Implementation of IEEE 754 using Lists

Kapil Aggarwal (16305R010)
Hareesh Kumar(16305R013)

IIT Bombay

November 22, 2017

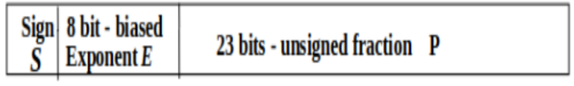
Problem Statement

IEEE 754 standard specifies interchange and arithmetic formats and methods for binary and decimal floating-point arithmetic in computer programming environments. An implementation of a floating-point system conforming to this standard may be realized entirely in software, entirely in hardware, or in any combination of software and hardware ([1]). The goal of this project is to develop a application capable of doing the floating point arithmetic in Haskell using Lists.

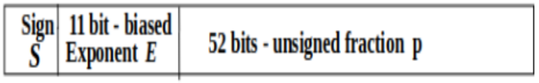
IEEE 754

- IEEE 754 is standard for Floating-Point Arithmetic.
- Represents Floating-Point numbers in 2 formats.

Figure: Floating Point Representation in IEEE 754



(a) IEEE single precision data format



(b) IEEE double precision data format

Single and Double Precision data formats in IEEE 754

Floating points number are represented in normalized format i.e.

$$-1^{S} \times (1.0 + 0.M) \times 2_{E-bias}$$

where
M is mantissa
E is exponent
S is sign bit.

Implementation

We begin the implementation of the application by first converting the input numbers that are in decimal format into the following format

(sign, exponent, mantissa).

After that we start implementing the arithmetic operations. We have implemented the following 3 arithmetic operations as functions in our project.

1. Addition
2. Subtraction
3. Multiplication

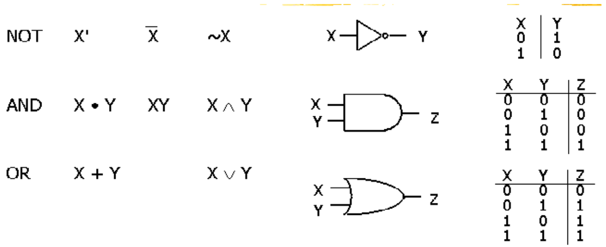
Above functions take 2 operand and return the results in (sign, exponent, mantissa) format. We have to convert it back to the decimal format.

Implementation Details - I

In the microprocessor the operation are implemented using gates. So we started the implementation by first designing following basic gates in boolean algebra as these are basic building blocks of any boolean circuit.

1. AND
2. OR
3. XOR
4. NOT

Figure: Boolean Gates



X	Y
0	1
1	0

X	Y	Z
0	0	0
0	1	0
1	0	0
1	1	1

X	Y	Z
0	0	0
0	1	1
1	0	1
1	1	1

Implementation Details - II

After implementing the basic boolean circuits , we have to implement the following arithmetic circuit using the basic gates.

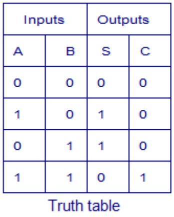
1. Half ADDER
2. Full ADDER
3. Half SUBTRACTOR
4. Full SUBTRACTOR
5. Multiplier

Since the above implemented circuits are capable of performing only single bit operations , we have to implement the circuits that can perform n-bit calculations using single bit operations.

Implementation Details - III

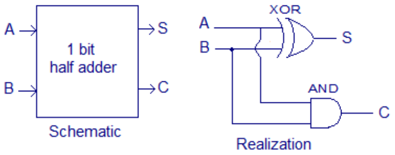
Lets have a look at how above circuits are implemented using boolean get Half adder circuit are implemented.

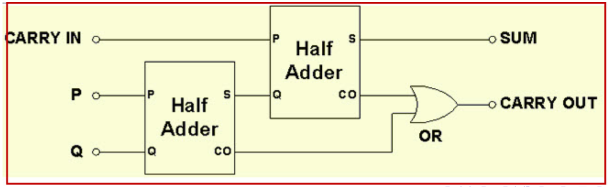
Figure: Half Adder



Inputs		Outputs	
A	B	S	C
0	0	0	0
1	0	1	0
0	1	1	0
1	1	0	1

Truth table





Algorithms

Addition

1. First, convert the two representations to scientific notation. Thus, we explicitly represent the hidden 1.
2. In order to add, we need the exponents of the two numbers to be the same. We do this by rewriting Y. This will result in Y being not normalized, but value is equivalent to the normalized Y.
3. Add $x - y$ to Y's exponent. Shift the radix point of the mantissa Y left by $x - y$ to compensate for the change in exponent.
4. Add the two mantissas of X and the adjusted Y together.
5. If the sum in the previous step does not have a single bit of value 1, left of the radix point, then adjust the radix point and exponent until it does.
6. Convert back to the normalized floating point representation.

Algorithms

Substraction

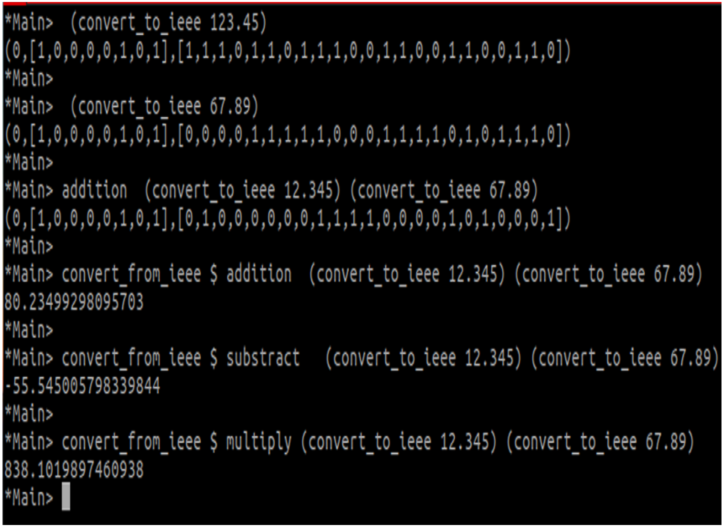
1. First, convert the two representations to scientific notation. Thus, we explicitly represent the hidden 1.
2. In order to add, we need the exponents of the two numbers to be the same. We do this by rewriting Y. This will result in Y being not normalized, but value is equivalent to the normalized Y.
3. Add $x - y$ to Y's exponent. Shift the radix point of the mantissa Y left by $x - y$ to compensate for the change in exponent.
4. Subtracts the two mantissas of X and the adjusted Y together.
5. If the difference in the previous step does not have a single bit of value 1, left of the radix point, then adjust the radix point and exponent until it does.
6. Convert back to the normalized floating point representation.

Algorithms

Multiplication

1. First, convert the two representations to scientific notation. Thus, we explicitly represent the hidden 1.
2. Let x be the exponent of X. Let y be the exponent of Y. The resulting exponent (call it z) is the sum of the two exponents. z may need to be adjusted after the next step.
3. Multiply the mantissa of X to the mantissa of Y. Call this result m.
4. If m is does not have a single 1 left of the radix point, then adjust the radix point so it does, and adjust the exponent z to compensate.
5. Add the sign bits, mod 2, to get the sign of the resulting multiplication.
6. Convert back to the one byte floating point representation, truncating bits if needed.

Working Example



```
*Main>  (convert_to_ieee 123.45)
(0,[1,0,0,0,0,1,0,1],[1,1,1,0,1,1,0,1,1,1,0,0,1,1,0,0,1,1,0,0,1,1,0])
*Main>
*Main>  (convert_to_ieee 67.89)
(0,[1,0,0,0,0,1,0,1],[0,0,0,0,1,1,1,1,1,0,0,0,1,1,1,1,0,1,0,1,1,1,0])
*Main>
*Main> addition  (convert_to_ieee 12.345) (convert_to_ieee 67.89)
(0,[1,0,0,0,0,1,0,1],[0,1,0,0,0,0,0,0,1,1,1,1,0,0,0,0,1,0,1,0,0,0,1])
*Main>
*Main> convert_from_ieee $ addition  (convert_to_ieee 12.345) (convert_to_ieee 67.89)
80.23499298095703
*Main>
*Main> convert_from_ieee $ substract   (convert_to_ieee 12.345) (convert_to_ieee 67.89)
-55.545005798339844
*Main>
*Main> convert_from_ieee $ multiply (convert_to_ieee 12.345) (convert_to_ieee 67.89)
838.1019897460938
*Main> █
```

Table: IEEE 754 - Arithmetic Operation

Conclusion

By using the algorithm of various operations on the input parameters , we can get the results that are similar to actual results of the various operations. So the application is able to do arithmetic operations on the inputs operands as per IEEE standards for floating points operations.

References

754-2008 - ieee standard for floating-point arithmetic.

The End