4190.308: Computer Architecture (Spring 2018)

Project #2: Tiny FP (8-bit floating point) Arithmetic Operations

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

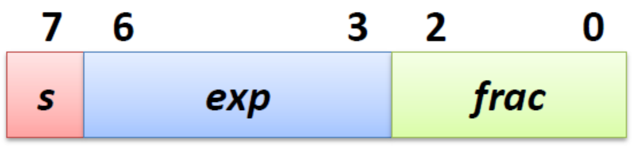
1. Introduction

The goal of this project is to implement several arithmetic operations using the simplified 8-bit TinyFP floating-point representation.

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`** is defined as follows.

```
typedef unsigned char tinyfp;
```

Your task is to implement the following four C functions that operate on the values in the **`tinyfp`** representation.

```
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 Implementation details

2.2.1. 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. They all mean the value zero.

2.2.2. 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

2.2.3. 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

2.2.4. 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

2.2.5. 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

2.3 Restrictions

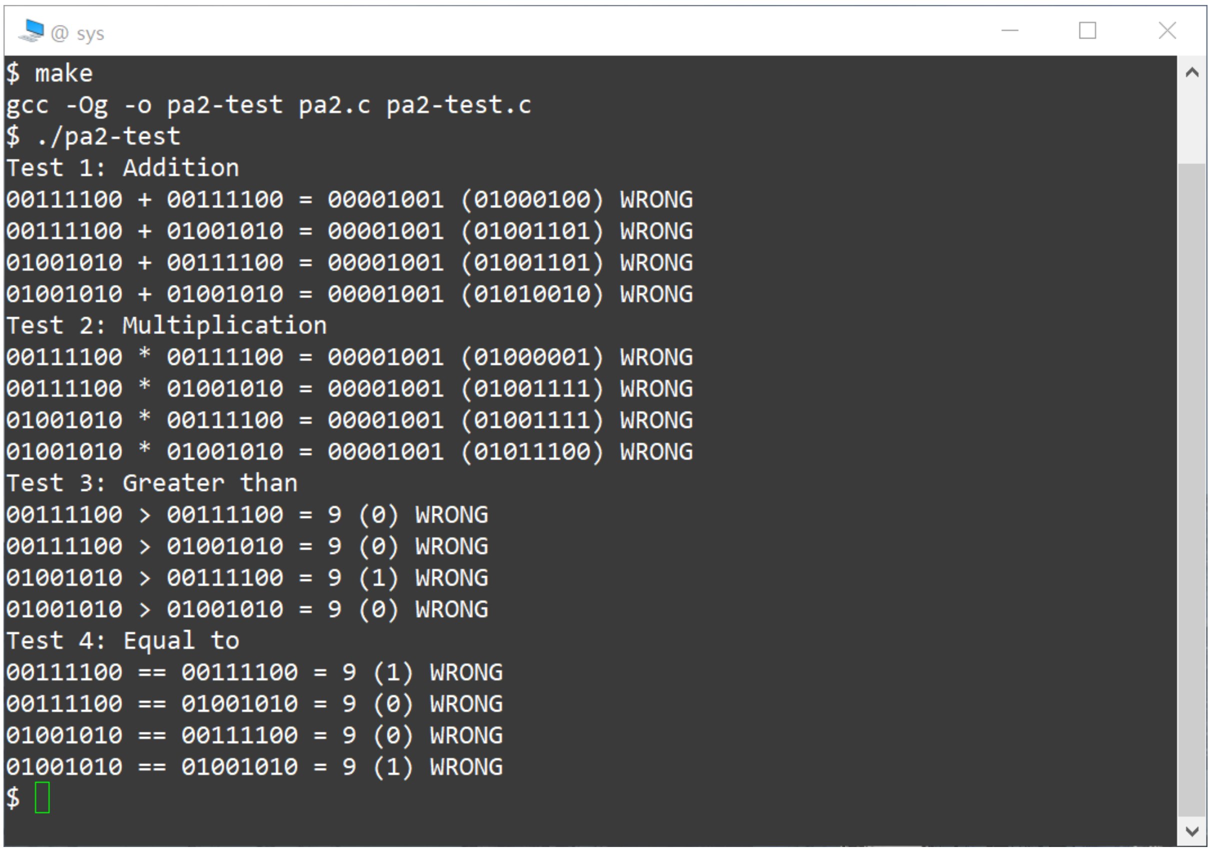
You will get NO points if you violate the following restrictions:

- You must not use any **float**-type or **double**-type variables or constants for implementing functions introduced above. Also, you should not use the conversion functions such as **tinyfp2float()** implemented in Project #1. You should implement the required functions using integer arithmetic and logical operations. If your source code contains "float" or "double" words (even in comments), it will be rejected by the automatic grader.
- Do not use any external library functions such as `memcpy()`, `abs()`, `pow()`, etc. defined in `math.h`, `stdio.h`, etc.
- Do not use any brute-force approach such as looking up the table where the results are precalculated.

3. Example

The skeleton code will be available in the submission site (http://sys.snu.ac.kr).

The result of a sample run is as follows.



```
$ make
gcc -Og -o pa2-test pa2.c pa2-test.c
$ ./pa2-test
Test 1: Addition
00111100 + 00111100 = 00001001 (01000100) WRONG
00111100 + 01001010 = 00001001 (01001101) WRONG
01001010 + 00111100 = 00001001 (01001101) WRONG
01001010 + 01001010 = 00001001 (01010010) WRONG
Test 2: Multiplication
00111100 * 00111100 = 00001001 (01000001) WRONG
00111100 * 01001010 = 00001001 (01001111) WRONG
01001010 * 00111100 = 00001001 (01001111) WRONG
01001010 * 01001010 = 00001001 (01011100) WRONG
Test 3: Greater than
00111100 > 00111100 = 9 (0) WRONG
00111100 > 01001010 = 9 (0) WRONG
01001010 > 00111100 = 9 (1) WRONG
01001010 > 01001010 = 9 (0) WRONG
Test 4: Equal to
00111100 == 00111100 = 9 (1) WRONG
00111100 == 01001010 = 9 (0) WRONG
01001010 == 00111100 = 9 (0) WRONG
01001010 == 01001010 = 9 (1) WRONG
$
```

4. Hand in instructions

– Submit only the **`pa2.c`** file to the submission site.

5. Logistics

- You will work on this project alone.
- Only the upload 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*** during this semester. Please let us know the number of slip days you want to use in the QnA board in the submission site within 1 week after the deadline.
- Any attempt to copy others' work will result in heavy penalty (for both the copier and the originator). Don't take a risk.

Have fun!

Jin-Soo Kim
Systems Software Laboratory
Dept. of Computer Science and Engineering
Seoul National University