CSARCH2 S15 Group 4
Genuino, Jose Mari Victorio
Ong, Nicole Daphne
Teves, Hannah Juliet
Tighe, Kaitlyn Patricia
Villavicencio, Josh Dane

Simulation Project Analysis Writeup

The simulation project assigned to our group was building a decimal-128 floating converter that displays the output for the binary and hexadecimal equivalents. The project was executed through a web-based app using JS, CSS, and HTML. The group successfully applied the principles and concepts learned in CSARCH2 to coding practices, demonstrating proficiency in formulating logic within programming and gaining insights into the constraints inherent in the selected programming languages. Understanding how to implement the converter, rounding methods, and special cases was all a part of the process of building the application.

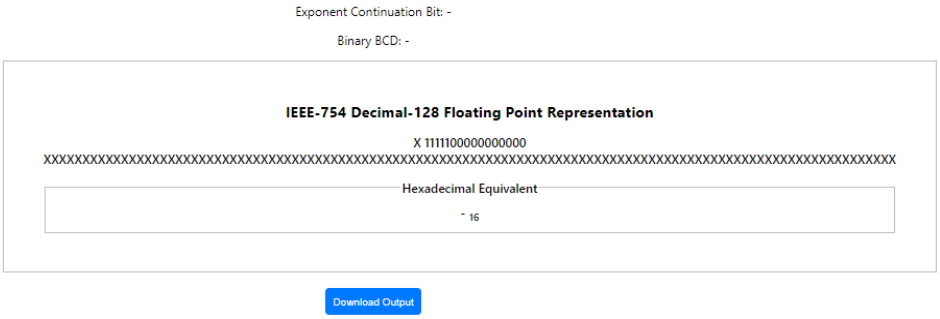
The web app is deployed using Vercel and can be accessed through this link:
https://csarch-2-s15-group4-sp.vercel.app/

The public repository can be accessed through this link:
github.com/kaitiepiee/CSARCH2-S15-Group4-SP

Problems and Solutions

1. **NaN Special Case**
   The problem we encountered was that we didn't know how to put NaN cases into the number field on our website. We had no way of inputting values that are considered to be NaN.

2. **Error when submitting empty fields**
   The garbage values displayed after clicking on the convert button do not clear up. It requires refreshing the website to remove it.



Fix

```
$(".convert-button").click(function() {
    var decimalInput = document.getElementById('decimalInput');
    var exponentInput = document.getElementById('exponentInput');

    if(decimalInput.value === '' || exponentInput.value === '') {
        alert('All fields are required!');
        return false;
    }
    convert();
});
```

It checks if the textboxes are empty. If either one of the text boxes is empty, it prompts an alert stating, “All fields are required.” Only if both fields are filled will the conversion work.

3. **Issues with handling numerical values**
   Due to limitations with JavaScript, too large numeric inputs become not what is expected due to precision issues. To solve this, we handle inputs as strings.

```
decimalInput: 1.23456 exponentInput: 0
decimalInput: 12.345600000000001 exponentInput: -1
decimalInput: 123.45600000000002 exponentInput: -2
decimalInput: 1234.5600000000002 exponentInput: -3
decimalInput: 12345.600000000002 exponentInput: -4
decimalInput: 123456.00000000003 exponentInput: -5
decimalInput: 1234560.0000000002 exponentInput: -6
decimalInput: 12345600.000000002 exponentInput: -7
decimalInput: 123456000.00000001 exponentInput: -8
decimalInput: 1234560000.0000002 exponentInput: -9
decimalInput: 12345600000.000002 exponentInput: -10
decimalInput: 123456000000.00002 exponentInput: -11
decimalInput: 1234560000000.0002 exponentInput: -12
decimalInput: 12345600000000.002 exponentInput: -13
decimalInput: 123456000000000.02 exponentInput: -14
decimalInput: 1234560000000000.2 exponentInput: -15
```