Presented to the College of Computer Studies

De La Salle University - Manila

2nd Term, A.Y. 2023-2024

In partial fulfillment

of the course

In CSARCH2 (S12)

Analysis Writeup - Simulation Project

Submitted by:

Karl Andre F. Aquino

Bien Aaron C. Miranda

Edward James E. Tan

Giusippi Maria II D. Apa

Luis Miguel D. F. Rana

Submitted to:

Dr. Roger Luis T. Uy

March 23, 2024

SIMULATION PROJECT: ANALYSIS WRITEUP

Aquino, Karl Andre; Apa, Giusippi Maria II ; Miranda, Bien Aaron; Rana, Luis Miguel; Tan, Edward James

GIVEN TASK:

The group's task was to create an IEEE- 754 Binary-128 floating-point converter. The input would be a binary mantissa with a base-2 exponent value (i.e., 101.01x2^5) or a decimal and base-10 exponent value (65.0x10^3). In certain cases, there may be some values that cannot be converted naturally. The converter should also be able to consider these special cases, signaling if they're zero, NaN, denormalized, or positive/negative infinities.

The output of the converter should be composed of the following:

- Binary output with space between each section
- The hexadecimal equivalent of the binary output

Afterward, the converter should allow for the option of a text file output of these values as well.

PROBLEMS ENCOUNTERED:

During the development phase of the project, some obstacles hindered its progress. For the majority of operations found within the converter, string manipulation was utilized for convenience in terms of grouping and shifting the numbers for reading; however, the string limit of Python, the language used for this project, is *4300*, which was lacking for the required operations of the program. There was a solution for this problem, which was to utilize the *sys* library that Python possessed to increase this limit up to *20000*.

Decimal inputs, on the other hand, had to be first converted into binary, which caused the need for the group to account for large numbers and exponents, the max positive normal number allowed for IEEE-754 Binary-128 being $1.189731495357231765085759326628007016 2 \times 10^{4932}$.

Another obstacle that proved to be an issue was the default setting for float inputs in Python only handling up to *64* bits of data, which was far below the required *128* bits of data for the converter. By utilizing the decimal library that Python possessed, the precision of the accommodations was enhanced to be able to handle up to *999999* digits, which was an overestimation that the group settled on. Note that fully using all those digits is not recommended by the group, as it may cause performance issues during runtime, chief among them being program crashes or minutes-worth lag times.

One minor problem encountered was the algorithms to be made when it came to the shifting of digits on certain conditions. The solution to this was a rudimentary trial-and-error methodology before finally settling on all possible cases.

Finally, there were some problems regarding the identification of criteria on how to represent *NaN*, specifically *qNaN* and *sNaN*. For this problem, the group settled on these conditions:

- If the input included operations that would normally result in an indeterminate number such as $0/0$, this would result in the program outputting the representation for *qNaN*.

- If the input contains alphabetical letters, it will be considered *NaN*, and will then be represented as *sNaN*.

- If the input included operations that would result in a complex number such as logarithm of a negative number ($ln(-1)$) or square root of a negative number *(√-1)*, it would be represented as *qNaN*.

- If the input still has mathematical operators, it will be considered as an invalid binary or decimal input and therefore be represented as *sNaN*.