Storing Numbers

Chapter No.	22
Status	Completed

- Storing Integers
  - Integers are stored in binary notation
    - For instance, a computer program may dedicate one byte (eight bits) to storing an integer
    - This means that the integer can be a value between 00000000 (0 in denary) and 11111111 (255 in denary)
  - The main advantage of the binary system is that very few arithmetic operations need to be hard-coded into the computer for it to be able to carry out mathematical operations
  - Performing operations in binary notation works as long as the final answer is within the range of integers that can be encoded with one byte (0 to 255)
  - If we want to store larger numbers, we would need more than one byte to do so
    - Most computer systems use either two or four bytes to store integers, allowing them to go up to (2^16 - 1) = 65,535 or (2^32 -1) = 4,294,967,295 respectively
    - However, overflow errors may still occur if the computer is not pre-empted of an incoming large result when it makes a calculation
  - 2 Methods of Storing Negative Integers
    - Using One Bit for the Sign
      - The first bit, which is also known as the most significant bit (MSB), represents the sign of the integer
      - For instance, 0 indicates a positive number and 1 indicates a negative number

- The remaining seven bits are used to indicate the magnitude of the number
- Therefore, using one byte, we can represent numbers between -127 and 127
- However, we have to modify the arithmetic operations accordingly to allow for adding negative numbers
- This system also has a positive and negative zero (10000000 and 00000000), which are two different encodings of the same number

▼ Two's Complement

- The most significant bit (MSB) represents the negative number 2^-7
- The remaining seven bits represent positive powers of 2 as usual
- If the MSB is 0, the remaining seven bits are the usual binary representation of a number from 0 to 127
- If the MSB is 1, we decode the remaining seven bits as a number between 0 and 127, then subtract 128 from it
- The advantage of this method is that the arithmetic operations do not need to be modified to allow for adding negative numbers
- Another way to obtain the binary encoding of a negative number is to write down the positive number with the same magnitude in binary, then change all the digits from 1 to 0 and vice versa, then add 1 (2^0) to the final answer

▼ Storing Real Numbers

▼ In the denary system, the digits after the decimal point indicate negative powers of 10

- We can write it in the form a x 10^b, where -1 < a < 1
- For example, 23.456 = 0.23456 x 10^2

- a = 0.23456 is the mantissa
- b = 2 is the exponent

▼ Likewise, in the binary system, the digits after the bicimal point (radix point) indicate negative powers of 2

- We can write it in the form a x 2^b, where -1 < a < 1
- For example, 10.1011 = 0.101011 x 2^2
- a = 0.101011 is the mantissa
- b = 2 is the exponent

▼ We can thus store any real number as a mantissa and an exponent

- This is known as floating point representation because the bicimal or decimal point floats into position, depending on the value of the exponent

▼ Suppose we want to store a number using two bytes

- We can use the first byte to store the mantissa and the second byte to store the exponent

▼ Example - Denary 112

- Denary 112 = Binary 1110000 = Binary 0.1110000 x 2^111
- The exponent (denary 7) is also in binary
- Thus, denary 112 is encoded as 01110000 00000111 (mantissa and exponent)

▼ Problems with Floating Point Numbers

- For infinitely recurring bicimal numbers, the mantissa has to be truncated at a certain number of significant figures before mathematical operations can be carried out
- This results in a loss of precision in the final answer
- These rounding errors can become significant if calculations are repeated enough times
- The only way of preventing this from becoming a serious problem is to increase the precision of the floating-point representation by

using more bits for the mantissa

- Another problem is that if a very small real number is divided by a very large one, the resultant value could be smaller than the smallest possible number than can be stored, leading to an underflow error