
De La Salle University - Manila

# IEEE-754 Decimal-32 Floating-Point Converter

CSARCH2
S13

In Partial Fulfillment of the Course
Introduction to Computer Organization and Architecture
(CSARCH2)

Group 7

**Accomplished by:**
Arteche, Pryne Melton H.
Bautista, Samantha Michaela O.
De Vera, Jonathan L.
Miguel, Angela Dominique C.

Submitted to:
Sir Ronald Pascual

Term 3, A.Y. 2023-2024
July 31, 2024

# 1. Introduction

## 1.1 Overview

The IEEE-754 Decimal-32 floating-point converter is a designed program using Python and Node Js to convert decimal numbers into their IEEE-754 Decimal-32 floating-point representation. The program aims to provide an accurate and efficient tool in decimal floating-point arithmetic to be used in various applications.

## 1.2 Objectives

To provide an accurate conversion of decimal numbers to the IEEE-754 Decimal-32 floating-point representation that correctly supports and manages the handling of special values, such as positive and negative infinity and NaN input. The program also provides options for the different rounding methods, including truncation, rounding up, rounding down, ties to even, and ties away from zero. In addition, to have an optimized process of computing for efficiency.

# 2. Program Description

## 2.1 Functionality

- *Sign Extraction:*
  - Determines the sign bit (0 for positive and 1 for negative).
  - Converts the value to its absolute form.
- *Normalization:*
  - Normalizes the number to have 7 whole digits.
  - Adjusts the exponent accordingly.
- *Exponent Biasing:*
  - Adds the bias value (101) to the exponent to fit the Decimal32 format.
- *Combination Field Calculation:*
  - Determines the most significant bits (MSB) of the value.
  - Forms the combination field based on the MSB and the biased exponent.
  - Sign Extraction:
  - Determines the sign bit (0 for positive and 1 for negative).
  - Converts the value to its absolute form.
- *Rounding:*
  - Applies the specified rounding method to the significand.
    - Truncation
    - Round up
    - Round down
    - Round to Nearest, Ties away from Zero
    - Round to Nearest, Ties to Even

- *Densely Packed BCD(Binary-Coded Decimal):*
  - Computes the decimal significand field into BCD
- *Binary Conversion:*
  - Converts the sign bit, combination field, exponent, and significand into binary format.
- *Combining Parts:*
  - Combines all parts (sign bit, combination field, exponent, and significand) into a single binary string.
- *Hexadecimal Conversion:*
  - Converts the combined binary string into its hexadecimal representation.
- *NaN (Not a Number):*
  - Returns '011111 10000 00000000000000000000' in binary and '7c000000' in hexadecimal.
- *Positive Infinity:*
  - Returns '011110 00000 00000000000000000000' in binary and '78000000' in hexadecimal.
- *Negative Infinity:*
  - Returns '111110 00000 00000000000000000000' in binary and 'f8000000' in hexadecimal.
- *Zero:*
  - Returns '000000 00000 00000000000000000000' in binary and '00000000' in hexadecimal.

**2.2 Inputs and Outputs**

*Inputs:*

- A decimal number (positive or negative, including special cases like NaN and infinity).
- An exponent to scale the decimal number.
- A rounding method (truncate, up, down,ties to even, ties away from zero).

*Outputs:*

- Sign bit, combination bit, exponent bit, and significand bits in binary.
- Combined binary output.
- Hexadecimal representation of the IEEE 754 Decimal32 format.

## 3. Testing and Validation

The *test_decimal_ieee.py* script was executed with *pytest*, which systematically tests the *decimal_to_ieee.py* program to ensure accurate conversion of decimal numbers to IEEE 754 Decimal32 format. The function encompasses a broad spectrum of test cases, including various decimal inputs, **different rounding methods**, and special values like **NaN** and **infinity**. Each test case defines expected output values such as sign bit, combination bit, exponent, significand, and their binary and hexadecimal representations.

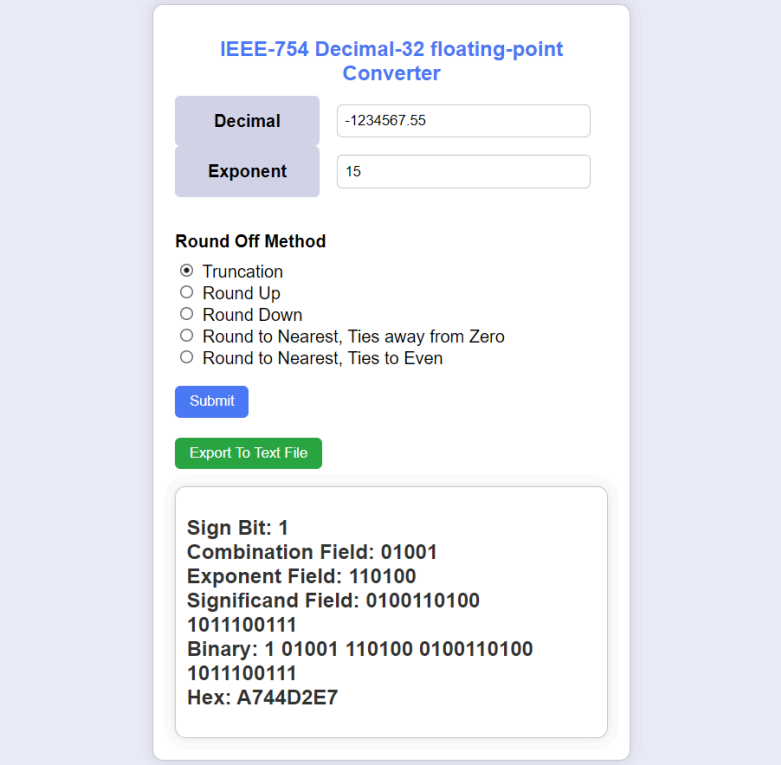
The *run_test* helper function orchestrates the testing by running the script with the specified inputs, capturing and parsing its JSON output, and then comparing it against the expected results. It includes error handling to manage script execution failures and JSON parsing issues. This structured testing process, facilitated by *pytest*, validates that the script correctly handles diverse scenarios and edge cases, ensuring its robustness and precision in converting decimal numbers to the IEEE 754 Decimal32 format.

### 3.1 Test Cases

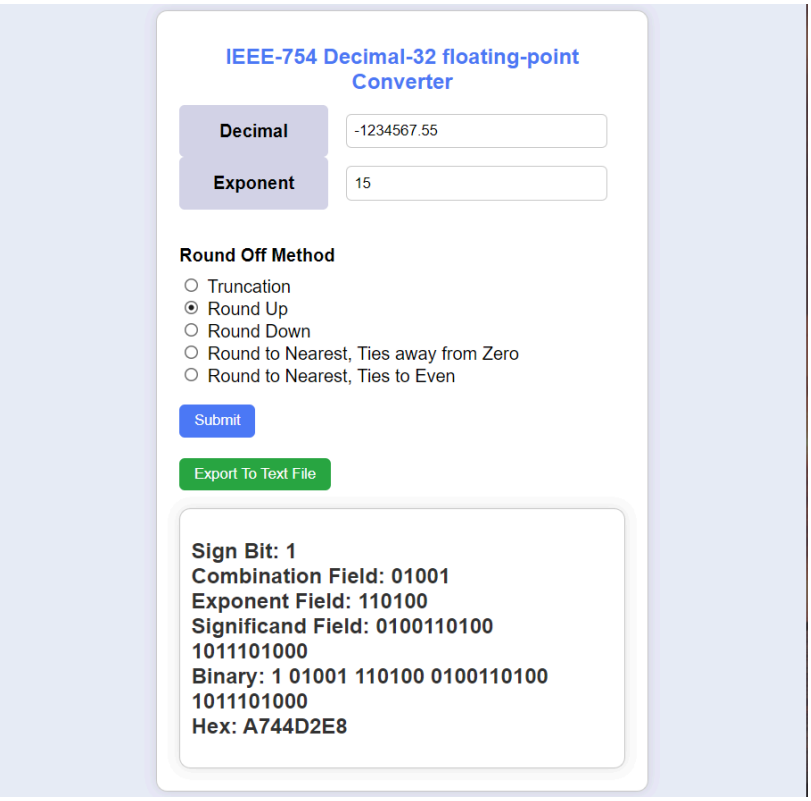
*Test cases included:*

- Positive and negative numbers.
- Zero, NaN, and infinity.
- Various rounding methods and different exponents.

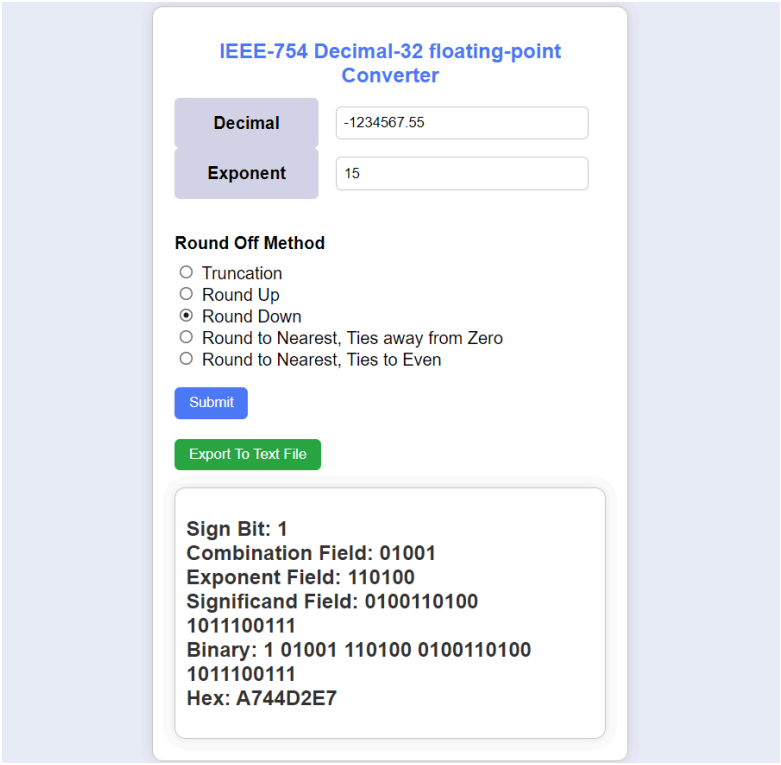
Denormalized (Negative) - Truncation



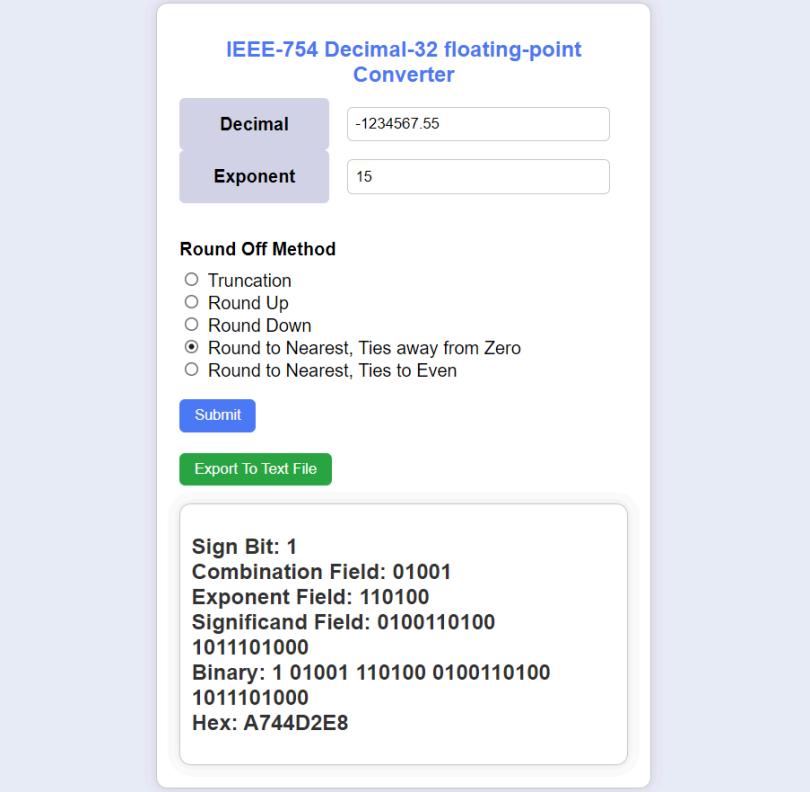
Denormalized (Negative) - Round Up



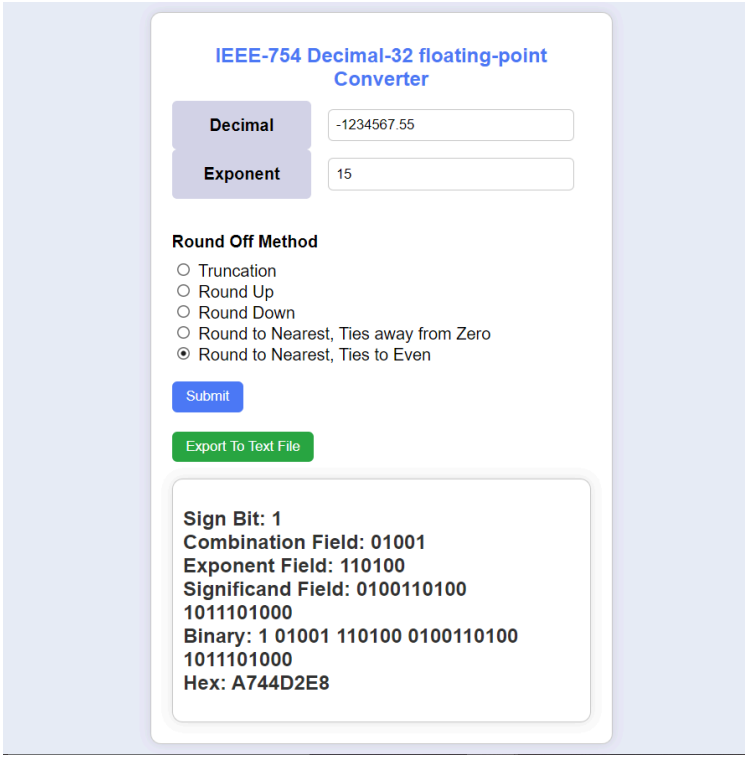

Denormalized (Negative) - Round Down



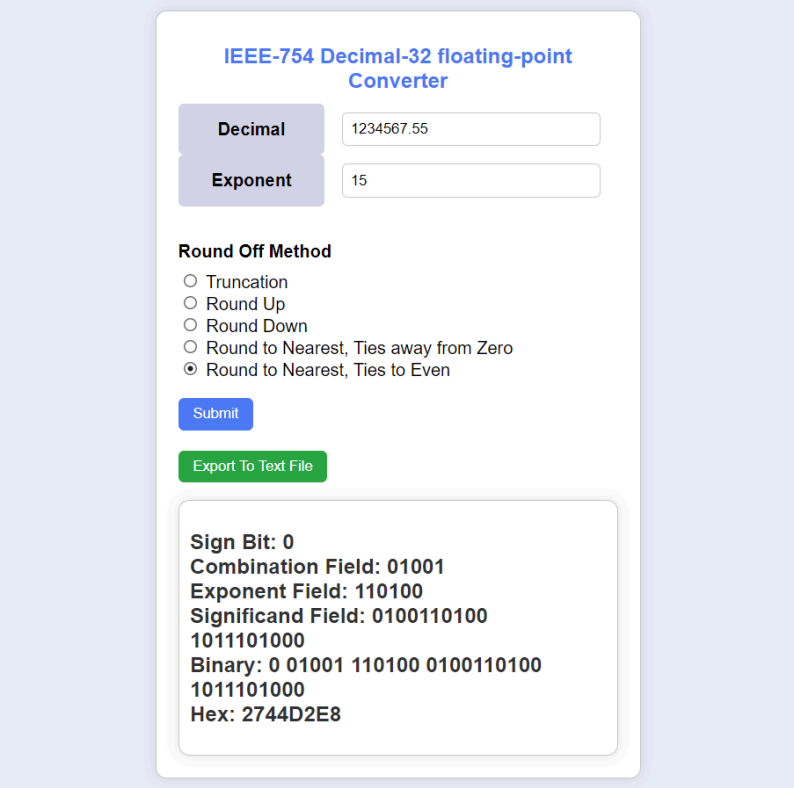
Denormalized (Negative) - Round to Nearest, Ties away form zero



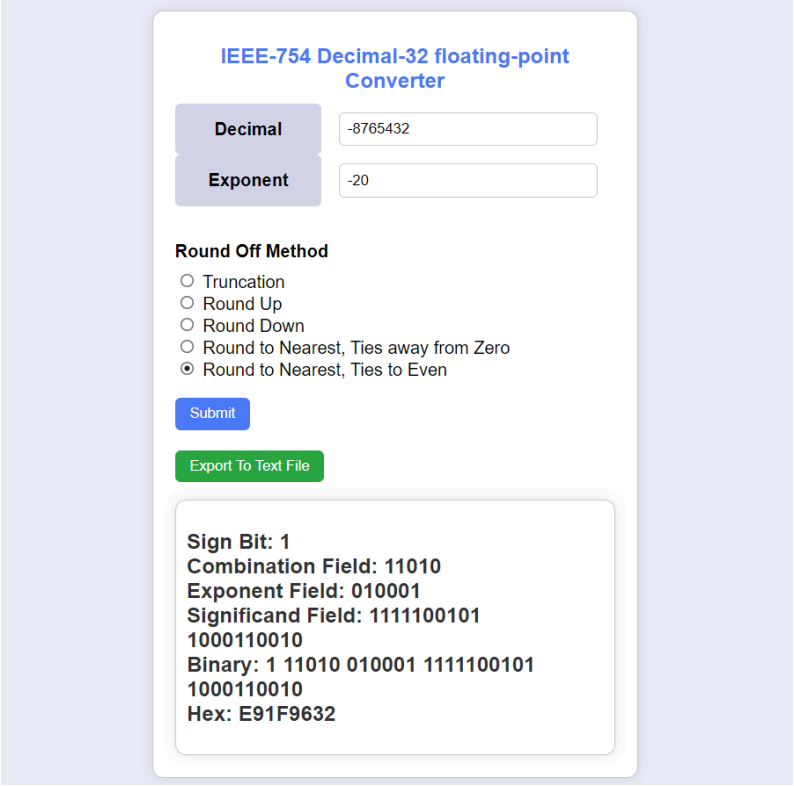

Denormalized (Negative) - Round to Nearest, Ties to Even



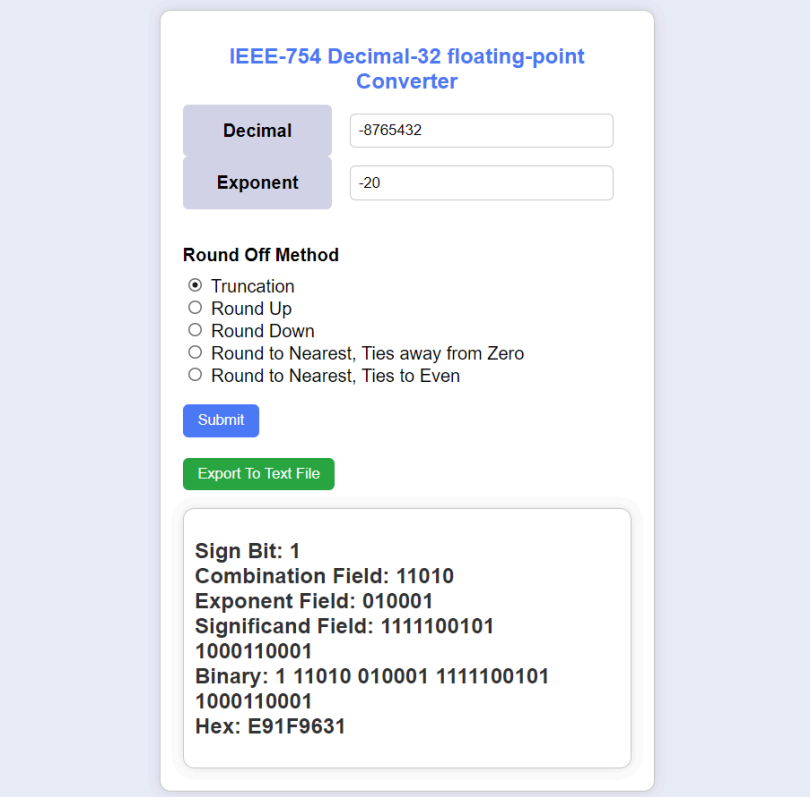
Denormalized (Positive) - Round to Nearest, Ties to Even



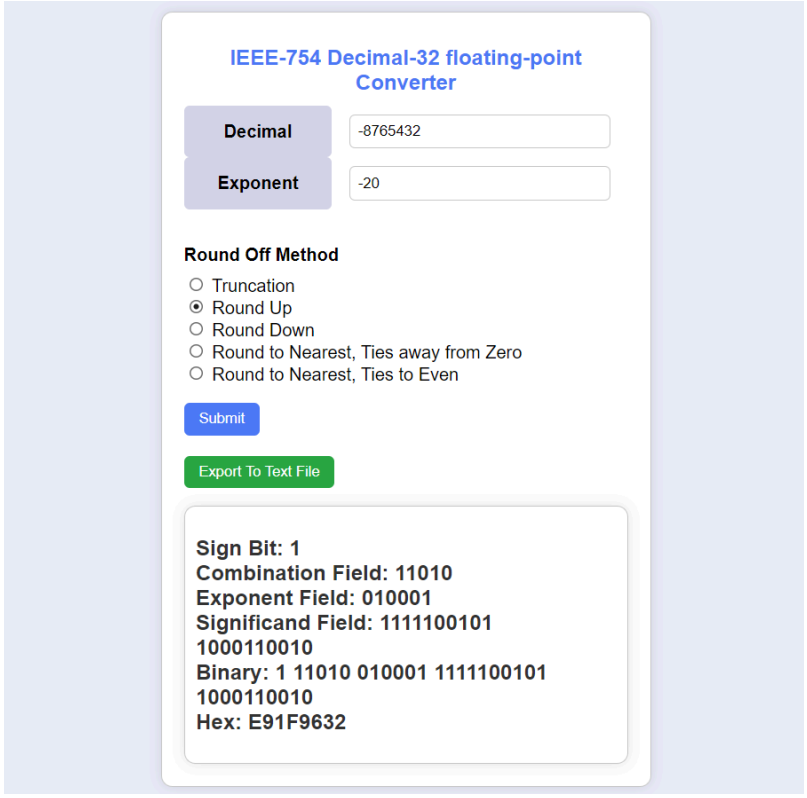

Normalized (Negative) - Round to Nearest, Ties to Even



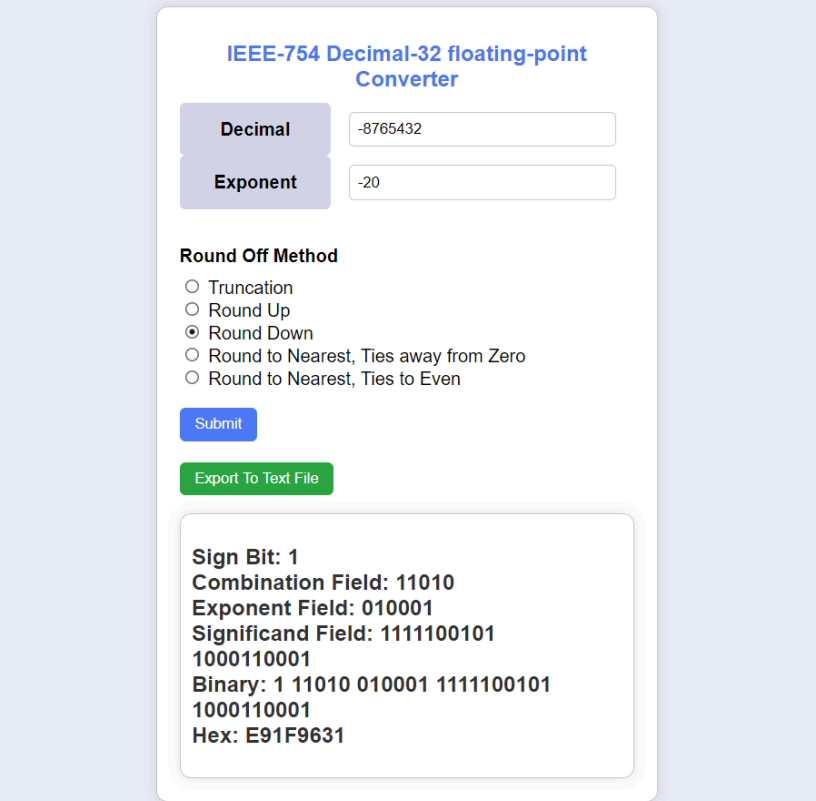
Normalized (Negative) - Truncation



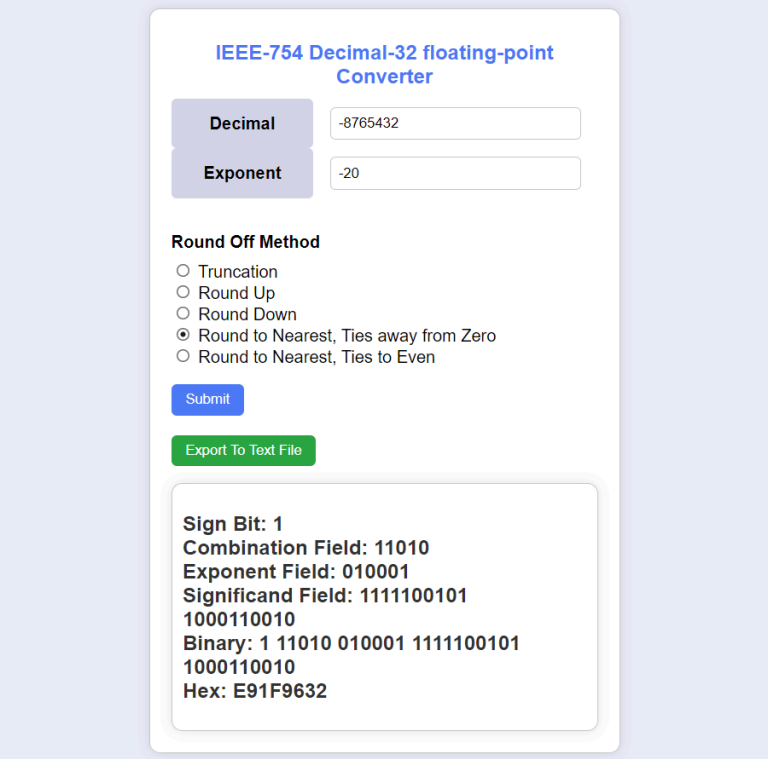

Normalized (Negative) - Round Up



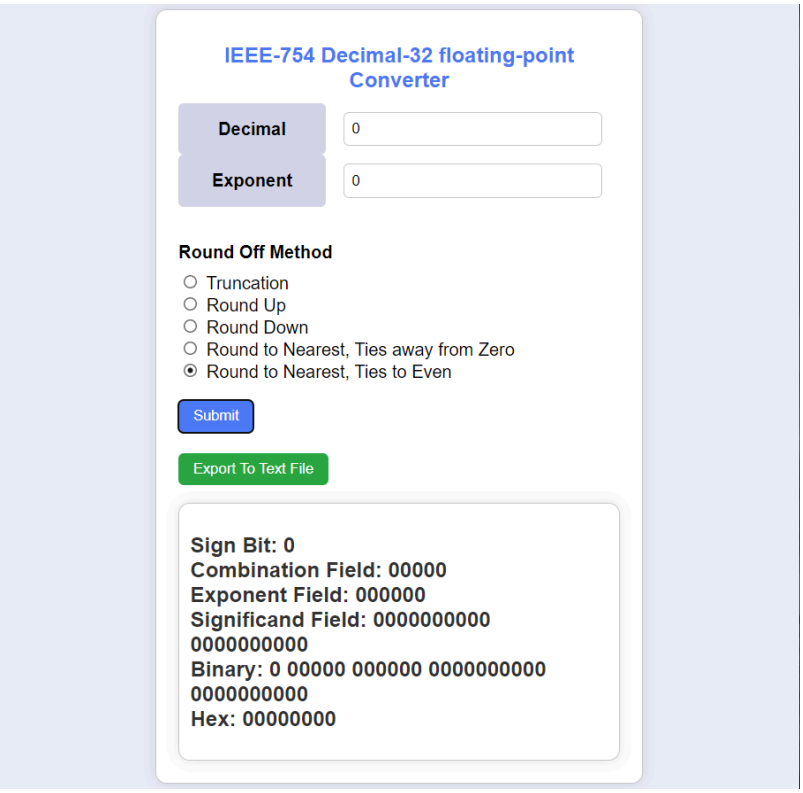
Normalized (Negative) - Round Down



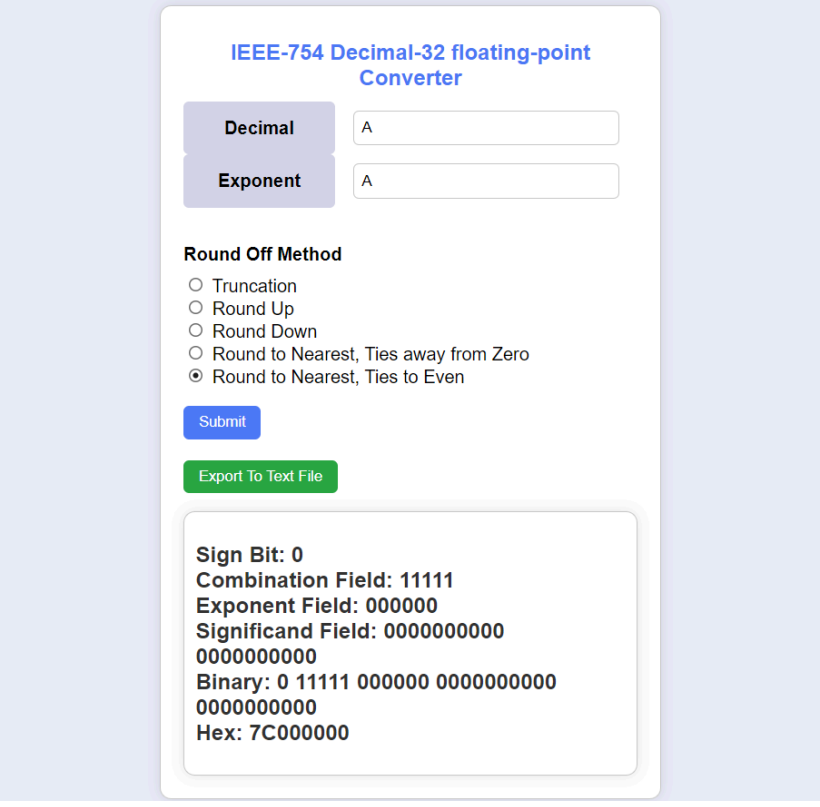

Normalized (Negative) - Round to Nearest, Ties away from Zero



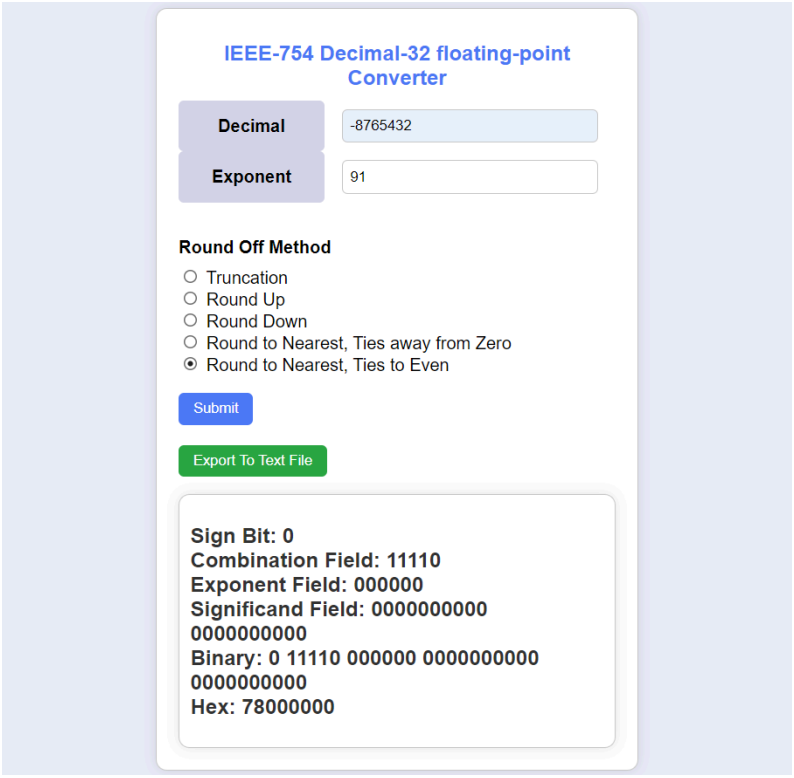
Zero Special Case



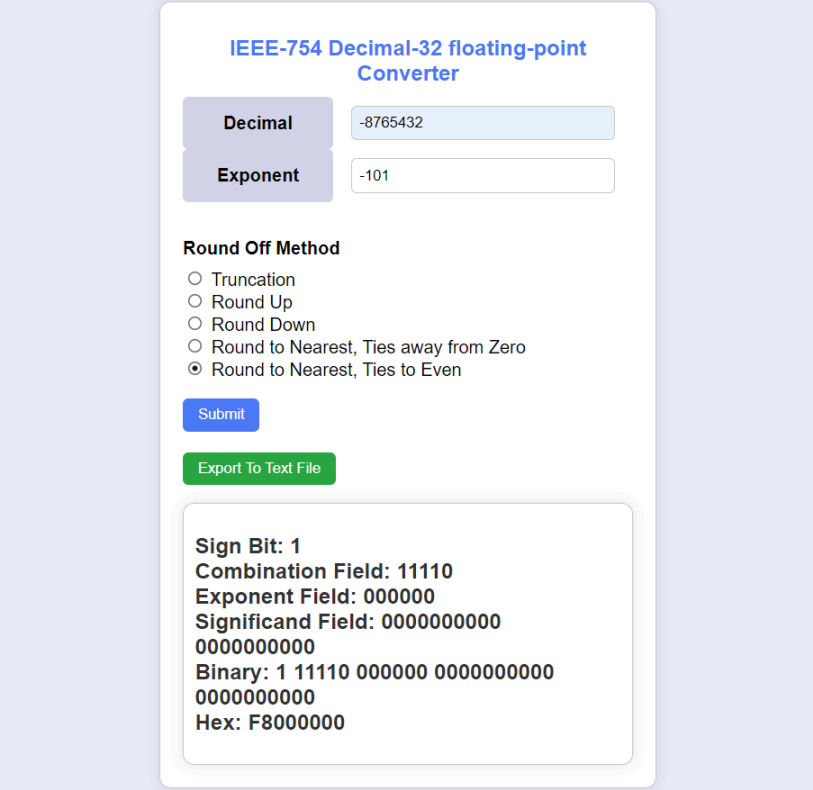

NaN Special Case



Positive Infinity Special Case




Negative Infinity Special Case

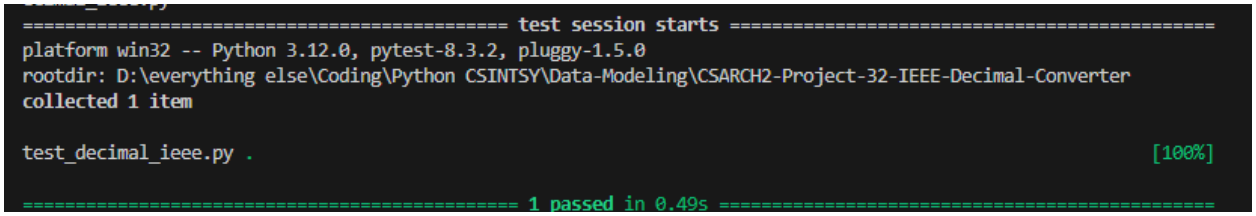


# Raw Python Test Case Inputs and Outputs using pytest

```
# positive decimal
'decimal': '1234567.55',
'exponent': 15,
'rounding_method': 'nearesteven',
'expected': {
    'sign_bit': '0',
    'combination_bit': '01001',
    'exponent_bit': '110100',
    'first_significand_bit': '0100110100',
    'second_significand_bit': '1011101000',
    'significand_bit': '01001101001011101000',
    'binary_output': '0 01001 110100 0100110100 1011101000',
    'hex_output': '2744d2e8'
}


# negative decimal
'decimal': '-1234567.55',
'exponent': 15,
'rounding_method': 'nearesteven',
'expected': {
    'sign_bit': '1',
    'combination_bit': '01001',
    'exponent_bit': '110100',
    'first_significand_bit': '0100110100',
    'second_significand_bit': '1011101000',
    'significand_bit': '01001101001011101000',
    'binary_output': '1 01001 110100 0100110100 1011101000',
    'hex_output': 'a744d2e8'
}


# negative exponent - also nearest even
'decimal': '-8765432',
'exponent': -20,
'rounding_method': 'nearesteven',
'expected': {
    'sign_bit': '1',
    'combination_bit': '11010',
    'exponent_bit': '010001',
    'first_significand_bit': '1111100101',
    'second_significand_bit': '1000110010',
    'significand_bit': '11111001011000110010',
    'binary_output': '1 11010 010001 1111100101 1000110010',
    'hex_output': 'e91f9632'
}


# negative exponent - truncation
'decimal': '-8765432',
'exponent': -20,
'rounding_method': 'truncate',
'expected': {
    'sign_bit': '1',
    'combination_bit': '11010',
    'exponent_bit': '010001',
    'first_significand_bit': '1111100101',
    'second_significand_bit': '1000110001',
    'significand_bit': '11111001011000110001',
    'binary_output': '1 11010 010001 1111100101 1000110001',
    'hex_output': 'e91f9631'
}


# negative exponent - Round up
'decimal': '-8765432',
'exponent': -20,
'rounding_method': 'up',
'expected': {
    'sign_bit': '1',
    'combination_bit': '11010',
    'exponent_bit': '010001',
    'first_significand_bit': '1111100101',
    'second_significand_bit': '1000110010',
    'significand_bit': '11111001011000110010',
    'binary_output': '1 11010 010001 1111100101 1000110010',
    'hex_output': 'e91f9632'
}
```

```
# negative exponent - Round down
'decimal': '-8765432',
'exponent': -20,
'rounding_method': 'down',
'expected': {
    'sign_bit': '1',
    'combination_bit': '11010',
    'exponent_bit': '010001',
    'first_significand_bit': '1111100101',
    'second_significand_bit': '1000110001',
    'significand_bit': '11111001011000110001',
    'binary_output': '1 11010 010001 1111100101 1000110001',
    'hex_output': 'e91f9631'
}
{
# negative exponent - Nearest Zero
'decimal': '-8765432',
'exponent': -20,
'rounding_method': 'nearestzero',
'expected': {
    'sign_bit': '1',
    'combination_bit': '11010',
    'exponent_bit': '010001',
    'first_significand_bit': '1111100101',
    'second_significand_bit': '1000110010',
    'significand_bit': '11111001011000110010',
    'binary_output': '1 11010 010001 1111100101 1000110010',
    'hex_output': 'e91f9632'
}


# zero
'decimal': '0',
'exponent': 0,
'rounding_method': 'truncate',
'expected': {
    'sign_bit': '0',
    'combination_bit': '00000',
    'exponent_bit': '000000',
    'first_significand_bit': '0000000000',
    'second_significand_bit': '0000000000',
    'significand_bit': '00000000000000000000',
    'binary_output': '0 00000 000000 0000000000 0000000000',
    'hex_output': '00000000'
}


# special cases - NaN
'decimal': 'NaN',
'exponent': 0,
'rounding_method': 'nearesteven',
'expected': {
    'sign_bit': '0',
    'combination_bit': '11111',
    'exponent_bit': '000000',
    'first_significand_bit': '0000000000',
    'second_significand_bit': '0000000000',
    'significand_bit': '00000000000000000000',
    'binary_output': '0 11111 000000 0000000000 0000000000',
    'hex_output': '7c000000'
}
{
#special cases - +Infinity
'decimal': '-8765432',
'exponent': 91,
'rounding_method': 'nearesteven',
'expected': {
    'sign_bit': '0',
    'combination_bit': '11110',
    'exponent_bit': '000000',
    'first_significand_bit': '0000000000',
    'second_significand_bit': '0000000000',
    'significand_bit': '00000000000000000000',
    'binary_output': '0 11110 000000 0000000000 0000000000',
    'hex_output': '78000000'
}
```

```
# special cases - -Infinity
'decimal': '-8765432',
'exponent': -101,
'rounding_method': 'truncate',
'expected': {
'sign_bit': '1',
'combination_bit': '11110',
'exponent_bit': '000000',
'first_significand_bit': '0000000000',
'second_significand_bit': '0000000000',
'significand_bit': '00000000000000000000',
'binary_output': '1 11110 000000 0000000000 0000000000',
'hex_output': 'f8000000'
}
```




*Program passing the test script*

## 4. Conclusion

The IEEE-754 Decimal-32 floating-point converter meets its design objectives by accurately converting decimal numbers to IEEE-754 Decimal-32 format, managing special values, and implementing multiple rounding methods. Testing confirmed the program's effectiveness and reliability, demonstrating its capability to handle both standard and special cases efficiently. The final program was developed using a Node.js-based tech stack with Python scripts for conversion, ensuring both accuracy and performance.