

Understanding Fixed Point Representation

Objective

To do computations involving real numbers using integer arithmetic

Representation

Represent real numbers as integers, scaled by an appropriate negative power of 2.

Example

Assume you are using 8-bits to represent real numbers. You would like to devote 4 bits to the whole part, and 4 bits for the fractional part.

- 1.5 can be represented as $24 * \frac{1}{16}$. The scale factor is $2^{-4} = \frac{1}{16}$. The number is simply represented as the integer 24, i.e. 00011000, with an implied binary point. One can think of the number as 0001.1000, that is 1.5.
- 2.25 can similarly be represented as $36 * \frac{1}{16}$, with an internal representation of 00100100, and an implied binary point. One can think of the number as 0010.0100, that is 2.25.

Doing Arithmetic

Addition

$1.5 + 2.25$ is implemented as $2^{-4} * 24 + 2^{-4} * 36$. Of course, the hardware simply adds $24+36$ to yield 60 which is interpreted with a scale factor of 2^{-4} as 0011.1100, which is 3.75.

Subtraction

$2.25 - 1.5$ is implemented as $2^{-4} * 36 - 2^{-4} * 24$. The hardware simply performs $36-24 = 12$, which is interpreted with a scale factor of 2^{-4} as 0000.1100, which is 0.75.

Multiplication

$2.25 * 1.5$ is implemented as $36 * 24 = 864$. The result, 0000001101100000, is a 16-bit number with a scale factor of 2^{-8} (Why?). The scale factor is adjusted to 2^{-4} by **right shifting** the result by 4 bits, and the result is stored in an 8-bit field as 00110110, which is interpreted as 3.375.

Division

$\frac{2.25}{1.5}$ can be implemented as $\frac{36}{24}$ with a result of 1, that is 00000001, and a scale factor of 2^0 . Adjust the scale factor by shifting the **quotient left** by 4 bits to yield 00010000, which the 4-4 representation for 1.0 and is obviously imprecise.

Precision can be improved by shifting the dividend to the left before division. Since integer division is implemented by using a 2n bit dividend and an n bit divisor, the division can be implemented by pre-shifting the dividend 4-bits to the left. Thus, the dividend has a value of $36 * 2^4$ with a scale factor of 2^{-8} , and is represented using 16-bits. The division is performed as $36 * 2^4$ divided by 24, which is $576/24$ which results in the value 24. Since the result has a scale factor of 2^{-4} , it can be interpreted as 0001.1000, which is 1.5.

Computing $\frac{1}{6}$ in the 4-4 format

Consider computing $\frac{1}{6}$ in the 4-4 format. It would be implemented as $\frac{16}{96}$, with both numerator and denominator having scale factors of 2^{-4} . Pre-shifting the dividend four bits to the left results in a 16-bit dividend with a value of $16 * 2^4$, that is 256 with a scale factor of 2^{-8} . The division proceeds as $256/96$, and the result is 2 with a scale factor of 2^{-4} . This is interpreted as 0000.0010, that is 0.125, which is obviously not very accurate.

Can you gain precision by pre-shifting the dividend on your own?

Assume you pre-shift the dividend left by 2 bits. The dividend is now 64 with a scale factor of 2^{-6} . The divisor is still 96 with a scale factor of 2^{-4} . The hardware left shifts the dividend four bits (as before) before performing the division as $\frac{64 * 2^4}{96}$ with a quotient of 10, that is 00001010. Note that since the dividend had a scale factor of 2^{-10} , and the divisor had a scale factor of 2^{-4} , the quotient has a scale factor of 2^{-6} . Adjust the scale factor to 2^{-4} by **right shifting** the quotient by 2 bits to yield 0000.0010. The result is 0.125, the same as before.

What if the result could be stored in a 16-bit field ?

We proceed exactly as before and the quotient is 10 with a scale factor of 2^{-6} . Since the result is to be stored in a 16-bit field with an 8-8 format, the scale factor needs to be 2^{-8} . Therefore we adjust the scale factor by **shifting the quotient left** by 2 bits to yield 00101000 with a scale factor of 2^{-8} . This is interpreted as 0.00101000 which represents 0.15625 and is closer to the actual value of $\frac{1}{6}$.