example 1 : (Decimal to floating point format conversion)

Given 246.8035, normalized it and represent it in SEEMMMMM format.

i.	Add exponent	246.8035×10^{0}
ii.	Position decimal point	0.2468035×10^{3}
iii.	Already normalized, no adjustment required.	
iv.	Cut to 5 digits	0.24680×10^{3}
v.	Convert number	05324680

3. Floating Point numbers (Continued)

Normalization & Formatting

Example 2 :

1255×10^{-3}

i. The number is already in exponential form.
ii. Position decimal point 0.1255×10^{1}
iii. Already Normalized
iv. 5 digits of precision 0.12550×10^{1}
v. Convert number 05112550

3. Floating Point numbers (Continued)

Normalization & Formatting

Example 3:

-0.00000075

i.	Add exponent	$-0.00000075 \times 10^{0}$
ii.	Position decimal point	-0.75×10^{-6}
iii.	Already Normalized	
iv.	5 digits of precision	-0.75000×10^{-6}
v.	Convert number	54475000

3. Floating Point numbers (Continued)

Floating Point Calculation : Add & Subtract

- Exponent and mantissa treated separately.
- Exponents of numbers must agree.
  - Align decimal points.
  - Least significant digits may be lost.
- Overflow of the most significant digit may occurs.
  - Number must be shifted right and the exponent incremented to accommodate overflow.

3. Floating Point numbers (Continued)

Floating Point Calculation : Add & Subtract

Example : Add the 2 floating-point numbers (Assume these numbers are formatted using SEEMMMMM and excess-50)

Add 2 floating point numbers	05199520 + 04967850
Align exponents	05199520
By adding two zeros in front of mantissa	0510067850
Add mantissa , (1) Indicates a carry	(1) 0019850
Carry requires right shift of exponent	05210019 (850)
Round	05210020

3. Floating Point numbers (Continued)

Floating Point Calculation : Add & Subtract

Check Results :

$05199520 = 0.99520 \times 10^{1} =$ $\quad$ $9 . 9520$

$04967850 = 0.67850 \times 10^{-1} =$ $\quad$ $0 . 067850$

$\quad$ $10 . 019850$

In sign-magnitude form $\quad$ 0.1001985×10^{2}

3. Floating Point numbers (Continued)

Floating Point Calculation : Multiply & divide

- Mantissas: multiplied or divided.
- Exponents: added or subtracted and adjusted excess value since added twice.
- Example :
  - Assume we have two numbers with exponent 3, Each is represented in excess-50 notation as 53.
  - Adding the two exponents g 53 + 53 = 106.
  - Since 50 added twice, subtract: 106 – 50 = 56.
- Normalization necessary to:
  - Restore location of decimal point.
  - Maintain precision of the result.

3. Floating Point numbers (Continued)

Floating Point Calculation : Multiply & divide

Example : Multiply the 2 floating point numbers

Multiply 2 floating point numbers	05220000 X 04712500
Add exponents, subtract offset	52 + 47 – 50 = 49
Multiply mantissa	0.20000 x 0.12500 = 0.025000000
Normalized the result	04825000

3. Floating Point numbers (Continued)

Floating Point Calculation : Multiply & divide

Check results :

$05220000 = 0.20000 \times 10^{2} =$

$04712500 = 0.125 \times 10^{-3} =$

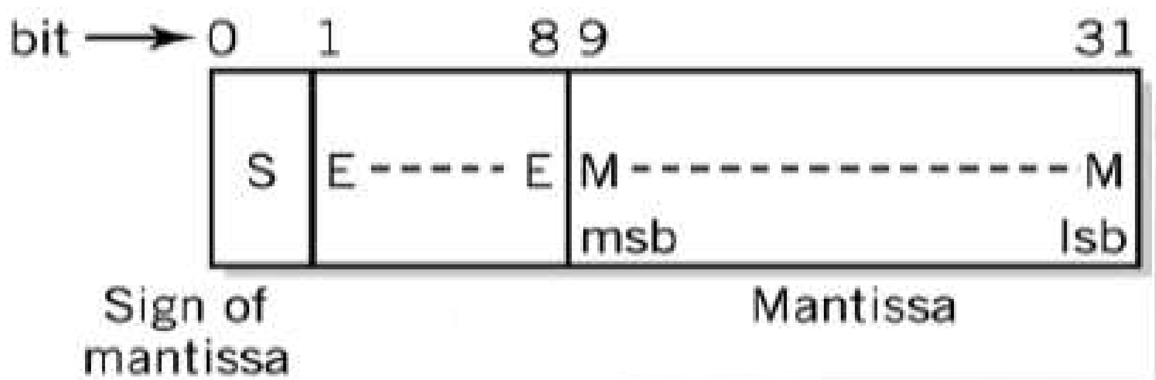
Multiply $0.20000 \times 10^{2} \times 0.125 \times 10^{-3} =$ $\quad 0.025000000 \times 10^{-1}$

Normalizing and rounding $\quad 0.25000 \times 10^{-2}$

3. Floating Point numbers (Continued)

Floating Point in the Computer

- Typical floating point format :
  - consists of 32 bits, divided into :
    - A sign bit
    - 8 bits of exponent, excess-127 notation, base 2
    - 23 bits of mantissas.



3. Floating Point numbers (Continued)

IEEE 754 Standard

- Normalized numbers must always start with a 1, the leading bit is not stored, but is instead implied.
- This bit is located to the left of the implied binary point. So, numbers are normalized to the form 1.MMMMMM.....

Precision	Single (32-bit)	Double (64-bit)
Sign	1 bit	1 bit
Exponent	8 bits	11 bits
Notation	Excess-127	Excess-1023
Implied Base	2	2
Range	2^{-126} to 2^{127}	2^{-1022} to 2^{1023}
Mantissa	23	52

3. Floating Point numbers (Continued)

IEEE 754 Standard

Example :

Convert this decimal number 253.75 to binary floating-point form.

253 .	75
1111 1101 .	11

253.75 → 1111 1101.11

→ 1.111 110111 x 2^{+7}

Sign bit ←-----------------------Mantissa (23 bits)---------------------→

0	1	0	0	0	0	1	1	0	1	1	1	1	1	0	1	1	1	0	0	..	..	0	0

Exponent **8 bits**

←-------- Excess-127----------→

3. Floating Point numbers (Continued)

IEEE 754 Standard

Solution:

Convert 253.75 to binary	11111101.11
Add exponent	11111101.11 x 2^0
Position Decimal Point	1.111110111 x 2^{+7}
Exponent	127 + 7 = 134
Change 134 to binary	10000110
Sign of Mantissa (positive)	0
Mantissa in 23 bits (the leading bit is not stored, but is instead implied)	11111011 10000000 0000000
IEEE 754 format	0 10000110 11111011 10000000 0000000

3. Floating Point numbers (Continued)

Conversion of fractional number

Example 1 : base 10 -> base 2

i. 16.5_{10}

2^2	2^1	2^0	2^{-1}	2^{-2}	2^{-3}
4	2	1	½	¼	1/8

The answer is :

$$
\begin{aligned}
16.5_{10} &= 16_{10} + 0.5_{10} \\
&= 10000_2 + 0.1_2 \\
&= 10000.1_2
\end{aligned}
$$

3. Floating Point numbers (Continued)

Conversion of fractional number

Example 1 : base 10 -> base 2

i. 0.828125_{10}

$$
\begin{array}{lr}
 & 0.828125 \\
\times & 2 \\
\hline
 & 1.656250 \\
\times & 2 \\
\hline
 & 1.312500 \\
\times & 2 \\
\hline
 & 0.625000 \\
\times & 2 \\
\hline
 & 1.250000 \\
\times & 2 \\
\hline
 & 0.500000 \\
\times & 2 \\
\hline
 & 1.000000
\end{array}
$$

x 2 because it needs to change to base 2.

Reading the answer from top-bottom before the .(Dot/Decimal Point) = 110101
But, since it is multiplied using fractional numbers, must add 0. in front of the answer
= 0.110101

3. Floating Point numbers (Continued)

Conversion of fractional number

Example 1 : base 10 -> base 2

i. **0.828125_{10}**

*** To validate it!

2^0	2^{-1}	2^{-2}	2^{-3}	2^{-4}	2^{-5}	2^{-6}
0	**0.5**	**0.25**	**0**	**0.0625**	**0**	**0.015625**
0	**1**	**1**	**0**	**1**	**0**	**1**

→ 0.5 + 0.25 + 0.0625 + 0.15625
= 0.828125_{10}

3. Floating Point numbers (Continued)

Conversion of fractional number

Example 2 : base 2 -> base 10

i. $11.110011_2 = 3.796875_{10}$

1	1	.	1	1	0	0	1	1
2^1	2^0	.	2^{-1}	2^{-2}	2^{-3}	2^{-4}	2^{-5}	2^{-6}

The answer is :

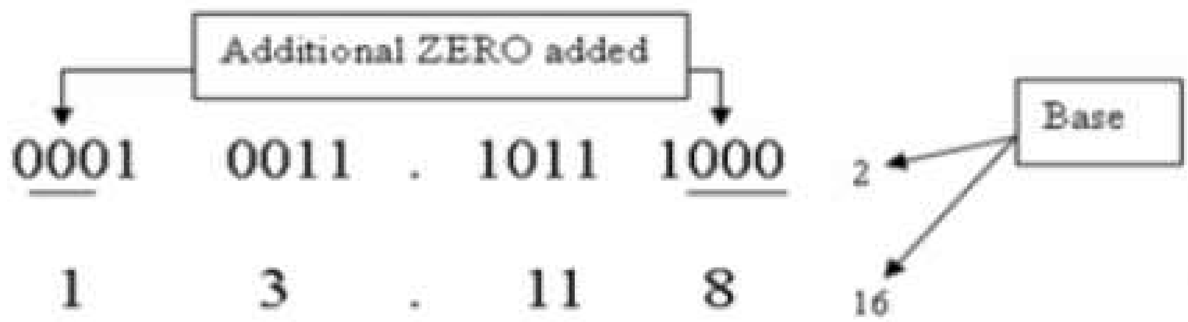
$= (1 \times 2^1) + (1 \times 2^0) + (.) + (1 \times 2^{-1}) + (1 \times 2^{-2}) + (1 \times {}^{2\text{-}5}) +$
(1×2^{-6})
$= 2 + 1 + (.) + 0.5 + 0.25 + 0.03125 + 0.015625$
$= 3.796875\text{d}$

3. Floating Point numbers (Continued)

Conversion of fractional number

Example 3 : base 2 -> base 16

$10011.10111_2 = 13.18_{16}$



3. Floating Point numbers (Continued)

Conversion of fractional number

Example 4 : base 16 -> base 10

39.B8H = 57.71875D

3	9	.	B	8	16
$\times 16^{1}$	$\times 16^{0}$	.	$\times 16^{-1}$	$\times 16^{-2}$	

The answer is :

$$= (3 \times 16^{1}) + (9 \times 16^{0}) + (.) + (11 \times 16^{-1}) + (8 \times 16^{-2})$$
$$= 48 + 9 + (.) + 0.6875 + 0.03125$$
$$= 57.71875_{10}$$

3. Floating Point numbers (Continued)

Conversion of fractional number

Example 5 : base 16 -> base 2

4F5.09H = 010011110101.00001001B

4	F	5	.	0	9
0100	1111	0101	.	0000	1001

Chapter Review

1) Unsigned Binary & BCD Representation
2) Signed Integers Representation
   - Sign-and-magnitude Representation
   - 1's Binary Complementary Representation
   - 2's Complement
   - Overflow and Carry Conditions
3) Floating point number
   - Exponential Notation
   - Floating Point Format
   - Normalizing and Formatting
   - Floating Point Calculations