Presented to the College of Computer Studies

De La Salle University - Manila

Term 2, A.Y. 2023-2024

In partial fulfillment of the course

In CSARCH2 S11

Simulation Project

Group 3

Submitted by:

Co Chiong, Shawn F.

Donato, Adriel Joseph Yadao

Sales, Liam Miguel Vitug

Tai Ai, Raphael C.

Villanueva, Miguel R.

Submitted to:

Roger Luis Uy

March 23, 2024

I. Introduction

The group was assigned to create a program that works as a converter for IEE-754 Decimal-64 floating point. The idea behind the program was to create a runnable application that acquires user input from the user, needing them to fill up the decimal value and the exponent value which is always used under Base-10. The program would then show the binary representation of the input and splitting it up to the different parts that form the whole 64 bit representation such as the sign bit, combination field, exponent continuation, and coefficient continuation. The output also provides the 16 digit hexadecimal value of the representation along with the option to write the input along with the output onto a file for the user.

II. General Process

The user is asked to input a decimal value for the significand and an integer value for the exponent and the rounding method they would like to use which includes, Truncate, Ceiling (Round Up), Floor (Round Down), and RTN-TE. After inputting, the values of the decimal and the integer are stored in different variables and undergo the normalization process. The decimal portion first goes through the process of checking the value of the sign-bit thus the function check_sign() is called. If the decimal value is less than 0 then the sign_bit value is '1' and else it is '0'.

After that process is done, the whole decimal number goes through the normalize_decimal() function where it takes the value of the decimal and is then converted into a string value and is split up to the left and right portion. So for example if the decimal is 12345.1230, the left value is 12345 and the right is 1230. If there are no values on the right side of the decimal or if the user inputs 123.0, it'll count as empty and will not be used. The left and right will now be combined together to make one string, that is assuming that the right side has a value of more than zero. The function will get the length of the string to determine if it should perform the chosen rounding up option or to add leading zeroes. If the length is less than 16, add leading zeroes and if the length is greater than 16, the function will perform the chosen rounding-up method the user has chosen.

After this process, the program will get the value of e prime in order to acquire the combination field and the exponent continuation. This function first starts getting the binary value of the e prime by calling the get_e_prime function that adds 398 to the exponent and adjusts accordingly if the decimal has been moved towards the right using the length of the right part of the string. The program now runs the get_combination_field function which acquires the value of the combination field depending on the first digit of the normalized decimal. If it's a minor number (0-7) then the first two digits of the combination field are the first two digits of the e_prime_binary value and the last three digits are the last three bits of the most significant digit in binary. However if the first digit is a major number, the first two digits of the combination field will always be 11 then the following two will be the first two binary bits of the e_prime_binary and the last bit depends on the first digit of the normalized decimal, if it's 8, the value is 0 and if the first digit is 9, the value of the last bit in the combination field will be 1. The program then proceeds to call the get_exponent_field function which gets the remaining 8 bits of the e_prime_binary. There were some problems at first if the length of e_prime_binary was less than 10 which happens when the starting binary values are 0 since python disregards them so a binary string of “0100011101” will be stored as “100011101”. This caused issues however it was resolved by checking the total length of the string and if it is less than 10, add the leading zeroes on the left. This logic was also applied in getting the binary value of the most significant digit where it needed to be always 4 binary bits.

For the next step, all that was left was to do the packed BCD conversion of the last 15 digits of the normalized decimal since the first digit was already used in the combination field which were grouped into three digits and were stored in a list to access. The get_BCD_values is then called which goes through the list of grouped decimals. It then checks for the amount of major numbers within the three digits and adjusts the algorithm accordingly based on if it has no major numbers, 1 major number, and 2 major numbers.

Lastly, all the values from the functions mentioned above were concatenated to fit the 64 bits that are needed to convert the answer to hexadecimal.

Before printing the values, the exponent that was inputted by the user will be checked which is used to determine if the input is a special case or not. So if the user inputs a value of 370 and greater, it'll output an infinity case and it varies between negative and positive depending on the sign bit .Next special case would be the NaN where it checks if the value of the decimal input is 0 and the exponent is out of bounds which is 370 and greater or -398 and less.

III. Problems Encountered and Solutions

Throughout the process of creating the program, the problems were focused on how to handle the binary numbers and adjust it to the needed bits since python's bit() function already removes any zeroes if they are not needed. For this the group just simply used the needed size of the binary numbers so if the binary string is less than 10 which is used for the e_prime_binary, then it would add the leading zeroes accordingly and if the values is less than 4 bits for the most significant digit, the program will add the zeros for convenience of the process. Another problem was that the output of bin() uses 0b as an indication of binary value so the group just adjusted the indices so that the string starts after the first two characters. Another problem is when performing the partition of the left and right, if the user did not input a ".0" and simply used "12345" it would read the right value as null which may cause problems in getting the value of e_prime, so in order to fix this, if right is null, it'll change its value to '0' and turns it into an int so that it'll be used in adjusting the e_prime value.

Another problem that related to this was when a user inputs multiple zeros after the radix point (e.g "1237.0000000"). Since the program partitions it and changes it into string, it'll include the zeros; however zeroes on the right side should not be included since the program uses leading zeroes to fill up the number of significant digits. The solution to this problem was to convert the value of the right variable which based on the given example would be “0000000” and change it into an integer which would convert the value to 0. A conditional statement will then be called that if the variable right is equal to 0, then the string won’t contain the right variable anymore so instead of “string = left + right”, it would be “string = left”.

The group also encountered an error that included the sign ‘-’ during acquiring the normalized decimal, since the sign bit was already set, the group just proceeded to remove the character from the string entirely since it is no longer used in the other steps of normalizing, only the sign_bit is needed which is used for the rounding up methods if the length of the string is greater than 16.

IV. Limitations

The program does not handle if the three digits used in the packed BCD functions contain 3 major numbers such as “999”, “889”,“899”, and such. It only handles if there are 0,1, or 2. The program also does not allow the input of characters within the input box for the decimal and the exponent, only allowing 0-9 and the ‘.’ character for the decimal portion since the input asks for a decimal data type. The program as well does not work correctly if the user inputs “+” before the decimal to say that the value is positive since it assumes that if the “-” character is not included, it is automatically positive.

V. Conclusion

Creating the program was a challenge since it required the group to switch up the data type various times such as from decimal to string and the other way around. Converting the decimal input to the binary was complicated due to it needing a lot of conversions happening at different points. There’s also the parts that included checking for major numbers especially during the packed BCD conversion. The packed BCD conversion was the most challenging part in creating the program since there is just so much one needs to take account for and it made it harder to create a short and quick code for the packed BCD conversion. Overall the start was the hardest part in creating the program since the group was not aware of the limited binary outputs in the conversions and was confused on how to make sure that the string is exactly following the 64 bit representation for the different parts. However after making sure that the conversions are correct, it was faster to finish the program with fewer challenges encountered.