서울대학교
SEOUL NATIONAL UNIVERSITY

Systems Software Laboratory
Dept. of Computer Science and Engineering

4190.308: Computer Architecture (Spring 2018)

Project #1: Tiny FP (8-bit floating point) representation

Due: April 1st (Sunday), 11:59PM

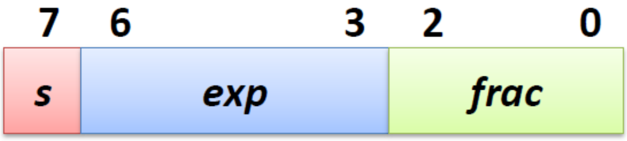
1. Introduction

The purpose of this project is to get familiar with the floating-point representation by implementing a simplified 8-bit 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 convert **`int`** or **`float`** type values to the **`tinyfp`** format and vice versa.

```
tinyfp int2tinyfp(int x);
int tinyfp2int(tinyfp x);
tinyfp float2tinyfp(float x);
float tinyfp2float(tinyfp x);
```

2.2. Implementation details

2.2.1. `int2tinyfp()`

- Integer zero (0) should be converted to plus zero (+0.0) in **`tinyfp`**.
- An integer value that exceeds the range of the **`tinyfp`** representation should be converted to the infinity in **`tinyfp`** (+∞ or -∞ depending on the sign).
- If necessary, use the **round-toward-zero** mode.

2.2.2. `tinyfp2int()`

- Drop the fractional part when you convert values in the **`tinyfp`** format to integers. (e.g. the value 1.5 in **`tinyfp`** is converted to 1)
- Convert +∞ and -∞ in **`tinyfp`** to **TMin** in integer. (**TMin** represents the smallest integer that can be represented in the 32-bit signed integer format.)
- +NaN and -NaN in **`tinyfp`** are also converted to **TMin** in integer.

2.2.3. `float2tinyfp()`

- A floating-point value that exceeds the range of the **`tinyfp`** representation should be converted to the infinity in **`tinyfp`** (+∞ or -∞ depending on the sign).
- +NaN and -NaN in **`float`** should be converted to the corresponding +NaN and -NaN in **`tinyfp`**, respectively.
- +∞ and -∞ in **`float`** should be converted to the corresponding +∞ and -∞ in **`tinyfp`**, respectively.
- If necessary, use the **round-toward-zero** mode.

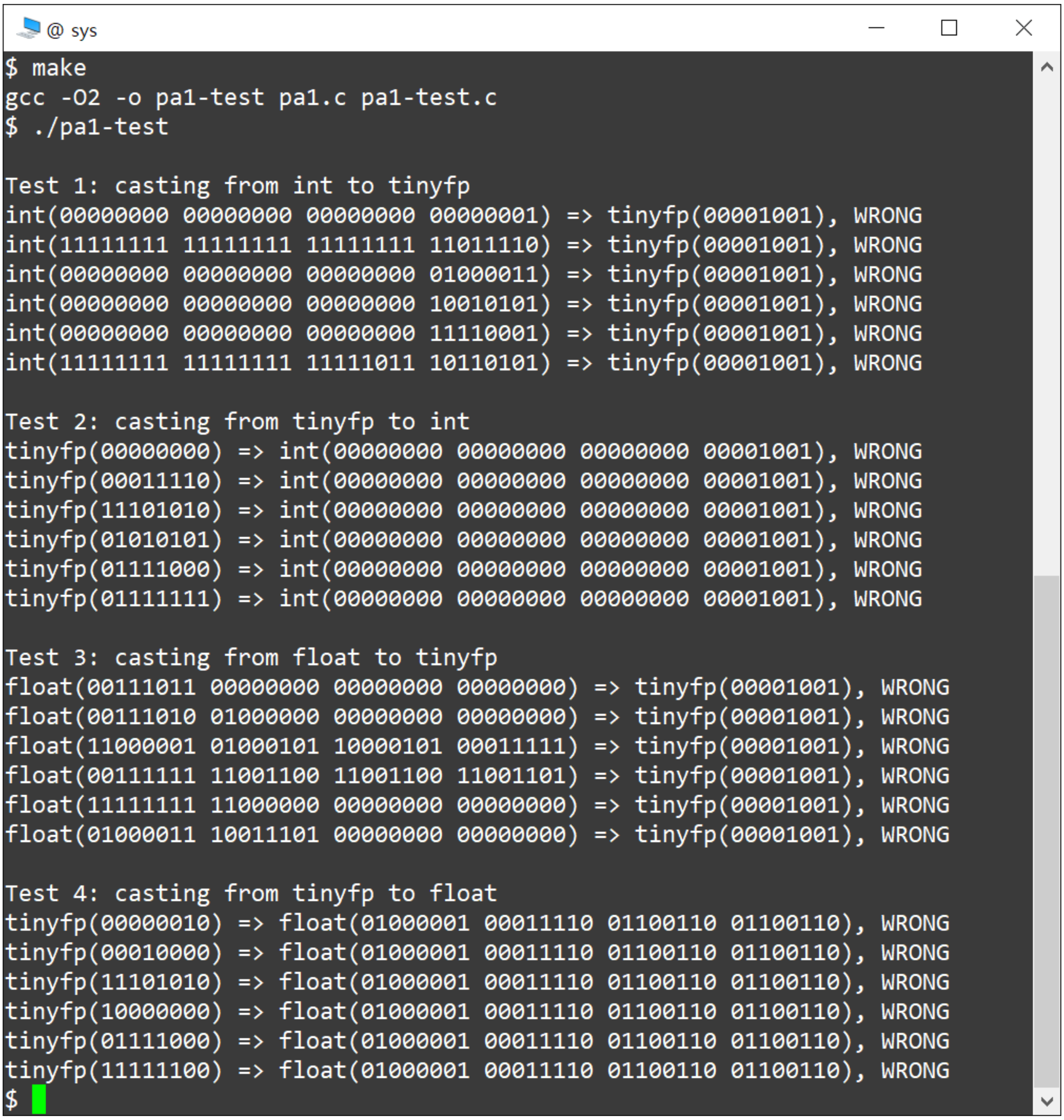
2.2.4. `tinyfp2float()`

- The **`tinyfp`** type is a subset of the **`float`** type. Hence, all the values in **`tinyfp`** can be represented in the **`float`** format without any error.
- Again, +NaN and -NaN in **`tinyfp`** should be converted to the corresponding +NaN and -NaN in **`float`**, respectively. +∞ and -∞ in **`tinyfp`** should be converted to the corresponding +∞ and -∞ in **`float`**, respectively.

3. Example

The skeleton code will be available in the eTL system.

The result of a sample run is as follows.



```
$ make
gcc -O2 -o pa1-test pa1.c pa1-test.c
$ ./pa1-test

Test 1: casting from int to tinyfp
int(00000000 00000000 00000000 00000001) => tinyfp(00001001), WRONG
int(11111111 11111111 11111111 11011110) => tinyfp(00001001), WRONG
int(00000000 00000000 00000000 01000011) => tinyfp(00001001), WRONG
int(00000000 00000000 00000000 10010101) => tinyfp(00001001), WRONG
int(00000000 00000000 00000000 11110001) => tinyfp(00001001), WRONG
int(11111111 11111111 11111011 10110101) => tinyfp(00001001), WRONG

Test 2: casting from tinyfp to int
tinyfp(00000000) => int(00000000 00000000 00000000 00001001), WRONG
tinyfp(00011110) => int(00000000 00000000 00000000 00001001), WRONG
tinyfp(11101010) => int(00000000 00000000 00000000 00001001), WRONG
tinyfp(01010101) => int(00000000 00000000 00000000 00001001), WRONG
tinyfp(01111000) => int(00000000 00000000 00000000 00001001), WRONG
tinyfp(01111111) => int(00000000 00000000 00000000 00001001), WRONG

Test 3: casting from float to tinyfp
float(00111011 00000000 00000000 00000000) => tinyfp(00001001), WRONG
float(00111010 01000000 00000000 00000000) => tinyfp(00001001), WRONG
float(11000001 01000101 10000101 00011111) => tinyfp(00001001), WRONG
float(00111111 11001100 11001100 11001101) => tinyfp(00001001), WRONG
float(11111111 11000000 00000000 00000000) => tinyfp(00001001), WRONG
float(01000011 10011101 00000000 00000000) => tinyfp(00001001), WRONG

Test 4: casting from tinyfp to float
tinyfp(00000010) => float(01000001 00011110 01100110 01100110), WRONG
tinyfp(00010000) => float(01000001 00011110 01100110 01100110), WRONG
tinyfp(11101010) => float(01000001 00011110 01100110 01100110), WRONG
tinyfp(10000000) => float(01000001 00011110 01100110 01100110), WRONG
tinyfp(01111000) => float(01000001 00011110 01100110 01100110), WRONG
tinyfp(11111100) => float(01000001 00011110 01100110 01100110), WRONG
$ 
```

4. Hand in instructions

- Register an account to the submission site (will be open soon)
  - You must enter your real name & student ID
  - Wait for an enrollment.
- Submit only the **`pa1.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 after each submission.
- 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!

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