CMPS1134
Fundamentals of Computing

Data Storage 2

Computer Science: An Overview
Eleventh Edition
J. Glenn Brookshear
Chapter 1

Chapter 1: Data Storage 2

- The Binary System
- Storing Integers
- Storing Fractions
- Data Compression
- Communications Errors

The Binary System

- The traditional decimal system is based on powers of ten.
- The Binary system is based on powers of two.

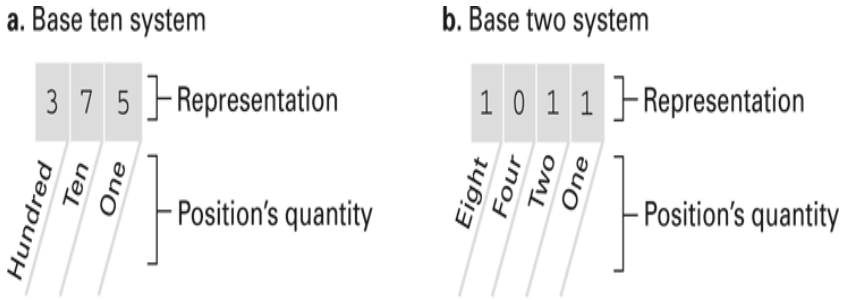
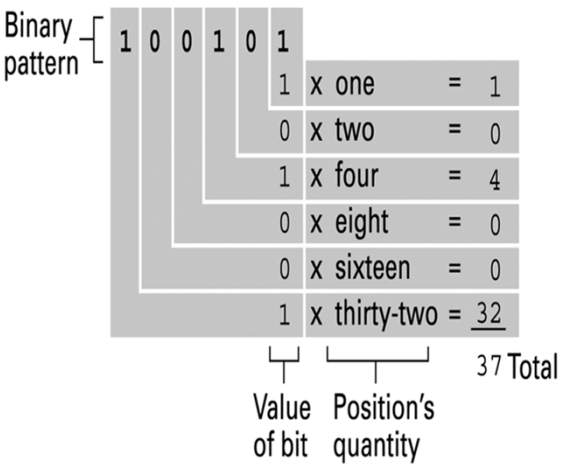
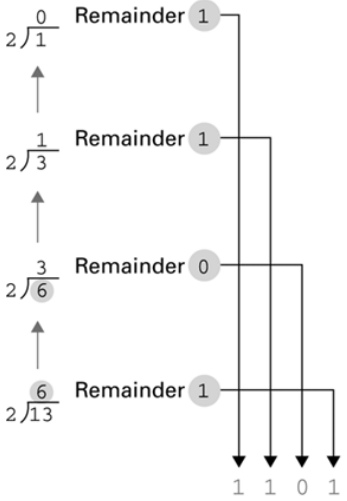


Figure 1.16 **Decoding the binary representation 100101**



1. 100110
2. 1101001
3. 1001111

Figure 1.17 **Algorithm to find 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.

Decoding the binary representation Exercise

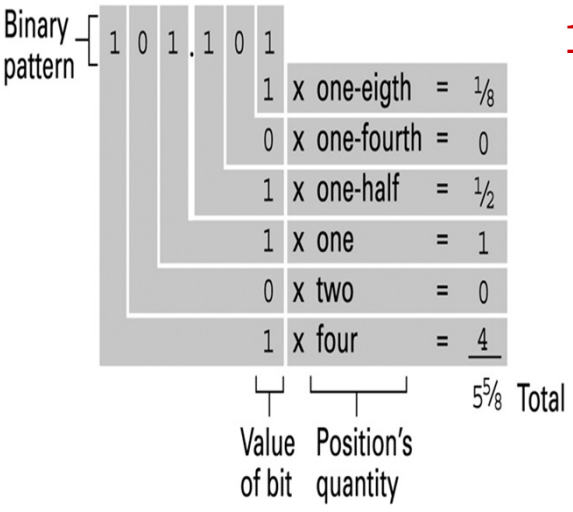
39 53

Binary Addition

```
  0      1      0      1
+ 0    + 0    + 1    + 1
  0      1      1     10
```

```
   10100          111010
+  11011       +  11011
```


Figure 1.20 **Decoding the binary representation 101.101**



1. 10101.10
2. 1010.1101

Storing Integers

- **Two's complement notation:** The most popular means of representing integer values
  - The use of a pattern of 32 bits is common in today's equipment
- **Excess notation:** Another means of representing integer values
- Both can suffer from overflow errors.
  - If a value exceeds the minimum or maximum represented by the notation

Figure 1.21 **Two's complement notation systems**

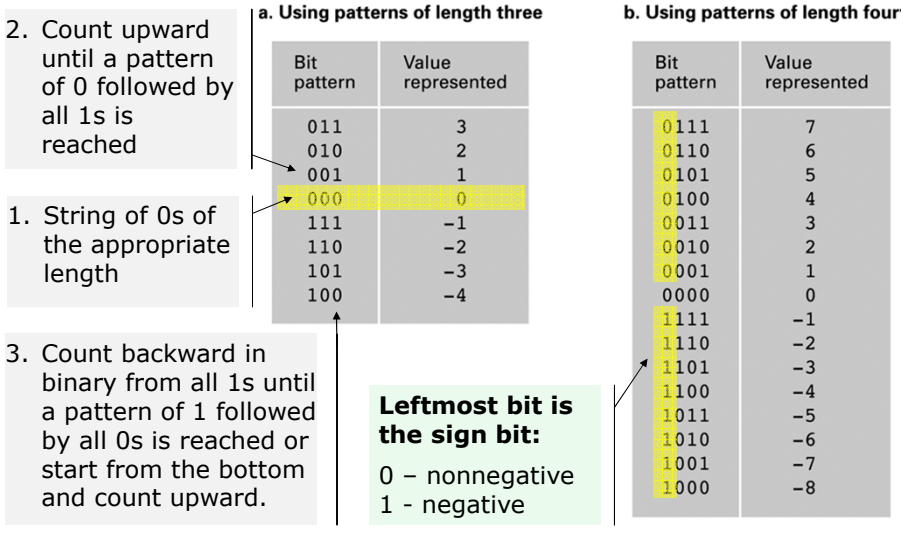


Figure 1.22 **Coding the value -6 in two's complement notation using four bits**

Nonnegative and negative numbers of the same magnitude are identical when read right to left, up to and including the first 1, and then the patterns are **complements** of each other. i.e. 0s and 1s in one are 1s and 0s in the other

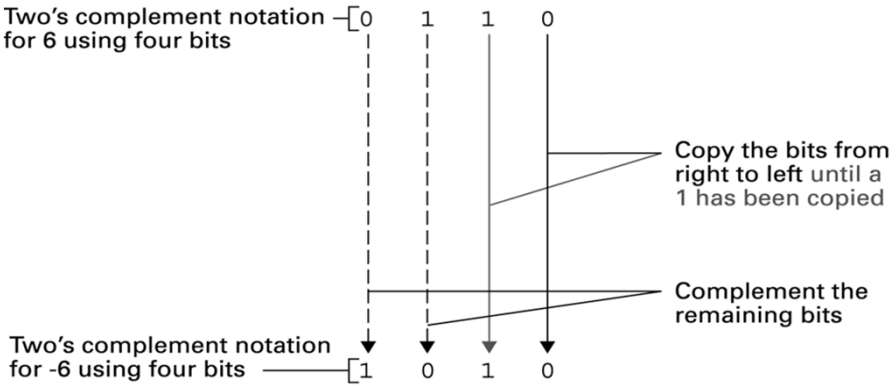


Figure 1.23 **Addition problems converted to two's complement notation (4 bits)**

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

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

With two's complement there is only one operation: addition.

There is no need for subtraction!

Addition problems converted to two's complement notation (4 bits): Exercise

$$\begin{array}{r} 4 \\ +\ 3 \\ \hline \end{array}$$

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

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 1.24 An excess eight conversion table

2. Find first pattern with 1 as its most significant bit – this pattern will represent 0

1. Starting with string of 0s of the appropriate length count upward

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

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

This is an excess 4 notation.

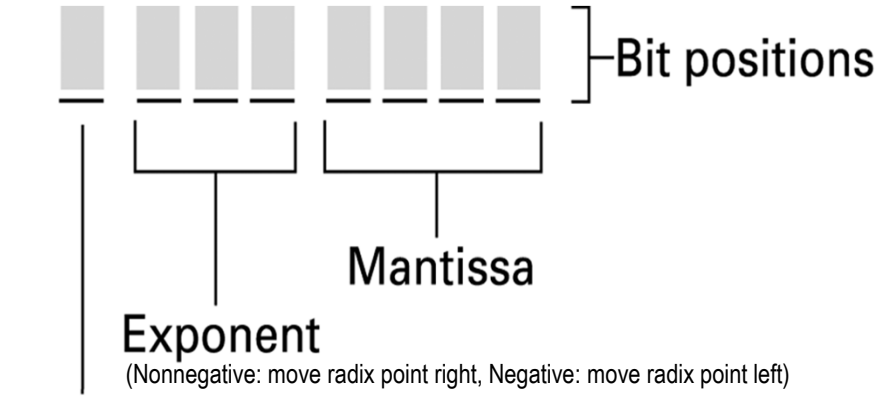
It is an excess 4 notation because the binary representation exceeds the excess notation by the order of 4. (e.g. 100=4 vs. 0)

Magnitude of the largest negative number indicates the excess value of the notation.

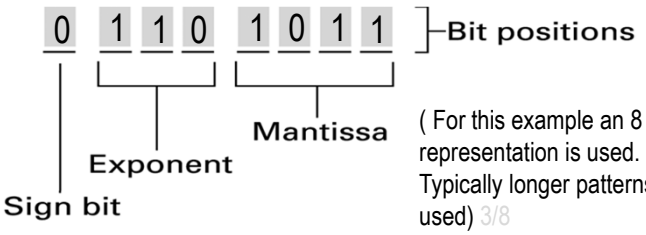
Storing Fractions

- **Floating-point Notation:** Consists of a sign bit, an exponent field, and a mantissa field.
- Related topics include
  - Normalized form
    - Representations that conform to filling in the mantissa field ***starting with the leftmost 1***.
  - Truncation errors (or round-off error)
    - Part of the value being stored is lost because the mantissa field is not large enough.

Figure 1.26 **Floating-point notation components**



Floating-point notation components Example



(For this example an 8 bit representation is used. Typically longer patterns are used) 3/8

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

1. **Mantissa**: 1011 – place a radix point on its left side obtaining **.1011**
2. **Exponent**: 110 – interpret as 3-bit excess notation for 2. Move radix to right by 2: **10.11**
3. **Sign bit**: 0 – value is nonnegative

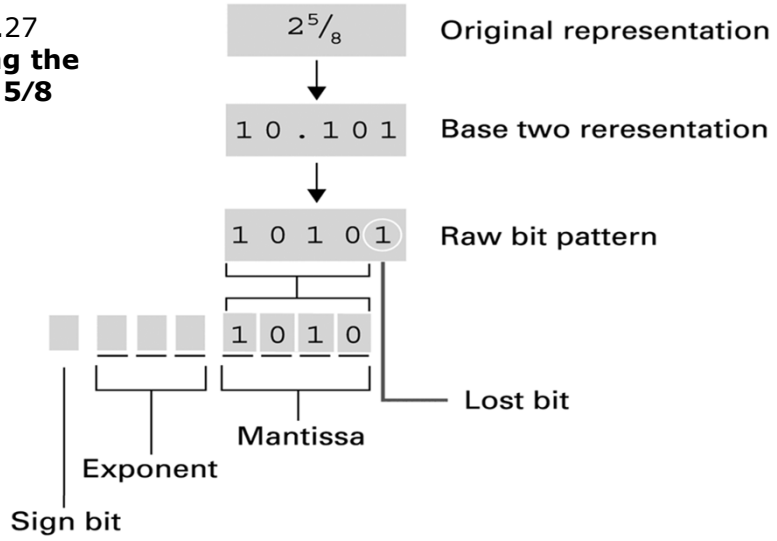
10.11 therefore represents the positive value of **2 ¾**

If the pattern had been **1**1101011 (same except for the sign bit) then the value represented would have been **-2 ¾** … what about 00111100

To store a value using floating point notation we reverse the process. M -> E -> SB
(fill the mantissa starting with the leftmost 1 that appears in the binary representation - normalized form)

Truncation

Figure 1.27
Encoding the value 2 5/8



Data Compression

- **Lossy** versus **lossless**
- **Run-length encoding** (lossless)
  - replacing sequences of identical data elements with a code indicating the element that is repeated and the number of times it occurs in the sequence.
- **Frequency-dependent encoding** (lossless)
  - length of the bit pattern used to represent a data item is inversely related to the frequency of the item's use (e.g. Huffman codes)
- **Relative encoding** (differential encoding)
  - record the differences between consecutive data units rather than entire units; that is, each unit is encoded in terms of its relationship to the previous unit (e.g. motion picture)
- **Dictionary encoding** (dynamic dictionary encoding)
  - compress text documents uses dictionaries that are allowed to change during the encoding process. Includes adaptive dictionary encoding such as LZW encoding.

Compressing Images

- **GIF** (Graphic Interchange Format)
  - Good for cartoons
- **JPEG** (Joint Photographic Experts Group)
  - Good for photographs
- **TIFF** (Tagged Image File Format)
  - Good for image archiving

Compressing Audio and Video

- **MPEG** (Motion Picture Experts Group)
  - Encompasses a variety of standards for different applications
    - High definition television broadcast
    - Video conferencing
- **MP3** (MPEG layer 3)
  - Among other compression techniques, MP3 takes advantage of the properties of the human ear, removing those details that the human ear cannot perceive
    - **Temporal masking**: For a short period after a loud sound, the human ear cannot detect softer sounds that would otherwise be audible
    - **Frequency masking**: A sound at one frequency tends to mask softer sounds at nearby frequencies.

Communication Errors

Error correcting techniques are used extensively to increase the reliability of computing equipment.

They are used in high capacity magnetic disk drives and CDs to reduce the error rate of the media.

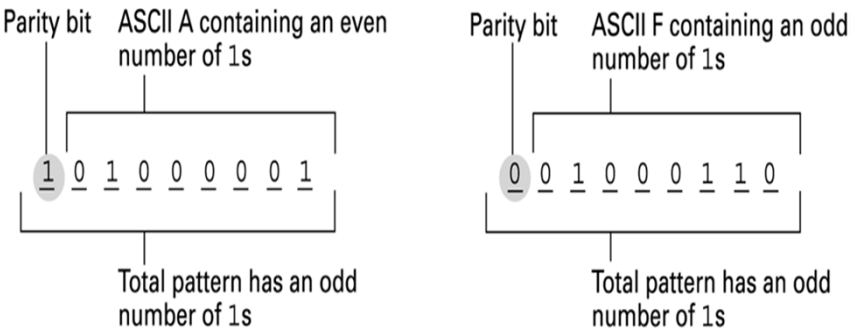
Examples of Encoding techniques:

- **Parity bits** (even versus odd)
- **Checkbytes**
- **Error correcting codes**

Parity Bits

Communication errors are detected based on the principle that all bit patterns manipulated have **odd parity** (an odd number of 1s), so if a pattern with **even parity** (an even number of 1s) is detected an error has occurred.

To ensure all bit patterns are odd, a **parity bit** is added.



Checkbytes

For long bit patterns, such as the string of bits recorded in a sector on a magnetic disk, the pattern is accompanied by a collection of parity bits making up a **checkbyte**.

- Each bit within the checkbyte is a parity bit associated with a particular collection of bits scattered throughout the pattern
- A collection of errors concentrated in one area of the original pattern is more likely to be detected, since it will be in the scope of several parity bits.
- Variations of this checkbyte concept lead to error detection schemes known as **checksums** and **cyclic redundancy checks** (CRC)

Error-correcting codes

Symbol	Code
A	000000
B	001111
C	010011
D	011100
E	100110
F	101001
G	110101
H	111010

Coding designed so that errors can not only be detected but also corrected.

The **Hamming distance** is the number of bits between two bit patterns that differ. (e.g. A & B: 4, A & C: 3)

From the example: if a bit is modified in transmission then the correct pattern will be a Hamming distance of 1 from its original form and 2 from other legal patterns.

i.e. at least 3 bits must be changed in any pattern before it will look like another legal pattern.

Figure 1.30 **Decoding the pattern 010100 using the code in Figure 1.30**

- The bit received **010100** is not a legal pattern.
- We compare it to the patterns in the code and conclude that it is D which is the closest match.
- This example allows us to detect 2 errors per pattern and correct 1.
- Error detection and correction increases with larger Hamming distances e.g. a distance of 5 would detect 4 and correct 2.

Character	Code	Pattern received	Distance between received pattern and code	
A	0 0 0 0 0 0	0 **1** 0 **1** 0 0	2	
B	0 0 1 1 1 1	0 **1** **0** 1 **0** **0**	4	
C	0 1 0 0 1 1	0 1 0 **1** **0** **0**	3	
D	0 1 1 1 0 0	0 1 **0** 1 0 0	**1**	Smallest distance
E	1 0 0 1 1 0	**0** **1** 0 1 **0** 0	3	
F	1 0 1 0 0 1	**0** **1** **0** **1** 0 **0**	5	
G	1 1 0 1 0 1	**0** 1 0 1 0 **0**	2	
H	1 1 1 0 1 0	**0** 1 **0** **1** **0** 0	4	