Number Systems

Introduction

A writing system for expressing numbers. Each number system defines a set of symbols that each represent a specific value.

Base (or radix)

Number of symbols defined by a number system.

Commonly used number systems

- Base 10 - 0 - 9
- Base 2 - $0, 1$
- Base 8 - 0 - 7
- Base 16 - 0 - 9, A - F

> ⚠ **Caution**
>
> These are required for s1:
>
> - Converting integers and floats between number systems
> - Addition, subtraction, multiplication, division in base 2
>
> But I don’t know how to include it in a easy-to-understand way. ⚫

One’s & Two’s Complement

One’s complement

The ones’ complement of a binary number is the value obtained by flipping all the bits in the binary representation of the number.

- If one’s complement of a is b, then one’s complement of b is a.
- Binary representation of $a + b$ will include all 1 s.

One’s complement system

In which negative numbers are represented by the inverse of the binary representations of their corresponding positive numbers. First bit denotes the sign of the number.

- Positive numbers are the denoted as basic binary numbers with 0 as the MSB.
- Negative values are denoted by the one’s complement of their absolute value.

For example, to find the one’s complement system representation of -7, one’s complement of 7 must be found. $7 = 0111_2$. One’s complement of -7 is 1000.

Two’s complement

In which negative numbers are represented using the MSB (sign bit).

If MSB is:

- 1: negative
- 0: positive

Positive numbers are represented as basic binary numbers with an additional 0 as the sign bit.

For example:

Following equation can be used to convert a number in two’s complement form to decimal.

$$b = -2^{n-1}b_{n-1} + \sum_{k=0}^{n-2} 2^k b_k$$

Steps

1. Starting with the absolute binary representation of the number
2. Add a leading 0 bit being a sign bit
3. Find the one’s complement: flip all bits (which effectively subtracts the value from -1)
4. Add 1, ignoring any overflows

Floating-point Representation

IEEE 754 standard.

2 types:

- single precision
- double precision

Single precision

Uses 32 bits.

- sign bit - 1 bit
- exponent - 8 bit
- mantissa - 23 bit

Sign bit

0 if positive or zero. 1 if negative.

Exponent

Exponent field range - $[0, 255]$. In this range $[1, 254]$ is defined for normal numbers. 0 and 255 are reserved for subnormal, infinite, signed zeros and NaN.

To support negative exponents, we subtract 127 (half of 254) from this range. $[-126, 127]$. This range is the representable range.

Mantissa

In scientific notation, the part that doesn’t contain the base and the power.

In binary scientific notation, there will always be exactly one 1 bit before the dot. So we don’t include that one.

> **Example**
>
> Take 31.3125.
>
> - In binary: 1111.0101_2
> - In binary scientific notation: $1.1110101_2 \times 2^3$
> - Add 127 to exponent: 130
> - Convert exponent to binary 10000010
> - Write the final result: $0\ 10000010\ 00000000000000001110101$
>
> Take 0.125.
>
> - In binary: -0.001_2
> - In binary scientific notation: $-1.0_2 \times 2^{-3}$
> - Add 127 to exponent: 124
> - Convert exponent to binary 01111100
> - Write the final result: $1\ 01111100\ 00000000000000000000000$

Double precision

Uses 64 bits.

- sign bit - 1 bit
- exponent - 11 bit
- mantissa - 53 bit

Sign bit

0 if positive or zero. 1 if negative.

Exponent

Exponent field range - $[0, 2047]$. In this range $[1, 2046]$ is defined for normal numbers. 0 and 2047 are reserved for subnormal, infinite, signed zeros and NaN.

To support negative exponents, we subtract 1023 (half of 2046) from this range. $[-1022, 1023]$. This range is the representable range.

Mantissa

In scientific notation, the part that doesn’t contain the base and the power.

In binary scientific notation, there will always be exactly one 1 bit before the dot. So we don’t include that one.

> **Example**
>
> Take 31.3125.
>
> - In binary: 1111.0101_2
> - In binary scientific notation: $1.1110101_2 \times 2^3$
> - Add 1023 to exponent: 1026
> - Convert exponent to binary: 10000000010
> - Write the final result: $0\ 10000000010\ 0000000000000000000000000000000000000000000001110101$
>
> Take 0.125.
>
> - In binary: -0.001_2
> - In binary scientific notation: $-1.0_2 \times 2^{-}3$
> - Add 1023 to exponent: 1020
> - Convert exponent to binary: 1111111100
> - Write the final result: $1\ 1111111100\ 0000000000000000000000000000000000000000000000000000$

String Representation

A way of representing non-numerical data.

Commonly used encodings

ASCII

Abbreviation for American Standard Code for Information Interchange. Uses 7 bits for letter representation and a parity bit (MSB). Can represent latin alphabet, digits, punctuations, and control characters.

Major limitation in ASCII is it can’t support multiple languages.

Unicode

Uses 32 bits. Supports multiple languages and emojis. Characters are presented by code points. A code point is a integer (in base 16).

Data Structures & Algorithms

Data structures

Common data types that are useful in many different places.

Abstract Data Type

A data type that has well defined properties and operations but not implementation.

Examples

- Array - fixed-length, one-dimensional
- Set
- Stack - Last in; first out
- Queue - First in; first out
- Binary search tree

> **Note**
>
> Implementations of stacks, queues, and binary search trees are required in s1.

Algorithms

An algorithm is a finite set of instructions, used to solve a problem.

> **Note**
>
> In s1, only sorting algorithms are discussed.

Selection sort

Here is selection sort algorithm that sorts a list of numbers in-place:

```
def selection_sort(arr):
    for current_starting_index in range(len(arr)):
        smallest_index = current_starting_index
        for i in range(current_starting_index + 1, len(arr)):
            if arr[i] < arr[smallest_index]:
                smallest_index = i
        arr[smallest_index], arr[current_starting_index] = arr[current_starting_index], arr[smallest
```

Bubble sort

Here is bubble sort algorithm that sorts a list of numbers in-place:

```
def bubble_sort(arr: list[int | float]):
    sorted_index_count = 0
    while sorted_index_count < len(arr):
        for i in range(len(arr)-sorted_index_count-1):
            if arr[i] > arr[i+1]:
                arr[i], arr[i+1] = arr[i+1], arr[i]
        sorted_index_count += 1
```

Software Engineering

Software

Refers to all the related things that are required to make a software system work.

Includes:

- programs
- configuration files
- system and user documentation
- user support system
- bug fixes and updates

Software engineering

An engineering discipline that is concerned with all aspects of software production. From the initial stage of writing the requirements to maintaining it while being used.

Software process

Set of activities that are associated with the development of a software product.

Fundamental activities that are common to all types of software development processes:

- Specification - defining the software to be produced and the runtime constraints
- Development - design and development of the software
- Validation - testing phase to check if the software meets the specifications
- Evolution - software is modified to adapt to new specifications

Waterfall

All before-mentioned activities are done sequentially, as clear separate phases. One phase is completed before the next phase is started.

Iterative & incremental

System is developed in iteration. Smaller parts of the system is completed in each iteration, that includes:

- Small amount of requirements specification
- Design and development for the specification
- Validation for the developed parts

Component based

Existing components are combined to implement the system. Main concentration is on the integration of the components.

Quality of software

Can be measured using these aspects:

- Maintainability - how easy it is to making changes
- Dependability - how secure, reliable it is to failures or other unusual activities
- Efficiency - how efficiently hardware resources (such as memory, processor time, disk space) are used
- Usability - how easy it is to use the software from user’s perspective
- Robustness - how resilient it is to invalid inputs

Challenges in software engineering

- Complexity
    - Essential - inherent, difficult to overcome
    - Accidental - not inherent, can be overcome
- Conformity
- Changeability - expected to be changeable to greater extent
- Invisibility - not visualizable
- Can’t guarantee defect free software - no amount of testing can prove absence of defects