

**FORM 4 COMPUTER STUDIES****Chapter 7: Floating Point Representation of Numbers**

March, 01

**7.1 LIMITATION OF FIXED POINT REPRESENTATION****7.1.1 Range**

- The range of numbers that can be represented is small.
- e.g. Below is an example of fixed point representation with the binary point fixed between 2 and 3 in an 8-bit word.

	Binary code	Decimal value
Largest number	01111.111	15.875
Smallest number	11111.111	-15.875

**7.1.2 Accuracy**

- *Truncation error* (截尾誤差)

e.g. Suppose that the number 1100.100111 is to be stored in the fixed point format shown in this section. The number would become 01100.100. The last three digits are removed because there is not enough space to accommodate them. A truncated number is *always less than* its original value.

- *Rounding error* (捨入誤差)

e.g. If the first bit removed is a 1, the number is rounded by adding 1 to the last bit retained. The same number will thus be represented as 01100.101. A rounded number is not always larger than its original value.

**7.2 FLOTATION POINT REPRESENTATION**

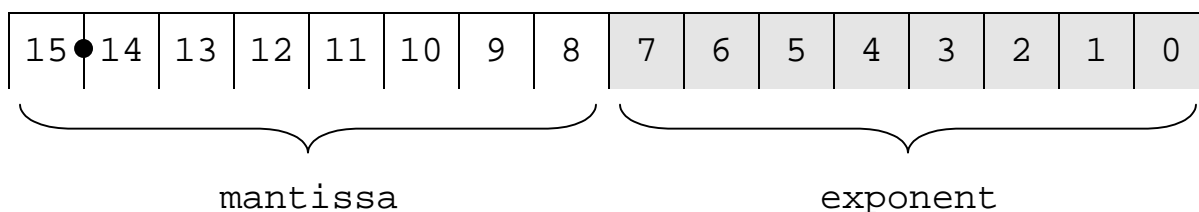
- the number is split up into TWO parts : a **mantissa** (尾數) and an **exponent** (指數). In mathematics, real numbers may be written in scientific notation as follows:

$$1357.9 = 1.3579 \times 10^3 = 1.3579E+03$$

Mantissa

Exponent

- Below is an example of a floating point format in a certain computer. 16 bits are used. Bit 15 to bit 8 represent the mantissa and bit 7 to bit 0 represent the exponent. For simplicity, sign-and-magnitude representation is used for both the mantissa and exponent.



- Normalization** : In order to standardize the format of representation, a technique called normalization (規格化) is used. Normalization ensures a unique representation of numbers and maximum possible accuracy.

e.g. Suppose the floating point format is used to represent the number  $4.25_{10}$ . It may be expressed in different ways:

	mantissa	exponent
$4.25_{10}$		
$= .10001_2 \times 2^3$	0 1000100	0 0000011
$= .010001_2 \times 2^4$	0 0100010	0 0000100
$= .00010001_2 \times 2^6$	0 0001000	0 0000110

In the third representation above, the last digit of the mantissa is lost, thus causing an error. The first representation is said to be normalized since the first bit of the mantissa is non-zero.

### Class Exercise

(1) Express  $1101.101_2$  in floating point representation.

(2) Express  $0.0000000001101_2$  in floating point representation.

(3) Find the decimal value of the number whose floating point representation is as follows:  
**1100101000000101**

■ **Range :** In floating point representation, the largest possible number is a number with the largest mantissa and the largest positive exponent.



i.e.  $0.9921875 \times 2^{127} = 1.688 \times 10^{38}$  approximately

An attempt to represent a number whose magnitude is greater than this upper limit will cause an overflow error.

On the other hand, to find the smallest positive number, we need the smallest mantissa and the largest negative exponent.



i.e.  $0.5 \times 2^{-127} = 2.939 \times 10^{-39}$  approximately

An underflow error occurs if the magnitude of a number exceeds this lower limit. That is, when the exponent is ‘too negative’ (i.e. equal to or less than -127). Some computers will set the number to zero.

To increase the range, we can allocate more bits to the exponent.

■ **Precision :** Long fractions cannot be represented exactly in a finite number of bits. Hence, truncation error or rounding error may occur. Precision is increased if more bits are allocated to the mantissa.

## 7.3 PARITY CHECKING

Accurate data transfer is extremely important in a computer system. Signals in the form of electrical impulses may get corrupted in the course of transmission by electrical noise. A binary zero may turn into a binary one or vice versa.

### 7.3.1 *Parity checking* (奇偶檢驗) :

**Parity checking** is a simple method of checking the correctness of received data. An ASCII code is 8 bits long. Normally, only the rightmost 7 bits are used to represent a character leaving the most significant bit 0. This 8th bit can be used as a parity bit. The parity bit is not always the 8th bit.

- **Even parity** (偶數奇偶檢驗) : To maintain even parity, the parity bit is set to 1 if the code being sent has an odd number of 1s. This ensures that the number of 1's is always even (hence the term even parity). When the code already has an even number of 1s, the parity bit is set to 0. In the example below, the message is sent with even parity.

Character	ASCII value		Binary code even parity
	Decimal	Binary	
H	72	01001000	01001000
E	69	01000101	11000101
L	76	01001100	11001100
O	79	01001111	11001111

If there is an even number of 1s in the byte of data received, then the data are assumed to be correct. Otherwise, an error occurs.

- **Odd parity** (奇數奇偶檢驗) : The idea is similar to even parity except that the number of 1s in each byte is made odd by setting the appropriate value of the parity bit.

END OF CHAPTER 7