CS 367 Project 2 – Fall 2024:
All Value Reference Manual

1. ref_all_values Program

This is a helper program we wrote to show us all the possible values that are legal in the **MicroFP** library that you are writing. This tool will help you in debugging and testing by allowing you to check what should be expected vs. what your program is outputting.

```
...
  M = 1.906250 (b1.11101), val=3.81250000000000000 [0x09d]
  M = 1.937500 (b1.11110), val=3.87500000000000000 [0x09e]
  M = 1.968750 (b1.11111), val=3.93750000000000000 [0x09f]
E= 2 exp=(b101)
  M = 1.000000 (b1.00000), val=4.00000000000000000 [0x0a0]
  M = 1.031250 (b1.00001), val=4.12500000000000000 [0x0a1]
  M = 1.062500 (b1.00010), val=4.25000000000000000 [0x0a2]
...
```

There are a few different pieces of information in here. Let's start with the E line.

```
E= 2 exp=(b101)
```

This line says that the values that follow it are all of the values that would have an E of 2. These would be (in binary) 1.00000 * 2^2 up through 1.11111 * 2^2

We also get extra debugging information here. The (b101) shows us what the bits of **exp** field should look like in your MicroFP value. So, when E == 2, you have 101 for exp.

Now, let's look at one of the lines that start with M

```
  M = 1.031250 (b1.00001), val=4.12500000000000000 [0x0a1]
```

The M here shows the value of the Mantissa in Decimal for the given value. The more Important piece of information is the (b1.00001) that follows it. That shows you what the M in Binary would look like for this value:

b1.00001 Represents M = 1.00001
Whole Number = 1, Fraction = 0.00001

Next is the value itself, listed as **val=**. In this case, the value is 4.125

Finally, you have the [0x0a1] field. This shows you what the hex value should be for the entire **MicroFP** encoding for this value.

Copyright: Prof. Kevin Andrea, George Mason University, 2024. All Rights Reserved

So, if we enter **4.125** in MUAN, we should expect to return 0x0a1 from **toMicroFP()**

We can verify this in MUAN using the **display()** function. Remember that display will call your **toMicroFP()** function to convert the number into the MicroFP value, but then will show you exactly what that function returns.

Here's a sample of this in MUAN:

```
¯\_(ツ)_/¯ $ display(4.125)
MicroFP Value in Binary: 0 101 00001 (0x0a1)
```

So when we displayed 4.125, we see toMicroFP returned **0 101 00001 (0x0a1)**

Looking at ref_all_values, we were expecting **101** for the exp and **00001** for the frac.

```
010100001 = 0 101 00001
            S exp  frac
```

We can also see that this matches [0x0a1], so we know we did our function right.

2. Using the ref_all_values Program

Let's say we assign **1.45** to **foo**. We don't know what to expect when we run **MUAN** because there are thousands of possible values; it's not possible to quickly know which of them are possible and which needed to be rounded.

So, we can use the ref_all_values program to check for us! Run ref_all_values and look for the closest Values (**val**) to what you want.

```
Mantissa  (Mantissa in Binary)   Value in Decimal  [microfp_s in Hex]
...
  M = 1.437500 (b1.01110), val=1.437500000000000000 [0x06e]
  M = 1.468750 (b1.01111), val=1.468750000000000000 [0x06f]
...
```

The closest values to **1.45** are **1.4375** and **1.46875**. So, we can see that 1.45 isn't a valid value; now we need to know how it should be rounded. Since we're rounding using **Round to Zero (truncation)**, we will always round to the lower fraction value. **1.4375**

Now we can check MUAN and see if we are getting the correct answer.
When we print 1.45, we should be getting 1.4375.

```
¯\_(ツ)_/¯ $ print(1.45)
Value = 1.4375
¯\_(ツ)_/¯ $
```


Copyright: Prof. Kevin Andrea, George Mason University, 2024. All Rights Reserved

3. Webpage Reference

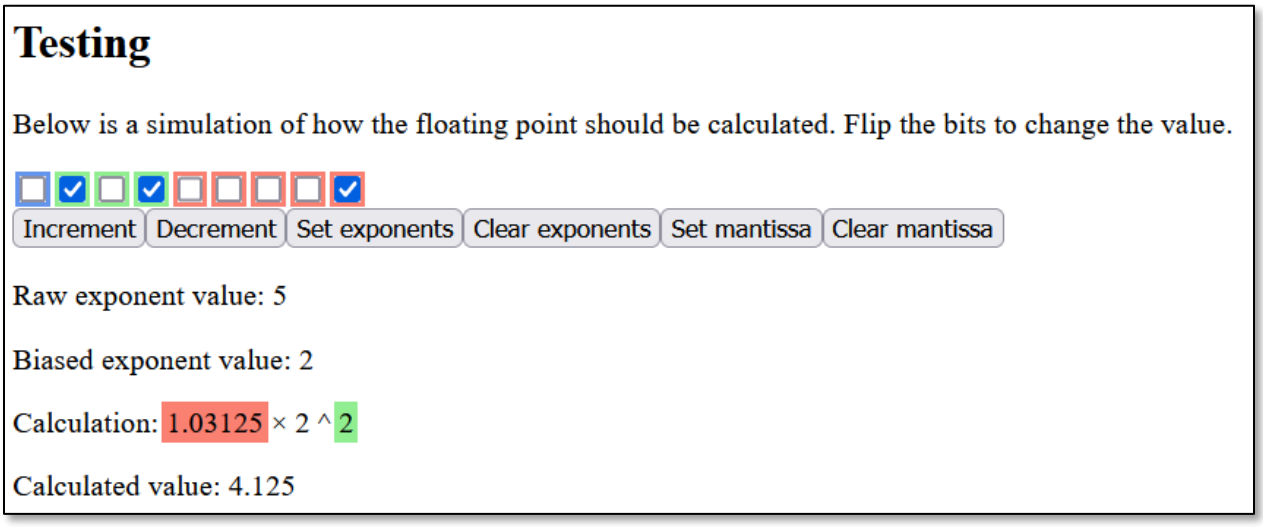
One of our GTAs, Andrew Hartman, has written an online tool that you can play with to see how the bits in a Floating-Point encoding affect the overall value.

As an example, we can see this entry from ref_all_values:

```
...
E= 2 exp=(b101)
...
   M = 1.031250 (b1.00001), val=4.125000000000000000 [0x0a1]
```

This tells us that if our **exp** is **101** and our **frac** is **00001**, then value will be **4.125**

We can do the same using this tool by manually selecting each of the bits, as shown here:



You can play with this interface on the webpage to see how each bit pattern will affect the overall calculated final value.

To use this, go to: https://cs.gmu.edu/~kandrea/F24/p2_calc.html

It will ask the floating point's width in bits. Keep the default 9 for this project and click Go. Then it will ask for the exponent width in bits. Keep the default 3 for this project and click Go.

Then you will see the interface at the bottom of the page.

You can also compare Hex values by looking at each bit as 1 or 0. So, for the above entry, it would be **0 101 00001**, which is 000**0 1010 0001** in groups of 4 bits.

0000 1010 0001 = **0x0a1**, which is exactly what the **ref_all_values** entry says it should be too.

Copyright: Prof. Kevin Andrea, George Mason University, 2024. All Rights Reserved

Changelog

v1.0: Sep 19: Release (Build: **2910c72a** in develop)

Copyright: Prof. Kevin Andrea, George Mason University, 2024. All Rights Reserved