Representation of Integers

There are three different ways of handling binary numbers

* Sign and magnitude
* One’s complement
* Two’s complement
* Binary coded decimal

Sign and Magnitude

In a binary number the first bit (left most bit) indicates the sign of the number

* 1 means negative
* 0 means positive
* The other bits give absolute value of the number

Example 10102 represents -2

01012  represents +5

One’s complement

The one’s complement of binary is obtained by flipping the bits

* 1 change 0
* 0 change 1

The one’s complement 01101 is 10010

What is the ones complement of -37 using 8 bits

Find the binary equivalent of 37 = 100101

To make 8 bits add 2 zeros 00100101  
the number is negative so add 1 to the left = **1**00100101

Two’s complement

Graphical user interface, text, application

Description automatically generated with medium confidence

Representing real numbers

Key Terms

* Mantissa – the fractional part of a floating-point number.
* Exponent – the power of 2 that the mantissa (fractional part) is raised to in a floating-point number.
* Binary floating-point number – a binary number written in the form M × 2E (where M is the mantissa and E is the exponent).
* Normalization (floating-point) – a method to improve the precision of binary floating-point numbers; positive numbers should be in the format 0.1 and negative numbers in the format 1.0.
* Overflow – the result of carrying out a calculation which produces a value too large for the computer’s allocated word size.
* Underflow – the result of carrying out a calculation which produces a value too small for the computer’s allocated word size.

Floating point numbers are normalized because:

* **For precision and range:** Floating point numbers have a limited number of bits allocated to the mantissa and exponent. By normalizing the numbers, we maximized the precision available within this limited range.
* Avoid redundancy. It removes un-necessary leading zeros in the mantissa. This reduces the storage requirement and improves computational efficiency.
* Normalized floating point numbers are easier to compare because they are represented in a consistent way format, making operations like addition, subtraction, multiplication and division more straight forward.
* Rounding errors can occur when performing operation on floating point numbers, this is due to finite precision.

Overflow:

Overflow happens when the exponent of a floating-point number becomes too large for the chosen format to represent.

Cause during Normalizatio: if the exponent grows too large, it may exceed the maximum allowable exponent for the chosen floating-point format. If this happens, the floating-point system might trigger an overflow condition.

The result of an overflow can vary depending on the system. It might lead to an error condition, return an "infinity" value (if supported), or cause undefined behavior.

Example: In a 32-bit single-precision format (like IEEE 754), if the exponent field reaches its maximum value of 255 (assuming no bias), any attempt to further increase the exponent would result in an overflow.

Underflow occurs when the exponent of a floating-point number becomes too small for the chosen format to represent accurately.

When normalizing a very small number, if the exponent becomes too negative, it might fall below the minimum allowable exponent. In some cases, underflow might result in a loss of precision or a representation of zero. Depending on the system, it may also trigger an underflow exception or return a subnormal (denormalized) value.

Example: In a 64-bit double-precision format (like IEEE 754), if the exponent field reaches its minimum value (assuming no bias), any further decrease in the exponent would lead to an underflow condition

What is Normalization

The process of ensuring the maximum accuracy for a fixed number of bits is known as ‘normalization’.

Normalization ensures that maximum accuracy of a number for a given range of bits. It also ensures that each number has only one possible bit pattern to represent it! 3004 x104 normalized is 0.3004 x 108

**Normalization of positive binary numbers**  
To normalize a positive binary the first two numbers in the mantissa must be '01'.

**Example 25. Convert this number into its normalized form: 0000 1101 0100 0100**

* 1. The mantissa has 10 bits and there are 6 bits for the exponent.
  2. The mantissa is 00001101011, which isn't normalized. We know this because this is a positive number (the first bit from the left of the mantissa is a zero) and the first two digits are not 01.
  3. We must aim to get rid of the leading zeros from the mantissa to normalize the number.
  4. The mantissa is 0000110101 If we show the decimal place, the number is actually 0.000110101
  5. Moving the decimal point between the first 01 pair we come to starting from the left of the number, the number becomes 0000.110101
  6. Discard any excess zeros at the front of the number and add excess zeros to the end, to maintain the correct mantissa size. We now have 0.110101000
  7. This is now in the form we want. The first two digits are "01".
  8. Now we need to adjust the exponent. The exponent was 000100, or 4 in denary.
  9. To get 0.110101000 back to 0.000110101 we must move the decimal point 3 places to the left.
  10. We must therefore subtract 3 places from the current exponent.
  11. This means the new exponent is 000001
  12. Putting mantissa and exponent together, the normalized form of the number is 0110 1010 0000 0001

Normalized numbers must begin with 01 for positive numbers and 10 for negative numbers.

Converting Negative Numbers

Graphical user interface, text, application, email

Description automatically generated

Standard Form Representation

Timeline

Description automatically generated

Floating Point Numbers

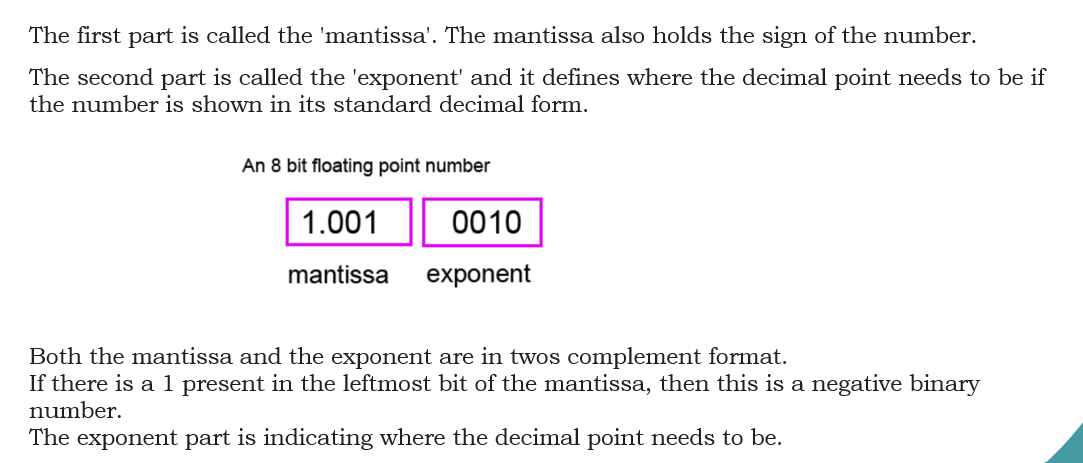
Real numbers are numbers that include fractions/values after the decimal point.

123.75 is a real number.

This type of number is also known as a **floating-point number**.

It is called floating point because effectively the location of the decimal/binary point moves

All floating-point numbers are stored by a computer system using a **mantissa** and an **exponent**.



Floating Point Conversion

Graphical user interface, text

Description automatically generated with medium confidence

Graphical user interface, text, application, email

Description automatically generated

Convert Denary to floating Point binaryGraphical user interface, text, application

Description automatically generated

Graphical user interface, text, application

Description automatically generated

Negative Numbers Conversion

A picture containing text

Description automatically generated

Chart

Description automatically generated