21世纪高等教育计算机规划教材

COMPUTER

大学计算机基础与应用（英文版）

Computer Fundamentals and Applications

刘朝斌 主编

- 讲授计算机基础知识
- 提高计算机应用技能
- 拓展信息技术通用英语能力

人民邮电出版社
POSTS & TELECOM PRESS

21世纪高等教育计算机规划教材

COMPUTER

大学计算机基础与应用（英文版）

Computer Fundamentals and Applications

刘朝斌 主编

人民邮电出版社
北京

图书在版编目（CIP）数据

大学计算机基础与应用 : 英文 / 刘朝斌主编. -- 北京 : 人民邮电出版社, 2014.3
21世纪高等教育计算机规划教材
ISBN 978-7-115-33587-6

Ⅰ. ①大… Ⅱ. ①刘… Ⅲ. ①电子计算机－高等学校－教材－英文 Ⅳ. ①TP3

中国版本图书馆CIP数据核字(2014)第015026号

内 容 提 要

本书根据教育部制定的《高等学校非计算机专业计算机课程基本要求》，结合目前计算机的发展和高校新生的现状而编写。

全书共分 7 章，主要内容包括计算机基础知识、操作系统基础、常用 Office 软件、Web 和 HTML 基础、Web 开发初步、数据库基础等。

本书内容翔实，结构合理，注重理论和实践相结合，具有通俗易懂、实用性和操作性强的特点。

本书适合作为各类高等学校非计算机专业教材，也可作为高等学校成人教育的计算机基础双语培训教材或自学参考书。

◆ 主　　编　刘朝斌
责任编辑　王亚娜
执行编辑　王志广
责任印制　张佳莹　焦志炜

◆ 人民邮电出版社出版发行　　北京市丰台区成寿寺路 11 号
邮编　100164　　电子邮件　315@ptpress.com.cn
网址　http://www.ptpress.com.cn
三河市海波印务有限公司印刷

◆ 开本：787×1092　1/16
印张：14.5　　2014 年 3 月第 1 版
字数：381 千字　　2014 年 3 月河北第 1 次印刷

定价：45.00 元

读者服务热线：(010)81055410　印装质量热线：(010)81055316
反盗版热线：(010)81055315

Preface

The transition from an industrial society to an information age is happening in decades. The main engine for this move has mainly come from the computer or computer technology. In the great growing age of information technology, students should learn how to use the latest computer technology and up-to-date applications.

This textbook mainly discusses the basic knowledge, fundamental concepts and applications of computer technology. It focuses on the important fundamentals and principles of computer hardware, network and operating system through the systematic explanation of computer science and technology. It is helpful to grasp the basic skills, understand the basic concepts for program designing, databases and the major computer application fields. It aims to offer a basic and very important subject for those who major in engineering. It is useful for students to be familiar with the important areas of typical computer cases and applications.

Especially, the book can be a relevant textbook or reference for a bilingual course in computer technology and its applications for college students. Students will master the basic knowledge of computer hardware, software, web development, database, operating system etc. As a result, a strong foundation will be laid for further study related to computer application.

This textbook covers 7 chapters, including computer basics, operating system, Microsoft Word 2003, PowerPoint 2003, Excel 2003, fundamentals of web development technology and database introduction.

This book couldn't have been finished without the support of my family and friends. Especially, I'd really like to thank Huihui Zhang, Tingting Wang, Kuo Liu and Peng Li for devoting their time and effort towards this book. This work has been partially supported by the National Natural Science Foundation of China (Grant NO.61370198) and China Scholarship Council Program. Last but not least, I also appreciate all of our technical reviewers and our press project coordinators.

2013-06-16

Zhaobin Liu

CONTENTS 目录

Chapter 1

Fundamentals of Computer

In this chapter, several important concepts about computer basics will be introduced, which mainly includes numbering systems, how computers represent data, the composition and working principle of the computer, the introduction of information security and computer virus.

1.1 Numbering Systems

People work with decimal digits every day. You've been using the decimal (base 10) numbering system for so long that you probably take it for granted. But, most modern computer systems do not represent numeric values using the decimal system. Instead, they typically use a binary or two's complement numbering system. Besides, octal (base 8) and hexadecimal (base 16) numbering systems are also in common use.

1.1.1 An Overview of the Numbering Systems

In any numbering system, a number can be written as:

$$N_r = a_{n-1} \times r^{n-1} + \cdots + a_1 \times r^1 + a_0 \times r^0 + a_{-1} \times r^{-1} + \cdots + a_{-m} \times r^{-m}$$

Where a_i is each digit of the number, r is the base of numbering systems, and r^i is the power of the base responding to a_i.

1. The Decimal Numbering System

In the decimal numbering system, each digit of a number can be numerical values between 0 and 9, and the base is 10. So, each digit appearing to the left of the decimal point represents a value between zero and nine timing an increasing power of ten. Digits appearing to the right of the decimal point represent a value between zero and nine timing an increasing negative power of ten. For example, the value 654.321 means:

$$(654.321)_{10} = 6 \times 10^2 + 5 \times 10^1 + 4 \times 10^0 + 3 \times 10^{-1} + 2 \times 10^{-2} + 1 \times 10^{-3}$$

2. The Binary Numbering System

In the binary numbering system, each digit of a number can only be numerical values between 0 and 1, and the base is 2. A binary digit (1 or 0) is called a bit and a group of 8 bits is called a byte. So 1011 is a 4-bit binary number. The left-end bit of a number represented in binary is called the most significant bit, abbreviated MSB, and the right-end bit is called the least significant bit, abbreviated LSB. Just like the decimal numbering system, a binary number [taking $(101.011)_2$ for example] is represented like the following.

$$(101.011)_2 = 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 + 0 \times 2^{-1} + 1 \times 2^{-2} + 1 \times 2^{-3}$$

Why do modern computer systems operate using binary numbering system? Most computer systems operate using binary logic. The computer represents value using two voltage levels, usually 0V for logic 0, and either +3.3V or +5V for logic 1. These two voltage levels represent exactly two different

values, and by convention, the values are zero and one. These two values coincidentally correspond to the two digits used by the binary number system. Since there is a correspondence between the logic levels used by the computer and the two digits used in the binary numbering system, it should come as no surprise that computers employ the binary system.

3. The Octal or Hexadecimal Numbering System

A big problem with the binary numbering system is verbosity. Sometimes, the binary number is hard to be remembered with more digits ('0'or'1'). So, the octal or hexadecimal number is often used as the mnemonic in computer systems.

In the octal numbering system, each digit of a number can be numerical values between 0 and 7, and the base is 8. Taking $(2405.71)_8$ for example, we can represent an octal number like the following.

$$(405.7)_8 = 4\times8^2 + 0\times8^1 + 5\times8^0 + 7\times8^{-1}$$

In the hexadecimal numbering system, each digit of a number can be numerical values between 0 and 15, and the base is 16. Usually, the letters A through F are used to represent the values in the range 10 through 15. So, an example of a hexadecimal number is as follows.

$$(A3E.D)_{16} = A\times16^2 + 3\times16^1 + E\times16^0 + D\times16^{-1}$$

1.1.2 Operations on Binary Numbers

1. Arithmetic Operations

There are several arithmetic operations on binary numbers, for example, addition, subtraction, multiplication, division. The following table lists the arithmetic operations on binary numbers.

Table 1-1 Arithmetic Operations

Addition	0+0=0	0+1=1	1+0=1	1+1=0
Subtraction	$0-0=0$	$0-1=1$	$1-0=1$	$1-1=0$
Multiplication	$0\times0=0$	$0\times1=0$	$1\times0=0$	$1\times1=1$
Division	$0\div0=0$	$0\div1=0$	$1\div0=0^*$	$1\div1=1$

*The arithmetic operation 1÷0 is of insignificance.

The following is an example of subtraction operation on binary numbers.

$$\begin{array}{r} (1101)_2\cdots(13)_{10} \\ -(1011)_2\cdots(11)_{10} \\ \hline (0010)_2\cdots(2)_{10} \end{array}$$

2. Logical Operations

There are four main logical operations on binary numbers: AND, OR, XOR (exclusive-or), and NOT. The logical AND, OR, XOR operators are dyadic operators (meaning they accept exactly two operands).

The logical AND operator is, "If the first operand is one and the second operand is one, the result is one; otherwise the result is zero." We could also state this as "If either operand (or both) is zero, the result is zero." The AND operation is usually represented by the symbol " $\wedge$" or " $\cdot$ ".Its definition is as follows.

$$0\wedge0=0$$
$$0\wedge1=0$$

$$1 \wedge 0 = 0$$
$$1 \wedge 1 = 1$$

The logical OR operator is, "If the first operand or the second operand (or both) is one, the result is one; otherwise the result is zero." This is also known as the inclusive-OR operation. The OR operation is usually represented by the symbol "$\vee$". Its definition is as follows.

$$0 \vee 0 = 0$$
$$0 \vee 1 = 1$$
$$1 \vee 0 = 1$$
$$1 \vee 1 = 1$$

The logical XOR operator is, "If the first operand or the second operand, but not both, is one, the result is one; otherwise the result is zero." Note that the exclusive-or operation is closer to the English meaning of the word "or" than is the logical OR operation. The XOR operation is usually represented by the symbol "$\oplus$". Its definition is as follows.

$$0 \oplus 0 = 0$$
$$0 \oplus 1 = 1$$
$$1 \oplus 0 = 1$$
$$1 \oplus 1 = 0$$

The logical NOT operator is a monadic operator (meaning it accepts only one operand). The NOT operation is usually represented by the symbol "$\overline{\ }$". Its definition is as follows.

$$\overline{1} = 0$$
$$\overline{0} = 1$$

The true table for the logical operation on binary numbers takes the following form: zero represents "false" and one represents "truth".

Table 1-2 Logical Operations

a	b	$a \wedge b$	$a \vee b$	$a \oplus b$	$\overline{a}$
0	0	0	0	0	1
0	1	0	1	1	1
1	0	0	1	1	0
1	1	1	1	0	0

1.1.3 Conversion Between Different Numbering Systems

1. Converting from Other Numbering Systems to Decimal

It is easy to convert a binary, octal or hexadecimal number to a decimal number, which can be done by three steps.

(1) Expanding a number in a power series;

(2) Multiplying each digit of a number by the responding power of its base, then adding up the result;

(3) If any digit is 0, the responding power is not to be taken into account.

For example, the binary number $(101.011)_2$ shown previously is represented as follows.

$$(101.011)_2 = 1\times2^2 + 0\times2^1 + 1\times2^0 + 0\times2^{-1} + 1\times2^{-2} + 1\times2^{-3} = 4 + 0.25 + 0.125 = (4.375)_{10}$$

In the same way, the octal number $(2405.71)_8$ and the hexadecimal number $(A3E.D)_{16}$ can

represent respectively decimal numbers as follows.

$$(405.7)_8 = 4\times8^2 + 0\times8^1 + 5\times8^0 + 7\times8^{-1} = 256 + 5 + 0.875 = (270.875)_{10}$$

$$(\text{A3E. D})_{16} = \text{A}\times16^2 + 3\times16^1 + \text{E}\times16^0 + \text{D}\times16^{-1}$$

$$= 2560 + 48 + 14 + 0.8125 = (2622.8125)_{10}$$

2. Converting from Decimal to Other Numbering systems

To convert a decimal number to a binary, octal or hexadecimal number is a reverse operation. If a decimal number includes a radix point, it is necessary to separate the number into an integer part and a fraction part, since each part must be converted differently. The conversion of a decimal integer to other numbering systems is done by dividing the number and all successive quotients by the responding base and accumulating the remainders. This procedure is best illustrated by the following example as shown in Figure 1-1.

Consider the conversion from the decimal $(51)_{10}$ to a binary number. First, 51 is divided by 2 to give an integer quotient of 25 and a remainder of 1. The quotient is again divided by 2 to give a new quotient and remainder. This process is continued until the integer quotient becomes 0. So, the desired binary number obtained from the remainders is as follows.

$$(51)_{10} = (a_5a_4a_3a_2a_1a_0)_2 = (110011)_2$$

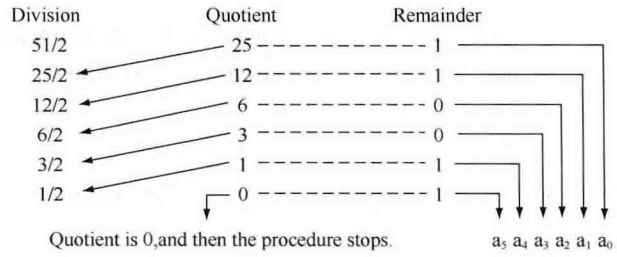


Figure 1-1 Conversion from a Decimal Integer to a Binary Number

The conversion of a decimal fraction to a binary number is accomplished by a method similar to that used for decimal integers. However, multiplication is used instead of division, and integers are accumulated instead of remainders. Again, the method is best explained by the following example.

Consider the conversion from the decimal $(0.74)_{10}$ to a binary number in Figure 1-2. First, 0.74 is multiplied by 2 to give an integer and a fraction. The new fraction is multiplied by 2 to give a new integer and a new fraction. This process is continued until the fraction becomes 0 or until the number of digits has sufficient accuracy. So, four figures required, the binary number is obtained from the integers as follows.

$$(0.74)_{10} = (a_{-1}a_{-2}a_{-3}a_{-4})_2 = (0.1011)_2$$

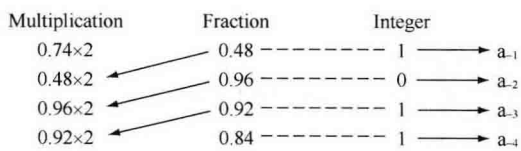


Figure 1-2 Conversion from a Decimal Fraction to a Binary Number

The conversion of decimal numbers with both integer and fraction parts is done by converting the integer and the fraction separately and then combining the two answers.

As similar as the conversion from decimal to binary, we can convert decimal to octal and hexadecimal just by being divided or multiplied by 8 and 16.

3. Conversion between Binary and Octal or Hexadecimal Numbering Systems

The conversion from and to binary, octal, and hexadecimal plays an important role in modern computers. Since $2^3 = 8$ and $2^4 = 16$, each octal digit corresponds to three binary digits and each hexadecimal digit corresponds to four binary digits. The first 16 numbers in the decimal, binary, octal, and hexadecimal number systems are listed in Table 1-3.

The conversion from binary to octal is easily accomplished by partitioning the binary number into groups of three digits each, starting from the binary point and proceeding to the left and to the right. The corresponding octal digit is then assigned to each group. The following example illustrates the procedure. Convert binary $(11101111.110010)_2$ to an octal number.

$$\left(\frac{011}{3}\ \frac{101}{5}\ \frac{111}{7}.\frac{110}{6}\ \frac{010}{2}\right)_2 = (357.62)_8$$

Conversion from binary to hexadecimal is similar, except that the binary number is divided into groups of four digits. Convert binary $(11110111110.0110101)_2$ to a hexadecimal number.

$$\left(\frac{0111}{7}\ \frac{1011}{B}\ \frac{1110}{E}.\frac{0110}{6}\ \frac{1010}{A}\right)_2 = (7BE.6A)_{16}$$

The corresponding hexadecimal (or octal) digit for each group of binary digits is easily remembered after studying the values listed in Table 1-3. Conversion from octal or hexadecimal to binary is done by reversing the preceding procedural. Each octal digit is converted to its three-digit binary equivalent; similarly, each hexadecimal digit is converted to its four-digit binary equivalent, as are illustrated in the following examples.

$$(352.46)_8 = (011\ 101\ 010.100\ 110)_2 = (11101010.10011)_2$$

$$(C48.DA)_{16} = (1100\ 0100\ 1000.1101\ 1010)_2 = (110001001000.1101101)_2$$

Table 1-3 Numbers with Different Bases

Decimal (base 10)	Binary (base 2)	Octal (base 8)	Hexadecimal (base 16)
0	000	0	0
1	001	1	1
2	010	2	2
3	011	3	3
4	100	4	4
5	101	5	5
6	110	6	6
7	111	7	7
8	1000	10	8
9	1001	11	9
10	1010	12	A
11	1011	13	B
12	1100	14	C
13	1101	15	D
14	1110	16	E
15	1111	17	F

1.1.4 Representation about Negative Numbers

So far we have discussed numbering systems which are mainly about positive number. In this section, we will discuss some representations about negative numbers.

1. Data Representation

In computers, common collections of data size are single bits, groups of four bits (called nibble), groups of eight bits (byte), groups of 16 bits (word), and groups of 32 bits (double word).

（1）Bit

The smallest "unit" of data on a binary computer is a single bit. With a single bit, you can represent any two distinct items. Examples include zero or one, true or false, and on or off.

（2）Nibble

A nibble is a collection of four bits. It wouldn't be a particularly interesting data structure except for two items: BCD (binary coded decimal) numbers and hexadecimal numbers. It takes four bits to represent a single BCD or hexadecimal digit.

（3）Byte

A byte consists of eight bits and is the smallest addressable datum used in main memory or I/O addresses. Since a byte contains eight bits, it can represent 2^8, or 256, different values. Generally, we will use a byte to represent numeric values in the range 0–255, signed numbers in the range-128–+127, ASCII character codes, and other special data types requiring no more than 256 different values. Many data types have fewer than 256 items, so eight bits are usually sufficient.

（4）Word

A word is a group of 16 bits. With 16 bits, you can represent 2^{16} (65,536) different values. These could be the values in the range 0–65,535 or, as is usually the case, -32,768–+32,767, or any other data type with no more than 65,536 values. The three major uses for words are unsigned integer values, signed integer values, and UNICODE characters. Unsigned numeric values are represented by the binary value corresponding to the bits in the word. Signed numeric values use the two's complement form for numeric values. As to UNICODE characters, words can represent up to 65,536 different characters, allowing the use of non-Roman character sets in a computer program.

（5）Double word

A double word is exactly what its name implies, a pair of words. Double words can represent all kinds of different things. A common item you will represent with a double word is a 32-bit integer value, 32-bit floating point values commonly used to store pointer variables.

2. Sign Magnitude Representation

Positive integers (including zero) can be represented as unsigned numbers. However, to represent negative integers, we need a notation for negative values. In ordinary arithmetic, a negative number is indicated by a minus sign and a positive number by a plus sign. It is customary to represent the sign with a bit placed in the leftmost position of the number. The sign and magnitude approach is to represent a number's sign by allocating one sign bit to represent the sign: set that bit (often the most significant bit) to 0 for a positive number, and set to 1 for a negative number. The remaining bits in the number indicate the magnitude (or absolute value).

In the below example showed in Figure 1-3, the string of bits 01011 can be considered as 11

(unsigned binary) or +11 (signed binary) because the leftmost bit is 0. The string of bits 11011 represent the binary equivalent of 27 when considered as an unsigned number or as -11 when considered as a signed number. This is because the 1 that is in the leftmost position designates a negative and the other four bits represent binary 11. Usually, there is no confusion in identifying the bits if the type of representation for the number is known in advance. Note that a positive number is represented similarly to an unsigned number. From the example, it is also evident that only 4-bits are used to represent the magnitude.

Figure 1-3 Sign Magnitude Representation

3. Complement Representation

Complements are used in digital computers for simplifying the subtraction operation and for logical manipulation. There are two types of complements for binary numbers: the 1's complement and 2's complement.

(1) 1's Complement

1's complement of a binary number is obtained simply by replacing each 1 by 0 and each 0 by 1. Alternately, 1's complement of a binary can be obtained by subtracting each bit from 1.

The following are some numerical examples.

Replace each 1 by 0 and each 0 by 1.

$$\begin{array}{ccccccc} 0 & 1 & 0 & 0 & 1 & 0 & 1 \\ \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow \\ 1 & 0 & 1 & 1 & 0 & 1 & 0 \end{array}$$

So, 1's complement of 0100101 is 1011010.

Subtract each binary bit from 1.

$$\begin{array}{ccccccc} 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ - \; 0 & 1 & 0 & 0 & 1 & 0 & 1 \\ \hline 1 & 0 & 1 & 1 & 0 & 1 & 0 \end{array}$$

We can see that both methods give the same result.

(2) 2's Complement

2's complement of a binary number can be obtained by adding 1 to its 1's complement, which is illustrated in examples below.

Example 1-1:

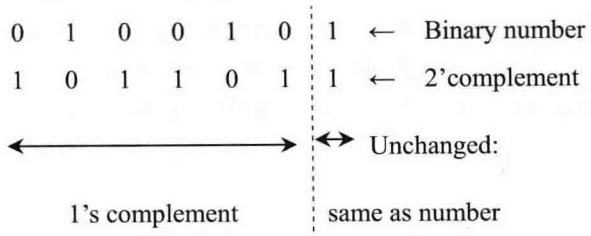
$$\begin{array}{ccccccc} 0 & 1 & 0 & 0 & 1 & 0 & 1 \quad \leftarrow \text{Binary number} \\ 1 & 0 & 1 & 1 & 0 & 1 & 0 \quad \leftarrow \text{1'complement} \\ & & & & & + & 1 \quad \leftarrow \text{Add 1 to 1'complement} \\ \hline 1 & 0 & 1 & 1 & 0 & 1 & 1 \quad \leftarrow \text{2'complement} \end{array}$$

Example 1-2:

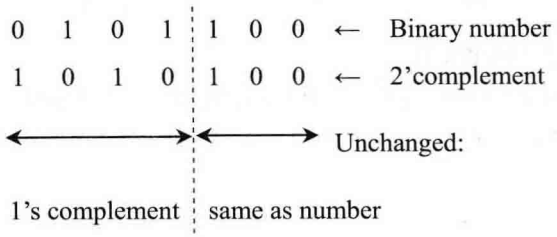
$$\begin{array}{ccccccc} 0 & 1 & 0 & 1 & 1 & 0 & 0 \quad \leftarrow \text{Binary number} \\ 1 & 0 & 1 & 0 & 0 & 1 & 1 \quad \leftarrow \text{1'complement} \\ & & & & & + & 1 \quad \leftarrow \text{Add 1 to 1'complement} \\ \hline 1 & 0 & 1 & 0 & 1 & 0 & 0 \quad \leftarrow \text{2'complement} \end{array}$$

There is an efficient method to find 2's complement based upon the observation of the above two examples. Consider the binary number and its 2's complement in examples as shown below.

Example 1-3:



Example 1-4:



The above examples clearly show that to find 2's complement of a binary number, starts from right towards left till the first 1 appears in the number. Take these bits (including first 1) as it is and take 1's complement of rest of the bits.

In the computer system, it is more convenient to use the signed-complement system for representing negative numbers. In this system, a negative number is indicated by its complement. Whereas the signed magnitude system negates a number by changing its sign, the signed-complement system negates a number by taking its complement. Since positive numbers always start with 0 (plus) in the leftmost position, the complement will always start with a 1, indicating a negative number. The signed-complement system can use either the 1's or the 2's complement, but the 2's complement is the most common.

So, to represent a negative number using complements involves two steps.

（1）Obtain the binary representation of positive number equivalent for a given negative number. For example, if a given number is –6, then the binary representation is +6.

（2）Take the appropriate complement of representation obtained in step 1.

As an example, consider the number -38 represented in binary with eight bits.

Signed-magnitude representation: 10100110

Signed-2's-complement representation: 11011010

In signed-magnitude, -38 is obtained from +38 by changing the sign bit in the leftmost position from 0 to 1. The signed-2's-complement representation of -38 is obtained by taking the 2's complement of the positive number, including the sign bit.

4. Arithmetic Addition and Subtraction

In the signed-complement system, the 1's complement imposes some difficulties and is seldom used for arithmetic operations. It is useful as a logical operation since the change of 1 to 0 or 0 to 1 is equivalent to a logical complement operation. The following discussion of signed binary arithmetic deals exclusively with the signed-2'scomplement representation of negative numbers.