Binary System

--Basics

The AND operation

- 0 AND 0 = 0
- 0 AND 1 = 0
- 1 AND 0 = 0
- 1 AND 1 = 1

The OR operation

- 0 OR 0 = 0
- 0 OR 1 = 1
- 1 OR 0 = 1
- 1 OR 1 = 1

exclusive OR

The XOR operation

- 0 XOR 0 = 0
- 0 XOR 1 = 1
- 1 XOR 0 = 1
- 1 XOR 1 = 0

The NOT operation

- NOT 0 = 1
- NOT 1 = 0

$$\begin{array}{r} 0 \\ +\,0 \\ \hline 00 \end{array} \qquad \begin{array}{r} 0 \\ +\,1 \\ \hline 01 \end{array} \qquad \begin{array}{r} 1 \\ +\,0 \\ \hline 01 \end{array} \qquad \begin{array}{r} 1 \\ +\,1 \\ \hline 10 \end{array} \qquad \begin{array}{r} A \\ +\,B \\ \hline \end{array}$$

A and B

A xor B

Summary

NOT

A	B	AND	OR	XOR	NAND	NOR
0	0	0	0	0	1	1
0	1	0	1	1	1	0
1	0	0	1	1	1	0
1	1	1	1	0	0	0

The hexadecimal coding system

$a_3 a_2 a_1 a_0 = a_3 \times 2^3 + a_2 \times 2^2 + a_1 \times 2^1 + a_0$

• 0000	0	1000	8	
• 0001	1	1001	9	
• 0010	2	1010	10	A
• 0011	3	1011	11	B
• 0100	4	1100	12	C
• 0101	5	1101	13	D
• 0110	6	1110	14	E
• 0111	7	1111	15	F

Figure 1.7: The organization of a byte-size memory cell

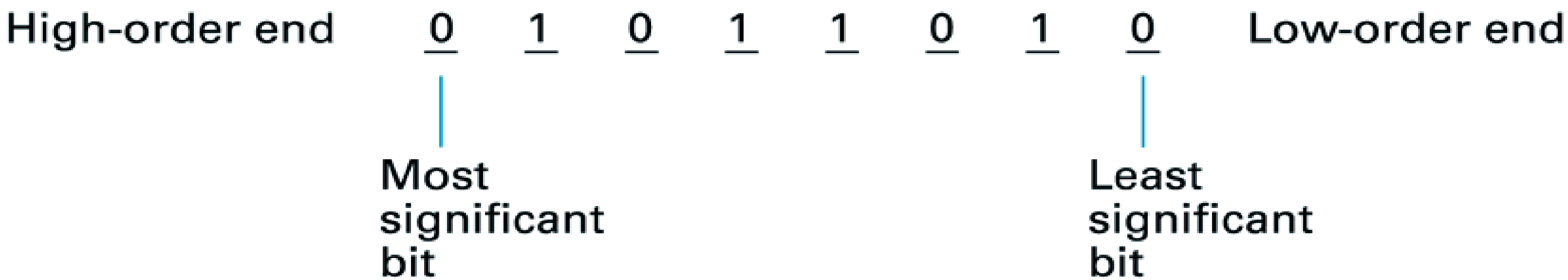


Figure 1.14: The base ten and binary systems

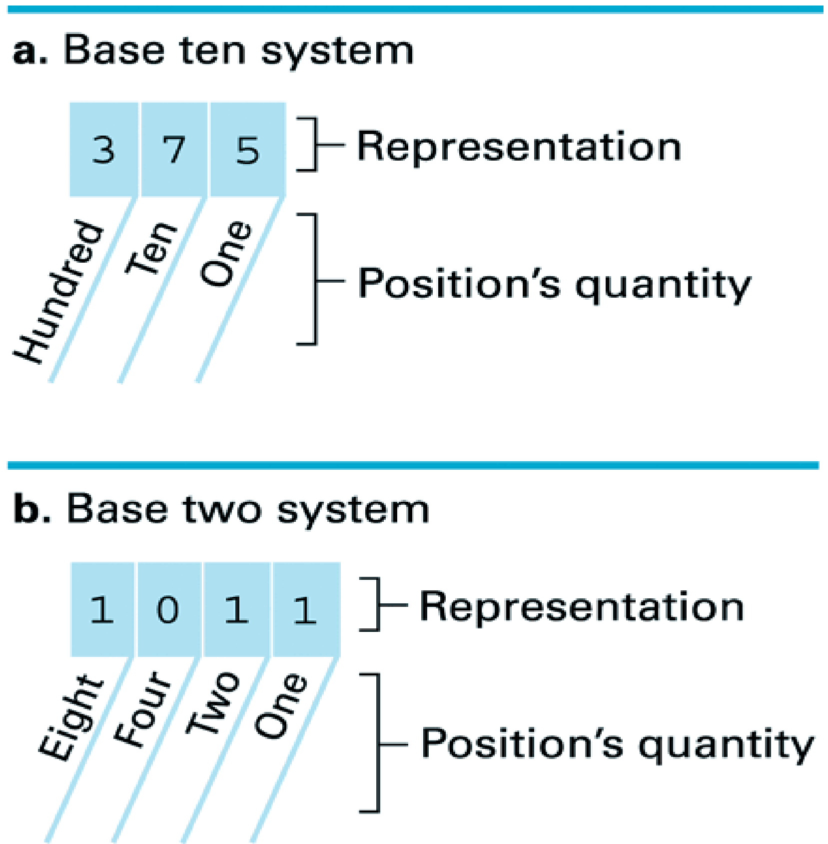
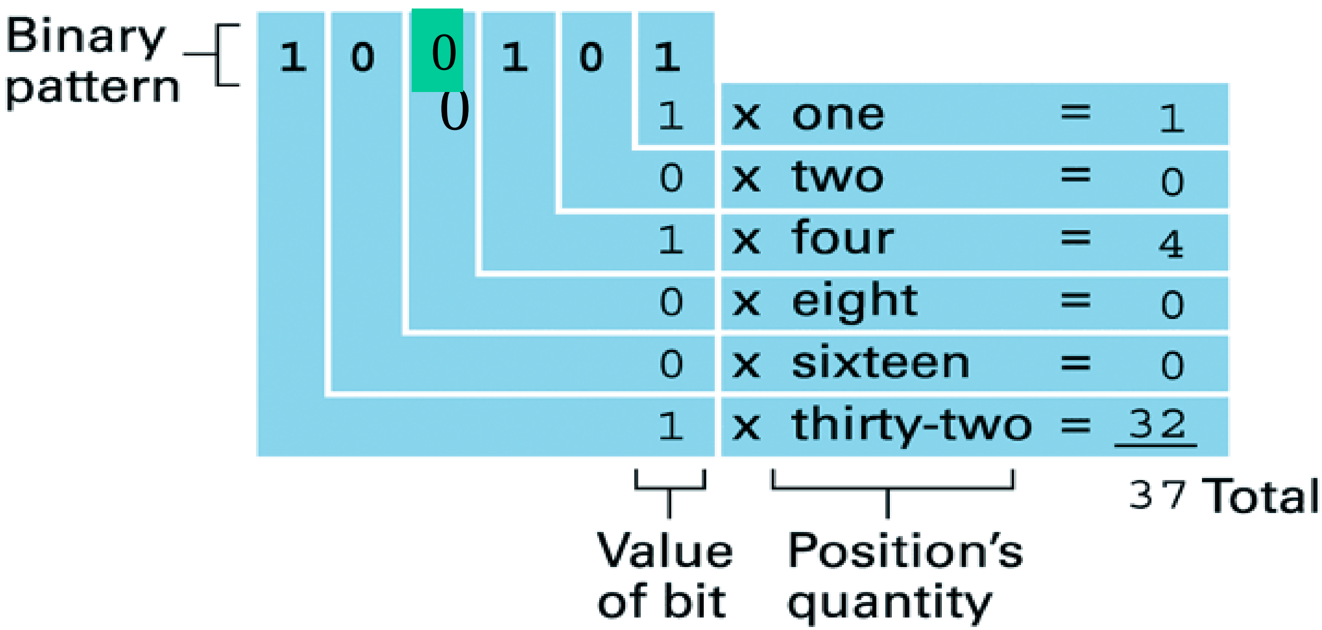


Figure 1.15: Decoding the binary representation 100101



An algorithm for finding the binary representation of a positive integer

- Step 1: Divide the value by two and record the remainder
- Step 2: As long as the quotient obtained is not zero, continue to divide the newest quotient by two and record the remainder.
- Step 3: Now that a quotient of zero has been obtained, the binary representation of the original value consists of the remainders listed from right to left in the order they were recorded.

Figure 1.17: Applying the algorithm in Figure 1.15 to obtain the binary representation of thirteen

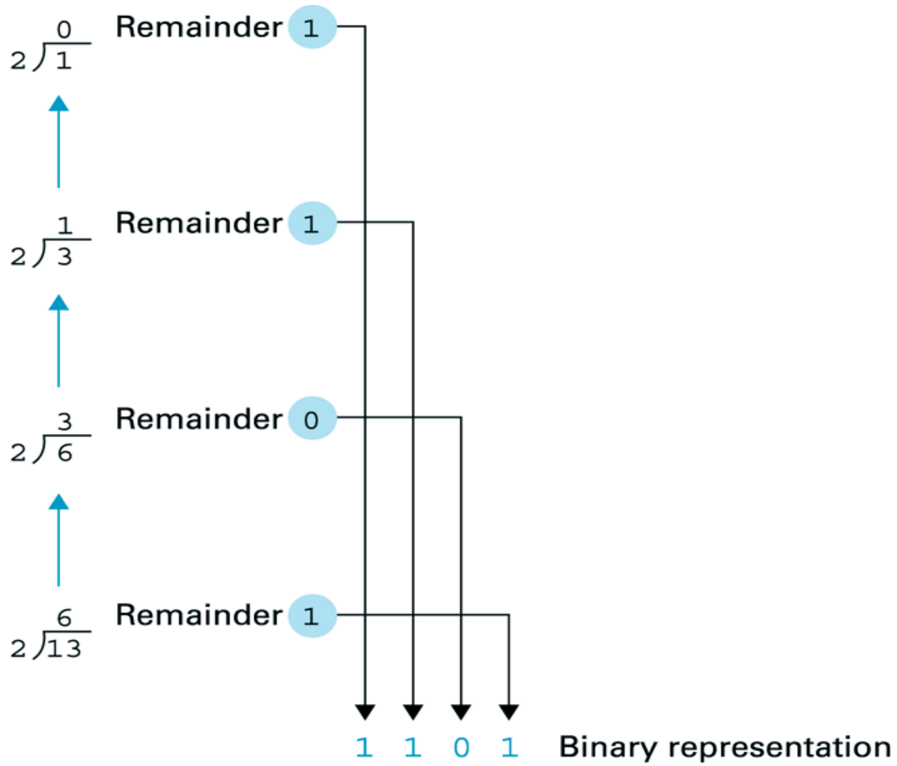


Figure 1.19: The binary addition facts

$$\begin{array}{r} 6 \\ +\ 8 \\ \hline 14 \end{array}$$

$$\begin{array}{r} 0 \\ +\ 0 \\ \hline 0 \end{array} \qquad \begin{array}{r} 1 \\ +\ 0 \\ \hline 1 \end{array} \qquad \begin{array}{r} 0 \\ +\ 1 \\ \hline 1 \end{array} \qquad \begin{array}{r} 1 \\ +\ 1 \\ \hline 10 \end{array}$$

Complement

- $67 - 55 = 67 - (100 - 45)$
- $= 67 + 45 - 100$
- $= 12$

Figure 1.21: Two's complement notation systems

a. Using patterns of length three

Bit pattern	Value represented
011	3
010	2
001	1
000	0
111	−1
110	−2
101	−3
100	−4

b. Using patterns of length four

Bit pattern	Value represented
0111	7
0110	6
0101	5
0100	4
0011	3
0010	2
0001	1
0000	0
1111	−1
1110	−2
1101	−3
1100	−4
1011	−5
1010	−6
1001	−7
1000	−8

Figure 1.22: Coding the value -6 in two's complement notation using four bits

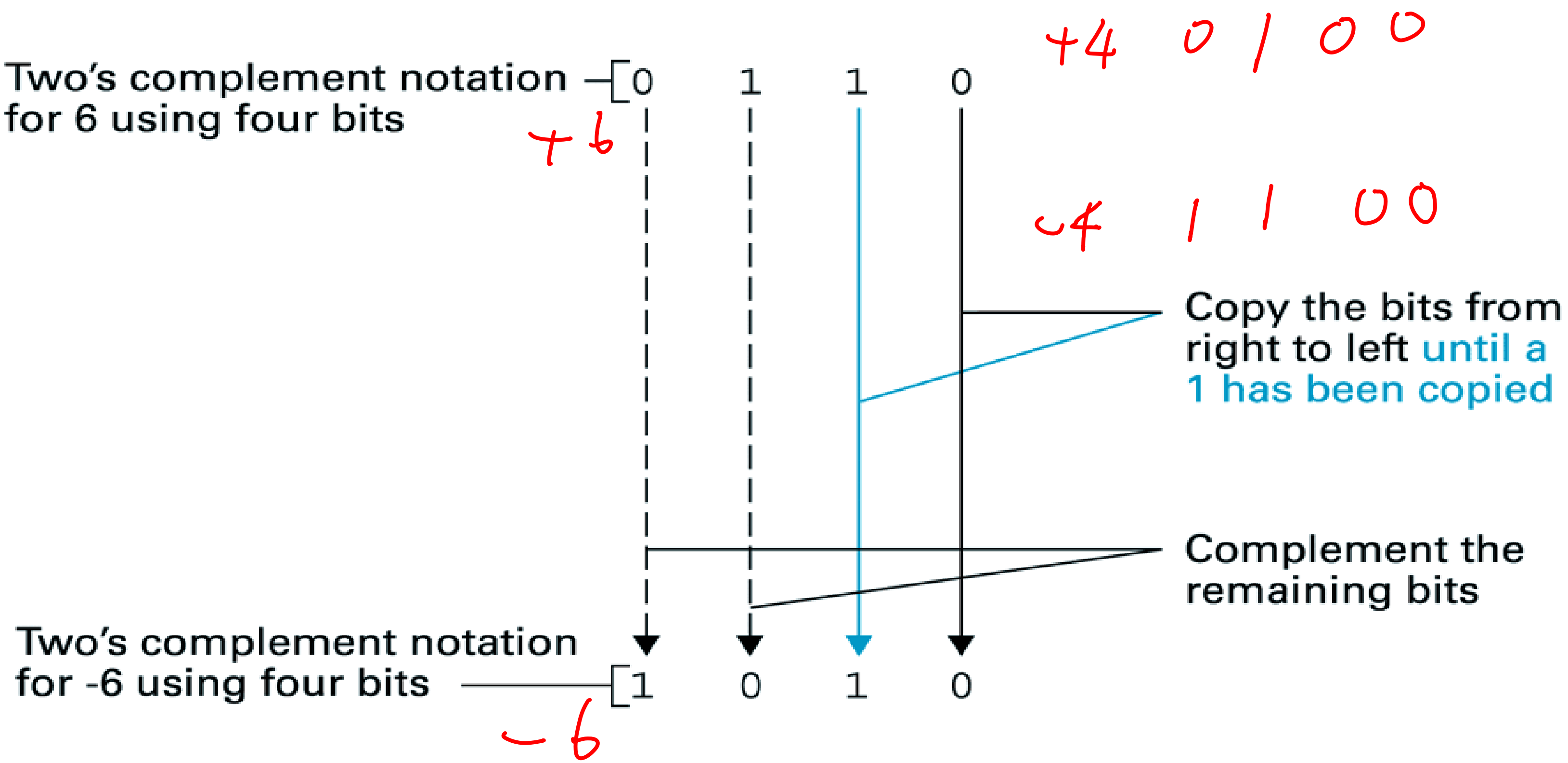


Figure 1.23: Addition problems converted to two's complement notation

Problem in base ten		Problem in two's complement		Answer in base ten
3 + 2	→	0011 + 0010 = 0101	→	5
−3 + −2	→	1101 + 1110 = 1011	→	−5
7 + −5	→	0111 + 1011 = 0010	→	2

6 0110
+7 0111
(−3) 1101

Overflow? E.g. 6+7= -3 or -6 - 8 = (-6)+(-8)=+2

Machine can make mistakes. It is treated with a special procedure.

skip

Figure 1.24: An excess eight conversion table

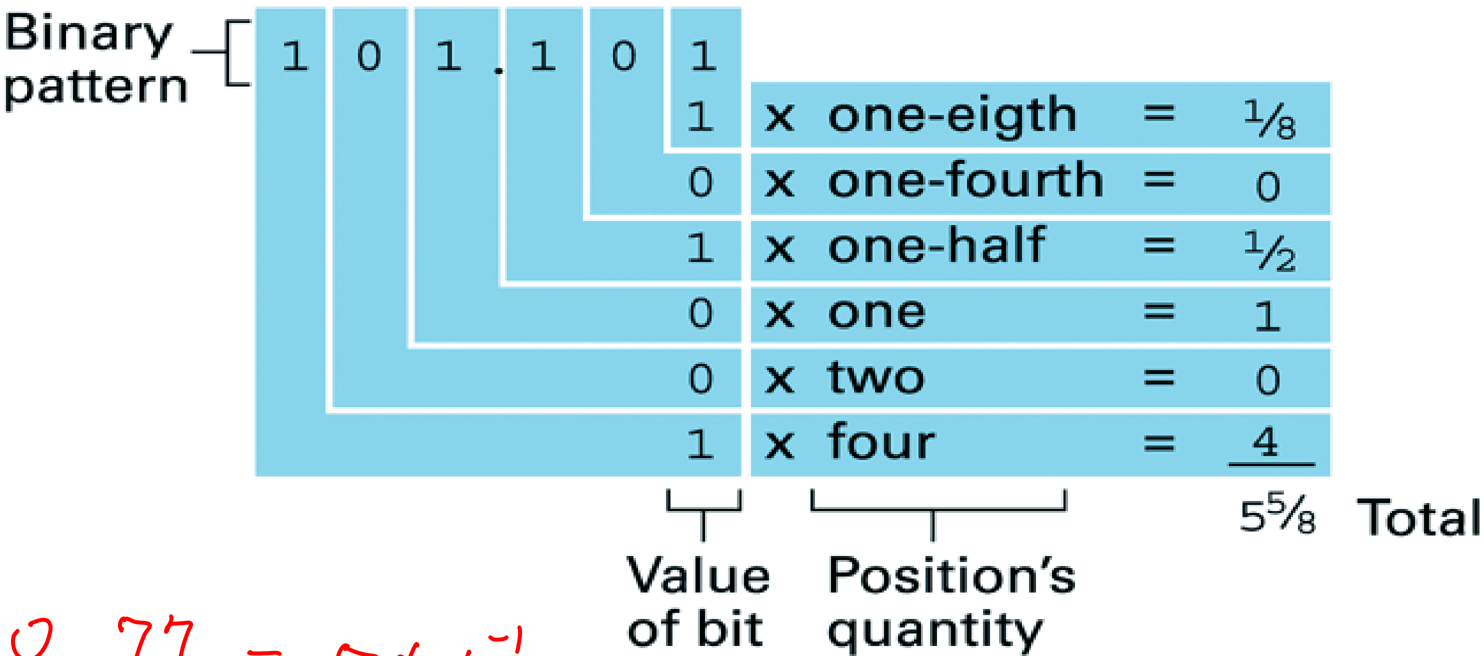
Bit pattern	Value represented
1111	7
1110	6
1101	5
1100	4
1011	3
1010	2
1001	1
1000	0
0111	−1
0110	−2
0101	−3
0100	−4
0011	−5
0010	−6
0001	−7
0000	−8

skip

Figure 1.25: An excess notation system using bit patterns of length three

Bit pattern	Value represented
111	3
110	2
101	1
100	0
011	−1
010	−2
001	−3
000	−4

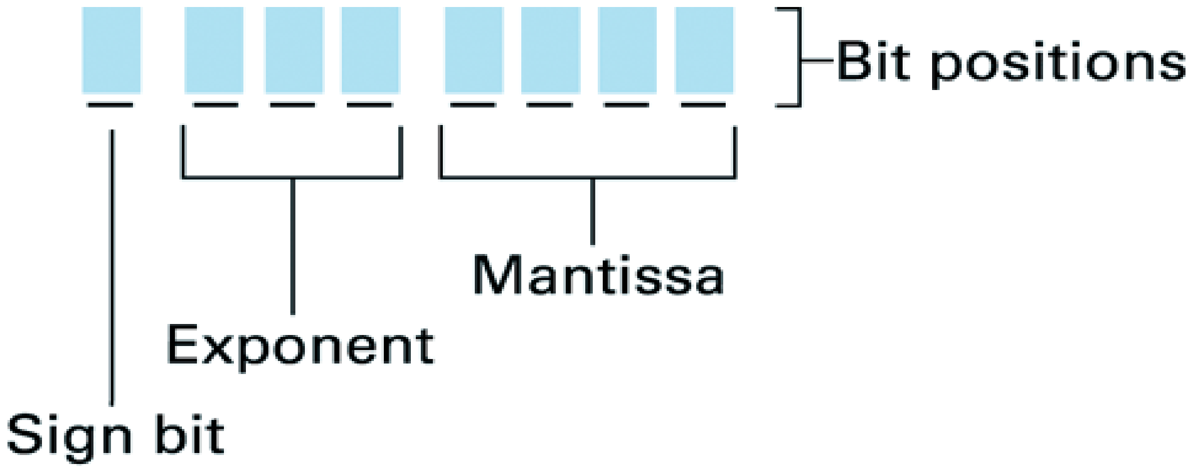
Figure 1.20: Decoding the binary representation 101.101



$0.77 = 7 \times 10^{-1} + 7 \times 10^{-2}$

Figure 1.26: Floating-point notation components

1×10^{6} 1×10^{-6}

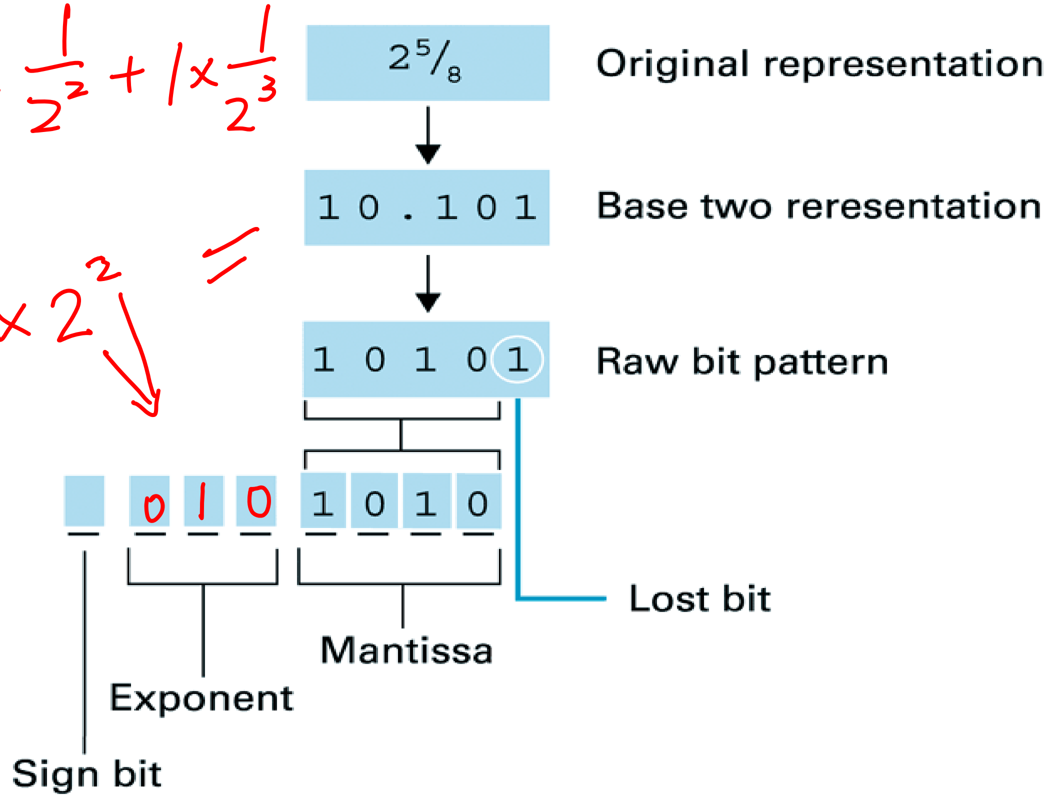


$1768231.1097 = 0.176823\cancel{1099} \times 10^7$

Figure 1.27: Coding the value 2 5/8

$\frac{5}{8} = 1 \times \frac{1}{2} + 0 \times \frac{1}{2^2} + 1 \times \frac{1}{2^3}$

0.10101×2^2 =



+ − 0

Converting 0.3 (Decimal) to binary =?

Truncation (round off) error