Class:	CPE300L 1001		Semester:	Fall 2024
Points		Document author:	**Alireza Bolourian**	
		Author's email:	**bolouria@unlv.nevada.edu**	
		Document topic:	**postlab 06**	
Instructor's comments:				

1. Theory of Operation

IEEE 754 defines a standard for representing floating point numbers with three components. It also distinguishes single and double precision floating numbers which use 32 and 64 bits respectively. The components include sign, exponent and significand also called mantissa. The exponent is biased because it summed with 127 and the fraction part hides the leading 1 to the left of the binary point. When a decimal number is converted to this form, then addition, subtraction, multiplication and division can be performed following respective algorithms.

2. Prelab

Add 7.875 , 0.1875

① 7.875 sign bit=0

② 111 → 0.875 × 2 = 1.750

0.750 × 2 = 1.500

0.5 × 2 = 1.0

111.111

③ Normalize

1.11111×2^{2}

④ Exponent

E = 2 + 127 = 129

⑤ Convert E to binary

10000001

⑥ Combine sign, E, F

0 10000001 1111100……0

① 0.1875 sign bit=0

② 0 → 0.1875 × 2 = 0.375

0.375 × 2 = 0.75

0.75 × 2 = 1.50

0.5 × 2 = 1.0

0.0011

③ 1.1×2^{-3}

④

E = −3 + 127 = 124

⑤

01111100

⑥

0 01111100 11000···0

step 1:
S1 E1 F1
0 10000001 1.11111000---0

step 2:
S2 E2 F2
0 01111100 1.1000....0

step 3: shift F2 right 5 times, add 5 to E2

S2 E2 F2 g r s
0 10000001 0.00000100...0 0 0 0

E2=E1

step4:
F1+F2
1.1111100000----0
+0.00000100.......0
10.00000100.... 0

step 5:
S E F
Result=0 10000010 0000001000-...0

8.0625

Subtraction

step 1:
S1 E1 F1
0 1000 0001 11111000---0

step 2:
S2 E2 F2
0 01111100 1.1000... 0

step 3:
20
S2 E2
0 10000001 0.000001100----0 0 0 0

step4:
F1-F2

1.11111000 ---- 0
-0.00000100 ... 0
1.11101100 ... 0

step 5-
S E F
Result= 0 10000001 11101100---0

7.6875

Multiplication

① Unpack: 7.875 0.1875

0 10000001 111100...0 0 01111100 1000...0

② Exponent handling:

$$\begin{array}{r} 10000001 \\ -\ 01111111 \\ \hline 00000010 \end{array} \qquad \begin{array}{r} 01111111 \\ -\ 01111100 \\ \hline 00000011 \end{array}$$

③ Multiply two significants

$$\begin{array}{r} 1.1111100...0 \\ \times\ 1.1000...0 \\ \hline +\ 01111111 \\ 11111110 \\ \hline 10.111101 \end{array}$$

④ Subtract the Exponents

$$\begin{array}{r} 00000010 \\ -\ 00000011 \\ \hline 11111111 \end{array} \qquad \begin{array}{r} 11111111 \\ +\ 01111111 \\ \hline 01111110 \end{array}$$

⑤ Normalize and Round

F: 1.0111101000--0 E: 01111111

⑥ Sign bit $S = S1 \oplus S2 = 0$

⑦ Pack

S	E	F	
0	01111111	0111101000---0	1.4765625

3. Results of the experiments

1. Addition:

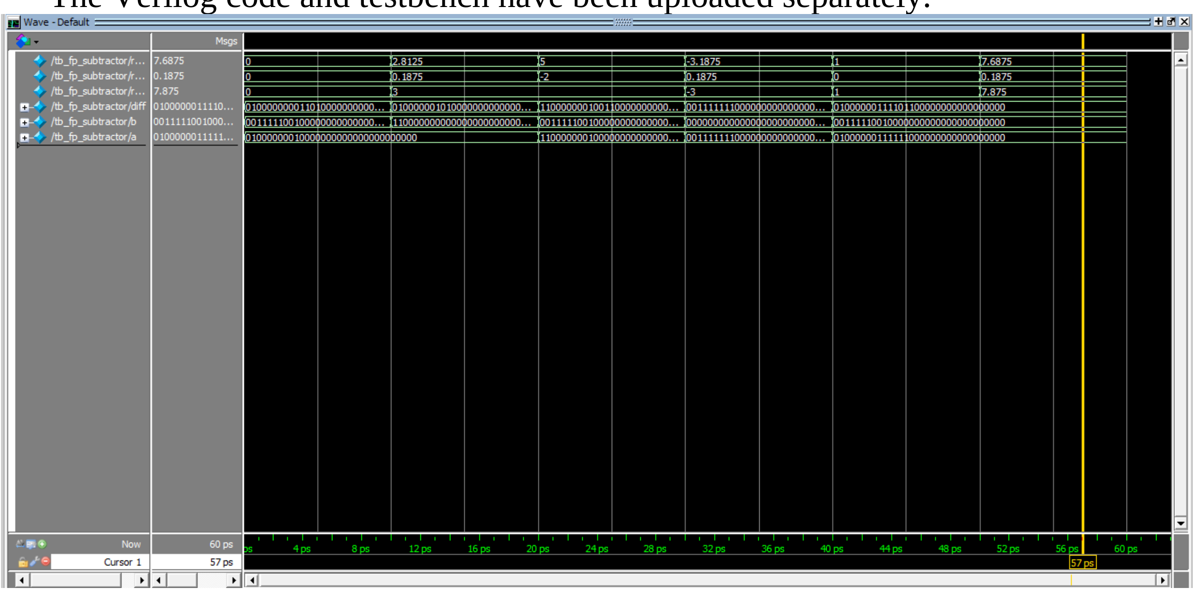
The Verilog code and testbench are uploaded separately.

```
# Floating-Point Adder Test
# ----------------------------------------------------
# Time: 10 | a: 3f800000 (1.000000) | b: 40000000 (2.000000) | sum: 40400000 (3.000000)
# Time: 20 | a: 40400000 (3.000000) | b: 40fc0000 (7.875000) | sum: 412e0000 (10.875000)
# Time: 30 | a: c0400000 (-3.000000) | b: c0000000 (-2.000000) | sum: c0a00000 (-5.000000)
# Time: 40 | a: 3f800000 (1.000000) | b: 00000000 (0.000000) | sum: 3f800000 (1.000000)
# Time: 50 | a: 40fc0000 (7.875000) | b: 3e400000 (0.187500) | sum: 41010000 (8.062500)
# Time: 60 | a: 40400000 (3.000000) | b: 3e400000 (0.187500) | sum: 404c0000 (3.187500)
# Test completed.
```

The waveform and console output test the results of the addition module.

2. Subtraction

The Verilog code and testbench have been uploaded separately.

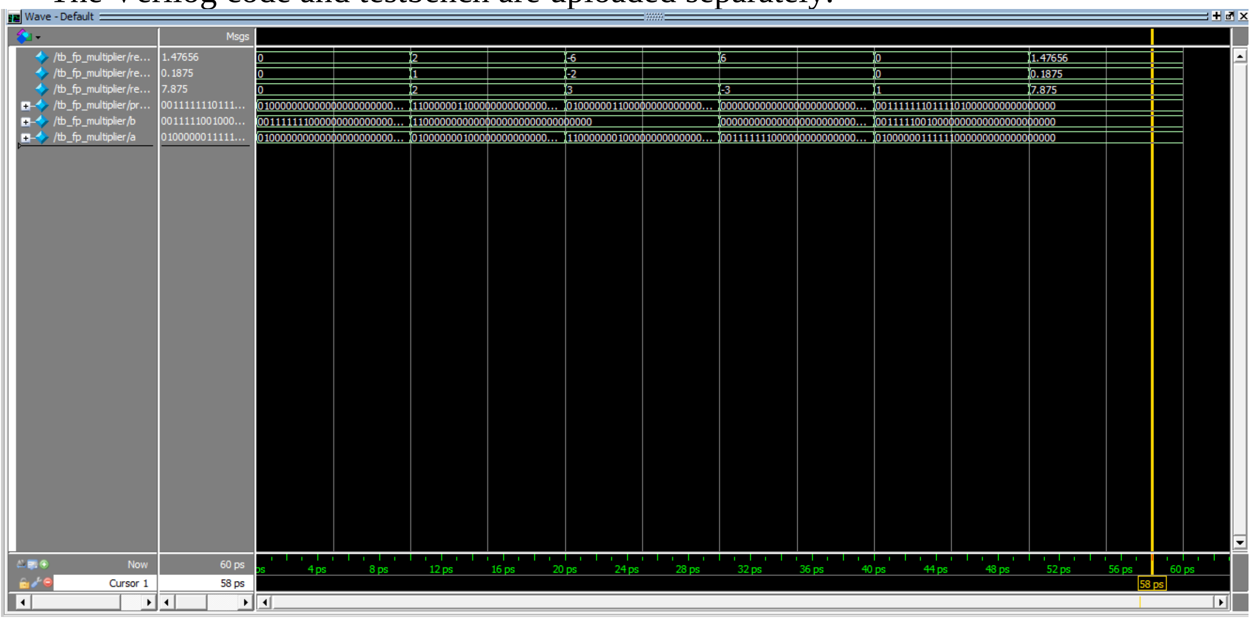


```
# Floating-Point Subtractor Test
# --------------------------------------------------
# Time: 10 | a: 40400000 (3.000000) | b: 3e400000 (0.187500) | diff: 40340000 (2.812500)
# Time: 20 | a: 40400000 (3.000000) | b: c0000000 (-2.000000) | diff: 40a00000 (5.000000)
# Time: 30 | a: c0400000 (-3.000000) | b: 3e400000 (0.187500) | diff: c04c0000 (-3.187500)
# Time: 40 | a: 3f800000 (1.000000) | b: 00000000 (0.000000) | diff: 3f800000 (1.000000)
# Time: 50 | a: 40fc0000 (7.875000) | b: 3e400000 (0.187500) | diff: 40f60000 (7.687500)
# Test completed.
```

The results of the subtraction are displayed in the waveform simulation and console output.

3. Multiplication

The Verilog code and testbench are uploaded separately.




```
# Floating-Point Multiplier Test
# --------------------------------------------------
# Time: 10 | a: 40000000 (2.000000) | b: 3f800000 (1.000000) | product: 40000000 (2.000000)
# Time: 20 | a: 40400000 (3.000000) | b: c0000000 (-2.000000) | product: c0c00000 (-6.000000)
# Time: 30 | a: c0400000 (-3.000000) | b: c0000000 (-2.000000) | product: 40c00000 (6.000000)
# Time: 40 | a: 3f800000 (1.000000) | b: 00000000 (0.000000) | product: 00000000 (0.000000)
# Time: 50 | a: 40fc0000 (7.875000) | b: 3e400000 (0.187500) | product: 3fbd0000 (1.476563)
# Test completed.
```

The results of the multiplication test are displayed which match the expected results.

4. Questions

1. What is rounding? When should rounding be performed?

Because floating points have limited precision, they can't represent every real number, so rounding to nearest floating-point number that can be represented is necessary. Rounding should be performed when it is essential to maintain a manageable precision and to prevent numbers from becoming too small or too large.

2. What is difference between single precision and double precision numbers

Single precision uses 32 bits to represent a floating-point number whereas double precision uses 64 bits. This makes double precision numbers suitable for representing numbers with higher precision and to higher accuracy of decimal digits.

5. Conclusions

In this lab, I learned how to represent floating points using IEEE 754 standard. I realize that operations using floating point numbers is more complicated than integer arithmetic and that is one reason sections of CPU is dedicated to floating point operations. When possible, programs using integer arithmetic are preferred to floating-point operations because of the better performance.