

The Logarithm Algorithm

By Clay S. Turner

Feb 9, 2010

This is a simple algorithm¹ for finding the radix two logarithm that is amenable to hardware/ assembly language implementations. To wit, we wish to find

$$y = \log_2(x) \tag{1}$$

Thus we can invert this and find

$$x = 2^y \tag{2}$$

Let's constrain x to be in the interval $[1, 2)$.

A simple series of divides/multiplies by two will put x into the proper range. The count of the divides/multiplies gives the characteristic of the logarithm. This gets added to the mantissa (calculated below) to form the complete logarithm. The number of divides is taken as a positive number and the number of multiplies is taken as a negative number.

Now with x properly scaled, then we know from (1) that $0 \leq y < 1$. So let's expand y into a binary series, thus

$$y = y_1 \cdot 2^{-1} + y_2 \cdot 2^{-2} + y_3 \cdot 2^{-3} + y_4 \cdot 2^{-4} + \dots \tag{3}$$

It will be efficacious to put this into nested parenthetical form.

$$y = 2^{-1}(y_1 + 2^{-1}(y_2 + 2^{-1}(y_3 + 2^{-1}(y_4 + \dots)))) \tag{4}$$

So putting (4) into (2) we have

$$x = 2^{2^{-1}(y_1 + 2^{-1}(y_2 + 2^{-1}(y_3 + 2^{-1}(y_4 + \dots))))} \tag{5}$$

Now we are ready to see how to sequentially extract the bits comprising y .

Step 1: Square x

$$x^2 = 2^{y_1} \cdot 2^{2^{-1}(y_2 + 2^{-1}(y_3 + 2^{-1}(y_4 + \dots)))} \tag{6}$$

We see the right hand side of (6) is a product of two factors, where the left factor is equal to either 1 or 2 depending on the value of y_1 . Thus we have the following two cases:

¹ The author first worked out this algorithm back in the 1970s after he saw a paper that showed how to calculate integral exponentiation (al Kashi's method) by repeated squaring and multiplying.

$$y_1 = 0: \quad x^2 = 2^{2^{-1}(y_2 + 2^{-1}(y_3 + 2^{-1}(y_4 + \dots)))} \quad (7)$$

Or

$$y_1 = 1: \quad x^2 = 2 \cdot 2^{2^{-1}(y_2 + 2^{-1}(y_3 + 2^{-1}(y_4 + \dots)))} \quad (8)$$

Since both (7) and (8) contain the common factor $2^{2^{-1}(y_2 + 2^{-1}(y_3 + 2^{-1}(y_4 + \dots)))}$, and this factor will have a value that is greater than or equal to 1 and less than 2, we see the square of x will be greater than or equal to 2 if and only if $y_1 = 1$, otherwise $y_1 = 0$.

Step 2: Compare result of squaring and if greater than or equal to two, then set mantissa bit to “1” and divide by 2. Otherwise set mantissa bit to “0.”

So now at the end of step 2, we have

$$x^2 = 2^{2^{-1}(y_2 + 2^{-1}(y_3 + 2^{-1}(y_4 + \dots)))} \quad (9)$$

If we realize x^2 is just a number, then we see that (9) has the exact same form as (5). Thus we can repeat steps 1 and 2 to extract out all of the bits in y .

For other radices, just use $\log_a(x) = \frac{\log_2(x)}{\log_2(a)}$

The following is “c” program to demonstrate the mantissal portion of the algorithm.

```
// Sample binary logarithm routine by Clay S. Turner
//
// Calculates binary logarithm of x assuming 1 <= x < 2
// Algo is designed to operate efficiently with binary fixed point
// arithmetic
// This version uses floating point for illustrating the algo.

#define PRECISION 10 // # of bits the log will be calculated to

double BinLog(double x)
{
    int p=0,q=1,i;
    for (i=0;i<PRECISION;i++) {
        p<<=1;
        q<<=1;
        x=x*x;
        if (x>=2.0) {
            x/=2.0;
            p|=1;
        }
    }
    return (double)p / q;
}
```