

1 Information representation – Part 1

Name: _____ Class: _____ Date: _____

Vocabulary 词汇

Write each word three times. 把每个单词抄写三遍。

English	中文	Write it 3 times (English)		
bit	位	_____	_____	_____
byte	字节	_____	_____	_____
binary	二进制	_____	_____	_____
denary	十进制	_____	_____	_____
number system	数制	_____	_____	_____
place value	位值	_____	_____	_____
hexadecimal	十六进制	_____	_____	_____
nibble	半字节	_____	_____	_____
remainders	余数	_____	_____	_____
two's complement	补码	_____	_____	_____
sign bit	符号位	_____	_____	_____
overflow	溢出	_____	_____	_____
BCD	二进制十进数	_____	_____	_____
character set	字符集	_____	_____	_____

Recall

You already started this at IGCSE. Bring it with you:

- A **bit** 位 is a single 0 or 1. Eight bits make one **byte** 字节.
- You used **binary** 二进制 (base 2) and **denary** 十进制 (base 10, the everyday numbers), and converted between them.
- At AS we add three new ideas: **hexadecimal**, **negative** numbers in binary, and **BCD**.

Nothing here is harder than IGCSE —it builds straight on the binary you already know.

New ideas

Read these first. Each idea has a worked example. Then do the Practice below.

■ 1 Number bases and the general method

A **number system** 数制 uses a fixed set of digits, and a digit's value depends on its **place value** 位值 (which column it is in). Binary is base 2, denary base 10, and **hexadecimal** 十六进制 base 16. Four bits make a **nibble** 半字节. The two methods below work for **any** base —learn these, not a binary-only trick.

Any base → **denary**. Multiply each digit by the base raised to its position (count positions from 0 at the right), then add.

2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
1	0	1	1	0	0	1	0

$$\begin{aligned} & (1 \times 2^7) + (1 \times 2^5) + (1 \times 2^4) + (1 \times 2^1) \\ & = 128 + 32 + 16 + 2 = \mathbf{178} \end{aligned}$$

$$10110010_2 = (1 \times 2^7) + (1 \times 2^5) + (1 \times 2^4) + (1 \times 2^1) = 128 + 32 + 16 + 2 = 178.$$

For hexadecimal you use powers of 16 instead —exactly the same method.

Denary → **any base**. Divide by the base again and again; the **remainders** 余数, read from bottom to top, are the digits.

$178 \div 2 = 89$	remainder 0	↑ read up = 10110010
$89 \div 2 = 44$	remainder 1	
$44 \div 2 = 22$	remainder 0	
$22 \div 2 = 11$	remainder 0	
$11 \div 2 = 5$	remainder 1	
$5 \div 2 = 2$	remainder 1	
$2 \div 2 = 1$	remainder 0	
$1 \div 2 = 0$	remainder 1	

To get hexadecimal instead, divide by 16 the same way.

■ 2 Working with hexadecimal

Hexadecimal digits are 0–9, then *A, B, C, D, E, F* for 10 to 15. It is a short way to write binary, so programmers use it for memory addresses and colour codes.

Hex → *denary* is the same powers method, with base 16:

$$0xB2 = (11 \times 16^1) + (2 \times 16^0) = 176 + 2 = 178.$$

Shortcut (binary → hex): group the bits into nibbles of four from the right, and convert each nibble on its own.



1011 0010 -> B 2 so 10110010 = 0xB2

■ 3 Negative numbers: two's complement

To store a negative whole number we use **two's complement** 补码. The left-most bit becomes the **sign bit** 符号位: 0 means positive, 1 means negative.

To write a negative number (8 bits): write the positive value, flip every bit, then add 1.

Worked example —store -5:

step	bits
+5	0000 0101
flip every bit	1111 1010
add 1	1111 1011

So -5 is 11111011. If a sum needs more bits than the register has, you get **overflow** 溢出 and the answer is wrong.

	0000 0101	= +5
↓		
	1111 1010	flip every bit
↓		
	1111 1011	add 1 = -5

Watch out: flip the bits *then add 1* —just once. Flipping a second time only takes you back to where you started.

■ 4 Binary Coded Decimal (BCD)

In **BCD** 二进制十进数 each denary digit is written as its own 4-bit nibble. It keeps the digits separate (useful for clocks and calculators).

Worked example: 59 in BCD is 0101 1001 —that is 5 then 9, **not** the pure binary 00111011.

■ 5 Text is data too

Letters are stored with a **character set** 字符集—a table giving each symbol a number (for example, ASCII stores A as 65). The computer still stores only bits; the character set says what they mean.

Practice

Try these in order. The methods are all above. Show your working.

2.1 Convert the denary number 45 to an 8-bit binary number. [2]

2.2 Convert the binary number 00110100 to denary. [1]

2.3 Convert the binary number 11101010 to hexadecimal. [2]

2.4 Convert the hexadecimal number 2F to denary. [1]

2.5 Using 8-bit two's complement, write the number -12 . Show each step. [3]

2.6 Write the denary number 72 in BCD. [2]

2.7 Give one reason programmers use hexadecimal instead of long binary numbers. [1]

2.8 In 8-bit two's complement, explain *why* the left-most bit alone tells you whether the number is negative. [2]
