Homework #2

(10% of the total grade, 40 points)

Part-1

Attempt the following.

A. Modify the algorithm Fig 10.9 (Week-4, slide #14) for signed binary multiplication (Hint: read slide #17/Week #4). Then implement the modified algorithm in Python or the language of your choice. The input for your program would be two numbers (could be signed or unsigned). The first number would be multiplicand and the second would be the multiplier. You need to convert these to binary notation and then perform the multiplication using the modified flowchart. The result is returned in a tuple (P, B). **[8 points]**

   a. P: product in decimal

   b. B: binary representation of the product

B. Implement the Booth algorithm (flowchart available in Slides #12/13 in Week5 lecture slides). **[8 points]**

C. Compare the performance of both Part 1.A and Part1.B. How much speed up the Part1.B i.e. Booth algorithm gets on an average over the standard Part1.B ? **[4 points]**

Part-2

A. Write a function that takes a floating-point number and a format string (one of the IEEE 754 formats as shown below i.e. “Binary32” or “Binary64” or “Binary128”), processes the input to output the floating-point representation. Steps and examples are provided below. **[8 points]**

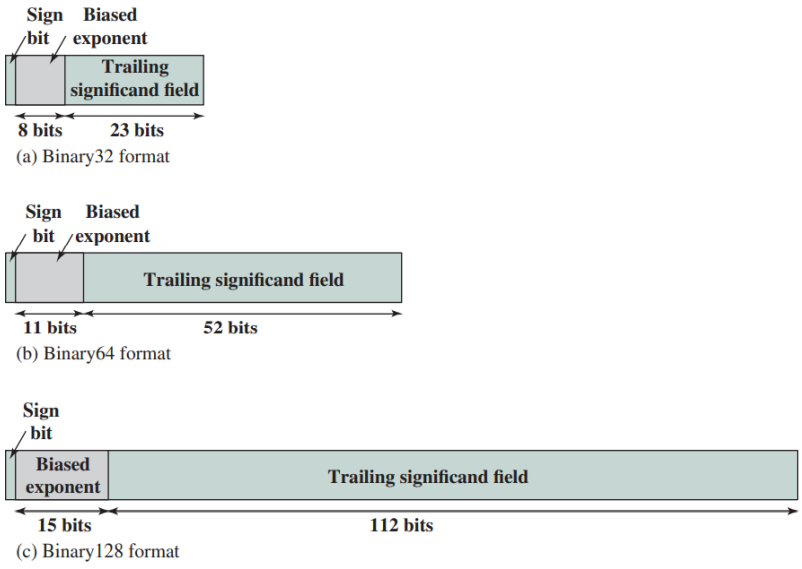

Figure 10.21 IEEE 754 Formats

Steps to convert a floating-point number into the floating-point format

1. Identify and set the sign bit (0: for positive, and 1: negative)
2. Convert the decimal to binary
3. Find the normalized mantissa (significand) in the standard form (1.Mantissa)
4. Find the exponent in Excess-k bit notation
5. Put the values on the format

Example,

Convert $(1460.125)_{10}$ in 32-bits floating point with 1.M format.

1. $(1460.125)_{10}$ => sign bit is 0
2. $(10110110100.001)_2$
3. Normalization: $1.0110110100001 * 2^{10}$
4. Exponent: 10 + 127 = 137 = 10001001
5. 0 10001001 01101101000010000000000 (purple is just padding of zeros to make it to 32 bits)

Suggested signature of the function,

```
# number: any fractional number
# format: {Binary32, Binary64, Binary128}
def convert_float_to_binary(number, format):
    pass
```

B. Write a function that takes a floating-point binary representation and a format string i.e. “Binary32” or “Binary64” or “Binary128”, processes the input to output the floating-point number in decimal format. Steps are basically the reverse engineering of Part2.A. **[8 points]**

Suggested signature of the function,

```
# binary_string: binary representation of a fractional number
# format: {Binary32, Binary64, Binary128}
def convert_binary_to_float(binary_string, format):
    pass
```

C. Conduct an analysis on Part2.A and Part2.B using 100 different inputs to show how many times we get the expected result. The expected result is explained below using an example: **[2 points]**

convert_float_to_binary(number, “Binary32”) => “bbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbb”

convert_binary_to_float(“bbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbb”, “Binary32”) => number

Submission checklist:

Part 1:

Code files: standard_multiplication.py, booth_multiplication.py, part1_analysis.py. A brief report Part1.pdf on the problem you faced, the process you followed, the things you learned along with the example outputs of runs or graphs as appropriate.

Part 2:

Code files: convert_float_to_binary.py, convert_binary_to_float.py, part2_analysis.py.

A brief report Part2.pdf on the problem you faced, the process you followed, the things you learned along with the example outputs of runs or graphs as appropriate.