Computer Architecture

CPE 315 - Section 03

Lab 2 - 01/24/18

Jenna Stephens
Venkat Akkinepally
Austin Whaley

Introduction:

In this lab, we learned binary arithmetic (unsigned fixed-point multiplication, and used floating point operations, specifically in the IEEE 754 floating point format. Through our programming in C, we simulate hardware implementation of these concepts. This lab was designed to teach us packing and unpacking IEEE 754 single precision values, as well as normalizing, adding, subtracting, and multiplying those values.

Functional Requirements:

By the end of this lab, we will have a better understanding of floating point numbers and their representations in hardware. In addition, we will be able to use the IEEE 754 floating point format.

Approach Used:

We used the given code skeletons as a base or starting point, trying to convert the steps from binary into bitwise operations in C and hex values.

Program Listing:

See lab2.c.

Source code, with comments explaining operation of code. Each function must include header that describes calling conventions, assumptions, and return parameters.

Discussion of Difficulties:

One issue of reliability is round off error. The IEEE format is an approximation of fractions, and will never be exact. This was particularly apparent in part 1d. We have run into many segmentation faults dealing with INTFLOAT struct that hampered our progress for the rest of the lab. We are aware of the issue and will continue to fix it.

Summary:

We have learned that it is very difficult to visualize and implement these concepts in C. In addition, procrastination makes assignments more complicated, especially for group work. Overall, we have a better understanding of IEEE 745 and how floating point numbers are stored in hardware.

Results:

=========Part 1==========

1a. a=0x0001, b=0x0001 c=0x0001

1b. a=0x0001, b=0xFFFF c=0xFFFF

1c. a=0x8000, b=0x0001 c=0x8000

1d. a=0x4000, b=0x4000 c=0x10000000

1e. a=0x8000, b=0x8000 c=0x40000000

==========

=========Part 2==========

2a. Test case: 0x40C80000

Float: 6.250000

Exponent:*

Fraction:0x********

2b. Test case: 0xc3000000

Float: -2.000000

Exponent:*

Fraction:0x********

2c. Test case: 0x3e000000

Float: 0.125000

Exponent:*

Fraction:0x********

2d. Test case: 0x3EAAAAAB

Float: 0.333333

Exponent:*

Fraction:0x********

==========

=========Part 3==========

3a. Test case: 0x40C80000

Float:*

3b. Test case: 0xC3000000

Float:*

3c. Test case: 0x3E000000

Float:*

3d. Test case: 0x3EAAAAAB

Float:*

==========

=========Part 4==========
4a. Exp = *, Frac = 0x********
4b. Exp = -9, Frac = 0x********
4c. Exp = 3, Frac = 0x********
4d. Exp = -13, Frac = 0x********
==========

=========Part 5==========
5a. 0xBF800000 and 0x3F800000 (-1 and +1)
 Sum:0x******** (* Decimal Value)
5b. 0x3F800000 and 0x3E800000 (1 + .25)
 Sum:0x******** (*)
5c. 0x40800000 and 0xBE000000 (4.0 + (-.125))
 Sum:0x******** (*)
==========

=========Part 6==========
6a. 0x40400000 and 0x3F800000 (3 – 1)
 Diff:0x******** (* Decimal Value)
6b. 0x40400000 and 0xBF800000 (3 – (-1))
 Diff:0x******** (*)
6c. 0x40000000 and 0x40000000
 Diff:0x******** (*)
==========

=========Part 7==========
7a. 0x40200000 and 0x40200000 (2.5 x 2.5)
 Product:0x******** (* Decimal Value)
7b. 0xc1700000 and 0x45800000 (-15 x 4096)
 Product:0x******** (*)
==========