EEE3530-01 **Spring 2024**

Assignment 3: Floating-Point Numbers

Due: Monday, May 5, 2024, 11:59PM

1 Introduction

- The objective of this assignment is to implement double-precision floating-point add and subtract functions that perform bit-level operations on the sign, exponent, and fraction fields of the numbers to calculate the results.
- This assignment is independent of Kite.
- The IEEE754 floating-point standard defines an 8-byte double-precision format with 1-bit sign, 11-bit exponent, and 52-bit fraction.
- It follows basically the same rule as the single-precision data format except for using the different number of bits for the exponent and fraction.
- The following shows a code that extracts the sign, exponent, and fraction of a double-precision value in C++.

```
/* example.cc */

#include <iostream>

using namespace std;

int main(void) {
    double u = -1.25;                          // A double value to test
    uint64_t v = *(uint64_t*)&u;               // Read the double as uint64_t

    uint64_t sign = v >> 63;                   // MSB for sign
    uint64_t exponent = (v >> 52) & 0x7ff;     // 11-bit exponent
    uint64_t fraction = v & 0xfffffffffffff;   // 52-bit fraction

    // Print out each field of the double.
    cout << "sign     = "   << sign            << endl;
    cout << "exponent = "   << exponent        << endl;
    cout << "fraction = 0x" << hex << fraction << endl;

    // Put back the sign, exponent, and fraction into a 64-bit unsigned.
    uint64_t x = 0;
    x |= sign     << 63;
    x |= exponent << 52;
    x |= fraction;

    double y = *(double*)&x;                   // Read uint64_t as double.
    cout << "double = " << y << endl;          // Print for confirmation.

    return 0;
}
```

- Executing the code above produces the following output. The sign bit is 1 (i.e., minus), and the exponent is 1023 (i.e., 0). The 52-bit fraction field is 0100 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 (i.e., 2^{-2}). Collectively, -1.25 is represented as $(-1)^{\text{sign}} \times 2^{\text{exponent}-\text{bias}} \times (1.0 + \text{fraction})$.

```
$ g++ -o example example.cc
$ ./example
sign     = 1
```

```
exponent = 1023
fraction = 0x4000000000000
double = -1.25
```

- As shown in the example, the sign, exponent, and fraction of a `double` value can be easily extracted using the bit-shifting and masking operations.
- Once the sign, exponent, and fraction of `double` variables are obtained, you can follow the steps of lecture slides to manipulate them as if two double-precision floating-point numbers are added or subtracted.
- When working on the extracted sign, exponent, and fraction parts, you do not have to perform bit-granular operations to add or subtract the exponents and fractions in this assignment. Let the native `+` and `-` handle the operations of the extracted parts.
- After the sign, exponent, and fraction of the result are derived, put them back into a `double` variable. Printing out the value of it will confirm if the sign, exponent, and fraction of the output were correctly calculated.
- To start the assignment, use the `wget` command to download a tar file containing a skeleton code. Compiling and executing the skeleton code will print the following.

```
$ wget https://casl.yonsei.ac.kr/teaching/eee3530/float.tar
$ tar xf float.tar
$ cd float
$ make
g++ -Wall -Werror -g -o float.o -c float.cc
g++ -Wall -Werror -g -o main.o -c main.cc
g++ -Wall -Werror -g -o float float.o main.o

$ ./float
======== [Tests for grading] ========
Test #1: 3.875 + 2.625 = 0
Test #2: -10.1 + 1.32 = 0
Test #3: -0.121 + -42.52 = 0
Test #4: 1.39e+52 + 8.24e+49 = 0
...

Test #19: 1.11254e-308 - -1.11254e-308 = 0
Test #20: inf - -inf = 0
======== [End of tests] ========
```

2 Implementation

- The `float/` directory includes the following five files besides `float.sh`.

```
$ ls
Makefile    float.cc    float.h    main.cc    tar.sh
```

- `Makefile` defines rules to compile the files. You do not have to touch this.
- `float.h` is a header file defining a class, `float64_t`. The class is fully defined for you to start the assignment. You do not have to change this file unless you want to add more functions or variables. Feel free to add more items if needed.
- If you are not familiar with C++ programming, seek help from the instructor or TA to clarify your understanding of the provided code. You won't have enough time to get help if you start the assignment late.
- `float.cc` has the actual implementation of the `float64_t` class. Your assignment is to fill in the two functions (i.e., `operator+` and `operator-`) in the file. You do not have to touch anything in the first half of the file above the assignment banner.

- Each operator function (i.e., `operator+` and `operator-`) in the skeleton code creates and returns a dummy variable (i.e., `float64_t r`), which is initialized to zero. Therefore, you should see all the calculation results as zeros when executing the skeleton code.
- The `operator+` function shows an example of how to extract the sign, exponent, and fraction of floating-point variables to add, although this part is commented out. Use this example to calculate the sign, exponent, and fraction of the output, and then put them into `r.data` to return.
- For rounding the fraction of intermediate values or the final result, simply discard the bits that do not fit into 52 bits. This may produce several lowest bits different from what you may get if the numbers are directly calculated as `double` variables in the code, but the differences should be insignificant - in many cases, `double` values are approximate anyway.
- Lastly, `main.cc` has the `main()` function, which runs 20 test sets. Do not modify the tests because they will be used for grading your assignment.
- The tests are fairly comprehensive, including ordinary floating-point numbers, denormalized values, ±infinity, NaN, and zeros. Make sure you handle the cases correctly.
- Once the operator functions are implemented, executing the code should produce the following results.
- `std::cout` or `printf` may not print all fraction digits, and the lowest digit being printed should be rounded. This is perfectly fine.

```
$ make
g++ -Wall -Werror -g -o float.o -c float.cc
g++ -Wall -Werror -g -o main.o -c main.cc
g++ -Wall -Werror -g -o float float.o main.o

$ ./float
======== [Tests for grading] ========
Test #1: 3.875 + 2.625 = 6.5
Test #2: -10.1 + 1.32 = -8.78
Test #3: -0.121 + -42.52 = -42.641
...
```

3 Submission

- After the assignment is done, execute the `tar.sh` script in the `float/` directory. It creates a tar file named after your student ID (e.g., `2024143530.tar`). Upload the tar file on LearnUs. Do not rename the file.

```
$ ./tar.sh
rm -f float.o main.o float
a float
a float/main.cc
...

$ ls
2024143530.tar    Makefile    float.cc    float.h    main.cc    tar.sh
```

4 Grading Rules

- The following is the general guideline for grading. A 30-point scale will be used for this assignment. The minimum score is zero, and negative scores will not be given. Grading rules are subject to change; a grader may add a few extra rules without notice for a fair evaluation of students' efforts.

  -5 points: The tar file includes redundant tags such as a student name, `hw3`, etc.

-5 points: The code has insufficient comments. Comments in the skeleton code do not count. You must clearly explain what each part of your code does.

-2 points each: There are a total of 20 sets of testing (i.e., `Test #1: 3.875 + 2.625`). For each incorrect result, 2 points will be deducted; missing more than 15 tests will earn no scores.

-30 points: No or late submission.

Final grade = F: The submitted tar file is copied from someone else. All students involved in the incidents will get Fs for the final grade.

- Your teaching assistant (TA) will grade your assignments. If you think your assignment score is incorrect, discuss your concerns with the TA. Always be courteous when contacting the TA. If you and the TA do not reach an agreement, elevate the case to the instructor for review of your assignment. Refer to the course website for the contact information of the TA and instructor: `https://casl.yonsei.ac.kr/eee3530`
- Arguing for partial credits for no valid reasons will be regarded as a cheating attempt; such a student will lose the assignment scores.