Presented to the College of Computer Studies

De La Salle University - Manila

Term 2, A.Y. 2023-2024

In partial fulfillment of the course

CSARCH2

Analysis Writeup

Cuales, Bianca Mari
Cuya, Vhonne Luigi
Jaime, Luis Rafayel
Pascual, Isaiah Sam
Reynado, Alyza

S15

Project Overview

The IEEE-754 Binary-32 floating-point converter is a tool designed to convert floating-point numbers represented in binary and decimal formats into their corresponding IEEE-754 Binary-32 format.

Input

1. **Binary Mantissa and Base - 2 Input** : Users can input a binary mantissa along with its base - 2 exponent (e.g., -101.01 x 2^5), representing a floating - point number.
2. **Decimal and Base - 10 Input** : Users can input a decimal number along with its base - 10 exponent (e.g., +25.5 x 10^3), representing a floating - point number.

Output

1. **Hexadecimal Equivalent** : The converter displays the hexadecimal equivalent of the IEEE-754 Binary-32 representation.

Methodology

The group decided to use JavaScript, React, and HTML for several reasons. JavaScript is a widely adopted programming language known for its versatility and ability to handle complex calculations, making it ideal for implementing the conversion logic required for the IEEE-754 Binary-32 floating point converter. React, a JavaScript library, was chosen for building the user interface (UI) components due to its component-based architecture, which promotes modularity, reusability, and efficient state management through features like hooks (e.g., useState). This choice allowed the team to create a dynamic and interactive UI that enhances the user experience. Additionally, HTML was used in conjunction with React to structure the UI elements and facilitate seamless integration of UI components within the web application. Overall, the combination of JavaScript, React, and HTML provided a robust foundation for developing a functional, user-friendly, and efficient IEEE-754 Binary-32 floating point converter.

Problems Encountered

One of the primary challenges the group encountered was translating the IEEE-754 Binary-32 conversion algorithm into code. To address this, we broke down the conversion algorithm into smaller, manageable steps. Next, we created pseudocode and flowcharts to outline the step-by-step process of converting a decimal or binary number to IEEE-754 Binary-32 format. Each step in the pseudocode represents a specific function or module in our code. We then implemented these steps one by one in JavaScript, ensuring that each function or module performed its designated task accurately. This approach allowed for incremental development and testing, with each part of the conversion process being validated independently. We conducted testing with various test cases to ensure accuracy and correctness at each stage of the algorithm. Refactoring and optimization were also performed iteratively, improving the code's readability, maintainability, and performance as our implementation progressed. This systematic breakdown and step-by-step implementation helped us overcome the complexity of the conversion algorithm and ensured a robust and accurate IEEE-754 Binary-32 floating point converter.

Deploying our React.js application was a challenge, but we resolved it by choosing Vercel as our hosting platform. We opted for Vercel due to its seamless integration with React.js, easy deployment process, and robust hosting capabilities. Setting up the deployment involved configuring the build process, deployment settings, and implementing continuous deployment using Vercel's features. Testing post-deployment ensured compatibility and responsiveness. Overall, Vercel streamlined our deployment workflow and facilitated the successful hosting of our application.

Effective group coordination was crucial for our project's success, and we encountered challenges such as communication gaps and task management issues. To improve coordination, we adopted several strategies. We utilized communication platforms such as Discord and Messenger for real-time discussions, updates, and file sharing. These platforms allowed us to maintain open communication channels, share project-related information, and collaborate efficiently. For code management and collaboration, we used GitHub as our version control platform. GitHub enabled us to work collaboratively on code, track changes, manage issues, and ensure version control integrity. Regular team meetings were scheduled to discuss project progress, challenges, upcoming tasks, and resource allocation. Clear roles and responsibilities were defined for each team member, ensuring transparency, accountability, and efficient collaboration throughout the project lifecycle. These collaborative tools and practices helped us overcome communication barriers, enhance productivity, and achieve our project goals effectively.

Throughout our project, one of the challenges we faced was the inability to generate a text file for the output, which unfortunately remained unsolved. Despite considering various approaches, including utilizing the Node.js File System (fs) module and implementing a specific function for file writing, we were not able to implement this feature within the project's scope. The task of creating a text file for the output involved complexities that we couldn't address during the development phase. As a result, the option to save the converted data to a text file was not included in the final implementation of our IEEE-754 Binary-32 floating point converter. Future iterations of the project may explore additional solutions or enhancements to incorporate this functionality.