Assignment 1 : Binary Encoding

Coding Documentation

Due Date: September 12, 2025

Introduction

The goal of this assignment is to write a program that converts a decimal number string into IEEE-754 64-bit binary. To complete this task, I wrote a Python script that takes a decimal number as input and outputs its binary representation in IEEE-754 64-bit format.

Cases Handled

The following cases are considered:

- Invalid inputs (non-numeric strings)
- Infinity (+ / -)
- NaN
- Zero (+ / -)
- Subnormal numbers (+ / -)
- Decimals that are too large to be represented in IEEE-754 64-bit format
- General case for valid decimal numbers within the representable range.

Algorithm

The algorithm follows these general steps:

1. Parse the input string and validate that it is a decimal number.
2. Handle special cases (e.g., NaN, Infinity, Zero).
3. Check the magnitude of the number to determine if it is subnormal or too large for IEEE-754 64-bit representation.
4. Convert the valid decimal number to its binary representation.
    (a) Determine the sign bit (0 for positive, 1 for negative).

(b) Separate the number into its integer and fractional parts.

(c) Convert both parts to binary, rounding/appending to keep 53 significant bits for the mantissa (pre-normalization) as needed.

   i. Convert the integer part to binary.

      A. Repeatedly divide the integer by 2, recording the remainders until the integer is < 1.

      B. Remove trailing zeros from the list of remainders.

      C. Return reversed list of remainders as binary representation.

      Note: if integer part is 0, an empty list is returned.

   ii. **If** integer part has at least than 53 bits, truncate and round to 53 bits. Also find the exponent.

      A. **If** integer binary length = 53, no rounding is needed. Set exponent = 52.

      B. **Else** if integer binary length $\geq$ 54, set exponent = length - 1. Truncate to 54 bits and round the last bit. If a carry occurs, prepend '1' to the integer binary, remove the last bit, and increment the exponent by 1. Repeat rounding if necessary.

   iii. **Else** convert the fractional part to binary until the total length (integer + fraction) is 53 significant bits, rounding if necessary.

      A. Calculate the number of significant bits in the mantissa already used by the integer part.

      B. Determine maximum number of significant bits remaining for the fractional part (54 - integer bits).

      C. Repeatedly multiply the fractional part by 2, recording the integer part (0 or 1) until the fractional part meets tolerance or max significant bits are reached.

      D. **If** the fractional part is not 0 after reaching max bits, round the binary representation. Rounding the integer part may be necessary if rounding the fractional part causes a carry. Repeat rounding until no carry is needed.

      E. Return a tuple of the binary representation of the fraction, and the (possibly rounded) binary representation of the integer part.

(d) **If** fractional binary representation is not empty, find the exponent.

   i. **If** integer part is not empty, exponent = (length of integer binary) - 1.

ii. Else exponent = - (index of first '1' in fractional binary + 1).

(e) Calculate the biased exponent and its binary representation.

i. Biased exponent = exponent + 1023.

ii. Convert biased exponent to binary using the same method as integer binary conversion.

iii. **If** biased exponent binary is less than 11 bits, prepend with leading zeros.

Note: Biased exponent should not be more than 11 bits, as we have already checked if the magnitude of the number is within representable range.

(f) Combine the integer and fractional binary parts to form the mantissa.

(g) Normalize the mantissa to 52 bits.

i. We already know there are 53 significant bits in total (integer + fraction) before normalization. The normalized mantissa is a splice that starts 1 index after the first '1' bit (the leading '1').

(h) Combine the sign, exponent, and mantissa to form the final binary representation.

(i) Return the final binary representation as a string formatted as "sign exponent mantissa".

Note: Extra exception handling occurs in steps 4b and 4d(ii) to ensure that conversion is possible within the default precision (28 significant digits) of the Decimal object. Due to this limitation, the actual range of representable numbers in this program is roughly 1e-14 to 1e27.

Function Documentation

Main Function

`decimal_to_binary(num_string)`

Description:

This function takes a decimal (string) and returns the number in IEEE 754 64-bit binary form (string). The returned string has spaces to indicate the sign, exponent, and fraction.

Parameters:

`num_string` : `(string)` The decimal provided from user input.

Returns:

`string` : `num_string` converted to IEEE 754 64-bit binary. Formatted as “sign exponent fraction”.

Raises:

`TypeError`: If the input is not a valid decimal number.

`ValueError`: If the input is too large for IEEE 754 64-bit conversion, or if the Decimal object cannot represent the input with its default precision.

Helper Functions

`integer_to_binary(integer)`

Description:

This function takes an integer and returns its binary representation as a list of chars. Returns an empty list if the integer entered is any form of 0.

Parameters:

`integer` : (`int` or `float`) The integer to be converted into binary.

Returns:

`list of string`: The binary representation of the integer as an ordered list. Each index contains a ‘0‘ or ‘1‘.

`fraction_to_binary(fraction, integer_binary)`

Description:

This function takes the fractional portion of a number (as a Decimal object) and returns a tuple containing:

1. The binary representation of the fraction as a list of chars.
2. The (possibly rounded) binary representation of the integer part.

The length of the binary fraction list is determined by the number of bits remaining in the mantissa after accounting for the integer part.

Parameters:

`fraction` : (`Decimal`) The fractional portion of the number to be converted into binary.

`integer_binary` : `(list of string)` The binary representation of the integer portion as a list of '0' and '1' chars. Used to calculate the remaining bits in the mantissa before normalization.

Returns:

`tuple (list of string, list of string) :`

- `tuple[0]: list of string`

  Contains the binary representation of the fraction as an ordered list of '0' and '1' chars.

- `tuple[1]: list of string`

  Contains either a rounded version of `integer_binary` (if rounding occurs) or a copy of the original `integer_binary`.

`round_up(binary)`

Description:

This function is a helper for `fraction_to_binary()`.

It rounds up the given binary number (provided as a list of '0' and '1' chars), starting from the least significant bit, *without prepending a new bit*.

It returns a tuple containing:

1. The rounded binary number as a list of '0' and '1' chars.
2. A boolean indicating whether a '1' needs to be appended to the beginning from rounding.

Parameters:

`binary` : `(list of string)` The binary representation of a number as an ordered list of '0' and '1' chars.

We assume that if the number is a(n):

- Integer: '1' to be rounded remains in the binary representation as the last digit.
- Fraction: The last digit '1' to be rounded is already removed from binary representation.

Returns:

`tuple (list of string, bool) :`

- `list of string`: The rounded binary number as a list of '0' and '1' chars.
- `bool`: Indicates if a '1' needs to be appended to the beginning due to rounding.
