CSARCH2 Simulation Project S11 Group 8
Binary-128 Floating Point Converter

Timothy Joshua O. Tan
John Kieffer Recato Dy L.
Aldwin Bennett Liobing
Shem Matthew J. Salih
Nicole Kate T. Uy

Introduction

As viewed from the computer architecture approach and numerical computation, floating point forms a key to orderly storage and manipulation of real numbers. IEEE-754 provides a set of floating-point formats that are broad in range, and so we were tasked to make a Binary-128 floating-point converter, as a requirement for our CSARCH2 course.

Project Description

This project has its main objectives laid out: to devise a tool through which decimal numbers alter to their binary-128 floating representation and back again, supporting users in understanding and making good use of this exact format.

Our floating-point decimal converter to binary-128 is adaptive and developed to cope with binary and decimal input values; therefore, the user can input whichever they prefer. In the same context or regard, the calculator was developed in Python, which utilizes the Tkinter library in the development of its GUI design.

Again the application supports two primary modes of operation: binary input mode and decimal input mode. In this mode, in conjunction with the use of the application's binary input mode, the user would key in a 128-bit binary value adjoined to signify a Binary-128 floating point number. The converter would accept the binary input where the sign bit, exponent, and fraction bits are present in a typical "sign-exponent-fraction" format as shown. It accepts several binary input formats with or without a radix point and the exponent separator. The application would normalize input in binary form automatically so that it gets properly converted.

It is an input from a decimal number and changes it to the respective binary 128-bit floating-point format. Input should be anything that resembles a normal decimal or scientific notation. It will translate the given decimal into its binary counter-respective as given in the binary 128 specification, which also includes its sign, exponent, and fraction.

Our converter has some of the major features which include consideration of all special cases which are part of floating-point arithmetic. The special cases that are considered are zero, -0, Infinity, -Infinity, Denormalized Numbers, Quiet NaN, and Signaling NaN.

This is the core functionality of this converter. It has extensive error handling and input validation mechanisms included for the service to be robust and accurate. The application checks for the different types of input formats not, and error messages in the case of invalid inputs.

Further, the converter must pass specific testing with many test cases based on functional, edge, and boundary conditions. We based the correctness verification of the results of conversion on the IEEE-754 Binary-128 standard to make this converter enable the user to have a correct and reliable output.

Implementation Details

The developed Binary-128 Floating-Point Decimal Converter has been written and compiled using the Python programming language as its backtesting programming environment. The package used in writing the Graphical User Interface (GUI) is Tkinter, a library package with a strong combination that aids in building an interactive and user-friendly interface.

Main Function: This is the function where the main logic is implemented. Here, the input number specified is treated as a string, and due to the chosen mode (binary or decimal), the arithmetic calculations for the conversions are done respectively. This supports many sub-functions and helps us deal with a few specific aspects of the conversion process.

This will be responsible for dealing with the input in decimal and converting it to its binary representation. It makes sure that its decimal input is majorly in two formats: scientific notation and normal notation. The major of the function is to partition both the integer part and the polygenic part of the decimal number and hence convert them into their respective appropriate binary through arithmetic operations.

It means the `process_binary_input()` function has to deal with binary input in the sense that the function recognizes several kinds of binary input and then undertakes normalization on these for further processing. It will ensure that the binary input is in a standardized format before it's handed over to the main public contract logic of transformations.

It also contains the special case-handling functions: sign(), exponent(), and fraction(). These are implemented with the means to extract relevant components from binary values and produce the requisite computation that would yield the value of the sign, exponent, or fraction.

A diversified GUI components application is accomplished in a variety of frames and widgets for ease of operation and presentation. The main frame of the application contains several sub-frames that involve; the title, the mode-picker, the input window, the output window, and the button group. All the buttons are explicitly edited in reaction to the current state from which it is operated. The `button_click()` function efficiently handles the button click events; the clicked button determines the action for such clicked button. The function `clear()` clears the input and output windows, and the function `backspace()` removes the last character from the input. On pressing the "Equals" key, the equal() function is called. It gets the input from the input window,

normalizes the given input with respect to input mode, and then passes the input in string form to give the technology driver the normalized input to perform the conversion. Formatting conversions follow subsequently on the result and are displayed in the output window or output as text with the given button "Save result as .txt".

Testing

The testing of the supposed Binary-128 floating point decimal converter for correctness and reliability has all been done. Examples of such test validation include real-life cases and most other issues that might occur during the testing of the output of the converter.

Some of the instances that were input include the settings and units of the converter, in finding various input values put into the conclusion by running a set of selected test cases.

For each test case, the expected Binary-128 floating point representation was manually calculated or obtained from a trusted reference.

This, in cases, where the outputs of the converter were the same as shown in the example, it proves the fact that the converter converts correctly, or rather, it performs the conversion correctly. Further to ensure that the converter is correct, here is the following manual crafting light and interesting set of test cases covering some edge, corner, and boundary conditions in regard:

- Numbers close to the limits of the Binary-128 format
- Numbers with varying precisions and exponent values
- Numbers with leading or trailing zeros

In our test video demo, we go over all of the test cases that are used in our testing.

The converter successfully handled these test cases, producing the expected Binary-128 representations. To accomplish this, several tests were executed, ranging from the simple to the hardest ones, manually, validating whether the produced output is in line with expectations and toward producing accurate and reliable output. This indeed proved the fact that the converter had the strength to be able to work for different inputs and yield a reliable binary 128 representation.

Limitations and Future Improvements

The Binary-128 floating point decimal converter software seems highly reliable for use, if it was not for the few downsides in the present software and a few areas of improvement.

In the current realization, the usage of the graphical user interface is purely with the Tkinter library. Tkinter, by the definition point of view, is a set of wrapper functions written at its highest, in Python, at the very top of the TIX library; it's a simple, free, and easy-to-use library. But still, its graphic performance may not be necessary, and that is why a beautiful—for the modern sense—probably, in the future, it could be replaced with some other GUI framework or library for further visually appealing attractiveness and user experience.

Another limitation is the non-direct support for changing input and output to hex. The converter is working for input and output of both decimal and binary right now, but direct support for input and output of the values in hexadecimal would no more be less than a few conveniences added for those related to the context.

Some of the future improvements that can help build on the functionalities and usability of the converter may include:

1. Binary-128 to Decimal and Binary: In general, an idea for a generalized converter of the conversion of any input floating-point number under Binary-128 to its corresponding decimal and binary representation would have been cool, since it will increase the feature of the tool so that it attains more versatility in use. By that, a user can key in a Binary-128 number and then be able to get the respective converted decimal and binary in BSD, providing satisfactory conversion tools.
2. Batch processing: It gives an option to the user, whether one wants to work in a batch—that is, if a user wants to convert a big set of values together, that gives him the capability to add more than one number at a time for conversion.
3. Added detailed explanations, with educational material included in the converter. This would most likely add much more value and make it an educational tool due to such inclusion. For example, a short explanation of the format, the format it represents, the process of the conversion, and most importantly what each constituent in that format stands for (sign, exponent, fraction) would be a great eye-opener toward understanding.
4. Enabled with Performance Optimization: Please note that the following implementation, although it works pretty well for generic use, might be a little bit too roomy in respect to performance. For really huge numbers or heavy computations, further speed and better resource consumption might be achieved by looking over more efficient algorithms, or by integrating specialized libraries.
5. Expanded support for the platform is the development of the website version of the converter that users will be able to get from any browser on the platform. This way, the web interface will ease access to the converter.
6. Integration with Other Tools: Now, contemplating the possible integration that can be had from such a solution into encoders, such as scientific computing libraries, and data analysis tools, for that matter, it would naturally be rolled into all tasks of conversion and manipulation of Binary-128.

Overcoming these limitations by the following suggestions provided, the third improvement on Binary-128 Floating-Point Decimal Converter facilitates the tool coming up with a much more

strong, friendly user, and flexible result in case of computation demanded by a Binary-128 number computation.

Conclusion

The project on the binary-128 floating-point decimal converter was an orientation towards developing a feasible and effective tool in aid to handle changing computation from its ordinal to binary-128 floating-point representation. With the design sketched well and the implementation and testing done with exactness, along with it hitting on the target, the converter will effectively achieve its objective and materialize as an aidful source for students, researchers, and professionals doing studies, construction, and implementations about Binary-128 numbers.

The input value may be a decimal or a binary to be inserted by the user, and then he receives a respective Binary-128 representation. The application doesn't restrict the number input and receives the numbers from both varied notations with either signed or unsigned format, and then does the conversion based exactly upon the sign bit, exponent bits, and fraction bits corresponding to Binary-128.

Further possibilities for dealing with issues include making the application usable and effective, like making it pretty, enabling functionality, optimizing performance, and deploying it to the web. As a deduction, this Binary-128 Floating Point Decimal Converter project ensures adequate standards of success in obtaining the functional and useful solution of the desired operation in process conversion from the decimal to Binary-128. Working in the form of academic assistance, the users get a chance to learn every aspect of the Binary-128 format and how it can be put into application in various sectors. So, this converter makes the job of numerical computation easy, and with the conveniences of a friendly user interface, it can surely land into the settings to work most effectively upon the Binary-128 floating-point numbers.