SEE2030: Introduction to Computer Systems (Fall 2017)

Programming Assignment #2:

Implementing Arithmetic Operations using the Tiny FP (8-bit floating point) representation

Due: October 15th (Sunday), 11:59PM

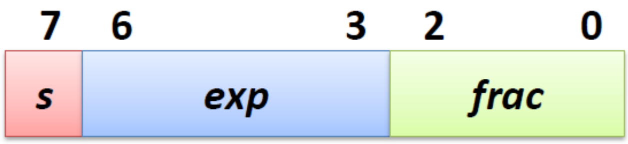
1. Introduction

The purpose of this assignment is to implement several arithmetic operations using the simplified 8-bit floating-point type.

2. Problem specification

2.1. Overview

`tinyfp` is a simplified 8-bit floating point representation which follows the IEEE 754 standard for floating-point arithmetic. The overall structure of the **`tinyfp`** representation is shown below. The MSB (Most Significant Bit) is used as a sign bit (***s***). The next four bits are used for exponents (***exp***) with a bias value of 7. The last three bits are used for the fractional part (***frac***).



In C, the new type **`tinyfp`** can be defined as follows.

```
typedef unsigned char tinyfp;
```

Your task is to implement the following four C functions that operate using the **`tinyfp`** data type.

```
tinyfp add(tinyfp tf1, tinyfp tf2);
tinyfp mul(tinyfp tf1, tinyfp tf2);
int gt(tinyfp tf1, tinyfp tf2);
int eq(tinyfp tf1, tinyfp tf2);
```

2.2. Restrictions

You must not use any **`float`**-type or **`double`**-type variables for implementing functions introduced above. Also, you should not use the conversion functions such as **`tinyfp2float()`** implemented in PA#1. You should implement these functions using 32-bit integer arithmetic and logical operations.

2.3. Implementation details

2.2.0. Common

- We do not differentiate between +NaN and -NaN. They are all treated as NaN.
  - e.g. $(0\ 1111\ 101)_{tinyfp}$ == $(1\ 1111\ 110)_{tinyfp}$ == NaN
- Also note that +0 and -0 are same by definition. The all mean the value zero.

2.2.1. **`tinyfp add(tinyfp tf1, tinyfp tf2)`**

- If the result exceeds the range of the **`tinyfp`** representation, the result will be infinity (+∞ or -∞ depending on the sign).
- Use the **round-to-even** mode when necessary.
- For the special cases which involve infinity and NaN, **`add()`** should return the value specified in Table 1.

Table 1. Return values of add() for special values

`tf1`	**`tf2`**	Return value
+∞	Normalized/Denormalized	+∞
-∞	Normalized/Denormalized	-∞
Normalized/Denormalized	+∞	+∞
Normalized/Denormalized	-∞	-∞
+∞	+∞	+∞
-∞	-∞	-∞
+∞	-∞	NaN
-∞	+∞	NaN
NaN	Anything	NaN
Anything	NaN	NaN

2.2.2. `tinyfp mul(tinyfp tf1, tinyfp tf2)`

- If the result exceeds the range of **`tinyfp`** representation, the result will be infinity (+∞ or -∞ depending on the sign).
- Use the **round-to-even** mode when necessary.
- For the special cases which involve infinity and NaN, **`mul()`** should return the value specified in Table 2.

Table 2. Return value of mul() for special values

`tf1`	**`tf2`**	Return value
±∞	Normalized/Denormalized	±∞ (depending on the sign of **`tf1`** and **`tf2`**)
Normalized/Denormalized	±∞	±∞ (depending on the sign of **`tf1`** and **`tf2`**)
±∞	±0	NaN
±0	±∞	NaN
+∞	+∞	+∞
-∞	-∞	+∞
+∞	−∞	−∞
-∞	+∞	−∞
NaN	Anything	NaN
Anything	NaN	NaN

2.2.3. `int gt(tinyfp tf1, tinyfp tf2)`

- Return 1 if **`tf1`** is greater than **`tf2`**. Otherwise, return 0.
- For the special cases which involve infinity and NaN, **`gt()`** should return the value specified in Table 3.

Table 3. Return value of gt() for special values

`tf1`	**`tf2`**	Return value
+∞	Normalized/Denormalized	1
Normalized/Denormalized	+∞	0
-∞	Normalized/Denormalized	0
Normalized/Denormalized	-∞	1
+∞	+∞	0
+∞	-∞	1
-∞	+∞	0
-∞	-∞	0
+0	+0	0
+0	-0	0
-0	+0	0
-0	-0	0
Anything	NaN	0
NaN	Anything	0

2.2.4. `int eq(tinyfp tf1, tinyfp tf2)`

- Return 1 if **`tf1`** and **`tf2`** are equal. Otherwise, return 0.
- For the special cases which involve infinity and NaN, **`eq()`** should return the value specified in Table 4.

Table 4. Return value of eq() for special values

`tf1`	**`tf2`**	Return value
+∞	+∞	1
-∞	-∞	1
+∞	-∞	0
-∞	+∞	0
+0	+0	1
+0	-0	1
-0	+0	1
-0	-0	1
NaN	Anything	0
Anything	NaN	0

3. Example

The skeleton code is available in the course homepage (http://csl.skku.edu/SSE2030F17/Assignment)

The results of some sample runs are as follows.

```
kisik@kisik-desktop:~/sse2030/pa2$ ./pa2-test
Test 1: Addition
00111100 + 00111100 = 01000100, CORRECT
00111100 + 01001010 = 01001101, CORRECT
01001010 + 01001010 = 01010010, CORRECT
Test 2: Multiplication
00111100 * 00111100 = 01000001, CORRECT
00111100 * 01001010 = 01001111, CORRECT
01001010 * 01001010 = 01011100, CORRECT
Test 3: Greater than
00111100 > 00111100: X, CORRECT
00111100 > 01001010: X, CORRECT
01001010 > 00111100: O, CORRECT
01001010 > 01001010: X, CORRECT
Test 4: Equal to
00111100 == 00111100: O, CORRECT
00111100 == 01001010: X, CORRECT
01001010 == 01001010: O, CORRECT
kisik@kisik-desktop:~/sse2030/pa2$
```

4. Hand in instructions

- Submit only the **pa2.c** file to the submission site.

5. Logistics

- You will work on this assignment alone.
- We will also measure the speed of your program. The correct solutions with fast running time will have the bonus.
- Only the assignments submitted before the deadline will receive the full credit. 25% of the credit will be deducted for every single day delay.
  - You can use up to 5 ***slip days***
- Any attempt to copy others' work will result in heavy penalty (for both the copier and the originator). Don't take a risk.

Good luck!

TA: Kisik Jeong (kisik.jeong@csl.skku.edu)

Computer Systems Laboratory

Sungkyunkwan University