Cairo University
Electronics and Communications Department

Computer Arithmetic:
Tables and series for many functions

Hossam A. H. Fahmy

How do we divide?

Three basic approaches are in use:

1. *Table lookup.*
2. Subtractive methods: (digit recurrence, converge linearly)
   (a) Restoring
   (b) Non-restoring
   (c) Shift over 0's
   (d) Brute force (multiple subtractors)
   (e) SRT
   (f) *High radix*
3. Multiplicative methods: (converge quadratically)
   (a) Newton-Raphson
   (b) Series expansion
   (c) *Higher order series*

The series expansion of a function

In general,

$$f(x_0+\Delta x) = f(x_0) + \Delta x \left.\frac{df(x)}{dx}\right|_{x_0} + \frac{(\Delta x)^2}{2!}\left.\frac{d^2f(x)}{dx^2}\right|_{x_0} + \frac{(\Delta x)^3}{3!}\left.\frac{d^3f(x)}{dx^3}\right|_{x_0} + \cdots.$$

For the reciprocal of b where $b = b_h + b_l$ we get:

$$\frac{1}{b} = \frac{1}{b_h} - b_l\left(\frac{1}{b_h}\right)^2 + b_l^2\left(\frac{1}{b_h}\right)^3 + \cdots.$$

A simple approach first

An interpolation table contains the approximate values of $\frac{1}{b_h}$. The hardware uses b_h to read two consecutive values and calculated the reciprocal as:

$$\frac{1}{b} = \frac{1}{b_h} - b_l\left(\frac{1}{b_h} - \frac{1}{b_h + 1ulp}\right)$$

Hence, with just a table and an adder we get a division. This is fast!

How good is interpolation

- For an n bit operand, the table has about $2^{\frac{n}{2}}$ entries depending on how many bits there is in b_h and b_l.
- While discussing multiplicative division, we found that the accuracy of the result from the table depends on how many bits are used to index it. Hence, with only $\frac{n}{2}$ input bits, we get only about $\frac{n}{2}$ accurate output bits.

This approach is useful mainly with short precisions and when the accuracy of the results is not very critical. (example: 3D graphics).

Bipartite tables

- uses two tables to get two approximations: the first term and the second terms of the reciprocal expansion ($\frac{1}{b} \approx \frac{1}{b_h} - b_l \left(\frac{1}{b_h}\right)^2$).
- divides the operand b into three parts: | b_1 | b_2 | b_3 |.
- indexes the first table with $b_1 + b_2$ and the second table with $b_1 + b_3$. (b_3 defines the derivative in the region of b_1.)

The bipartite is more accurate than the interpolation but with more hardware.

High radix division

In SRT, we produce 2 or 3 bits in each iteration. The high radix algorithms (Wong 1992) are able to produce about 14 bits per iteration.

The first algorithm uses the m most significant bits of b to get $\frac{1}{b_h}$ from a table then:

$$\begin{aligned} a' &= a - a_h \frac{1}{b_h} b \\ q' &= q + \frac{a_h}{b_h \times 2^{j-k}} \end{aligned}$$

(b_h here is slighlty different from the earlier definition and $j - k$ is a shift amount to correctly align the quotient bits.)

- With an m bit lookup, we get $m - 2$ bits per iteration.
- We can use a redundant format to keep the dividend and quotient.

Second high radix algorithm

The second algorithm uses the m most significant bits of b to index several tables and get, simultaneously, $\frac{1}{b_h}$, $\frac{1}{b_h^2}$, $\frac{1}{b_h^3}$, ... then calculate

$$B = \frac{1}{b_h} - \frac{\Delta b}{b_h^2} + \frac{(\Delta b)^2}{b_h^3} - \frac{(\Delta b)^3}{b_h^4} + \cdots$$

The new dividend and quotient are calculated as:

$$\begin{aligned} a' &= a - a_h B b \\ q' &= q + a_h B \frac{1}{2^{j-k}} \end{aligned}$$

With an m bit lookup and t terms in the expansion, we get $(mt - t - 1)$ bits per iteration.

Reduce the number of tables

The third algorithm combines the first two terms of the expansion together and requires *one* table.

$$\begin{aligned}\frac{a}{b} &= \frac{a}{b_h + b_l} \\ &= \frac{a}{b_h}\left(1 - \left(\frac{b_l}{b_h}\right) + \left(\frac{b_l}{b_h}\right)^2 - \left(\frac{b_l}{b_h}\right)^3 + \cdots\right) \\ &\approx \frac{a(b_h - b_l)}{b_h^2}\end{aligned}$$

- While looking up the table to find out $\frac{1}{b_h^2}$, multiply $a(b_h - b_l)$. With one more multiplication, the result is ready.
- With an m bit lookup, we get $\approx 2m$ bits per iteration.

Higher order series

So far, we only considered the Newton-Raphson iteration of the first order with a quadratic convergence:

$$0 \approx f(x_i) + (x_{i+1} - x_i)f'(x_i)$$

Higher order series yield faster convergence but require the parallel calculation of the square, cube, and higher powers of the operand.

A look at the expansions

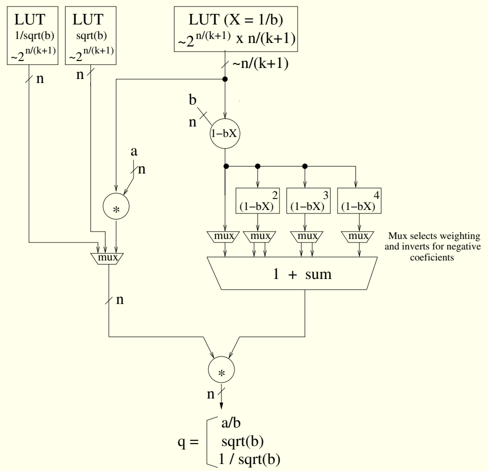
With $d = 1 - bx_0$ and $x_0 \approx \frac{1}{b}$, $y_0 \approx \frac{1}{\sqrt{b}}$, and $z_0 \approx \sqrt{b}$ then:

Reciprocal :$\frac{1}{b} = x_0(1 + d + d^2 + d^3 + \cdots)$

Square root :$\sqrt{b} = y_0(1 - \frac{1}{2}d - \frac{1}{8}d^2 - \frac{1}{16}d^3 - \frac{15}{128}d^4 - \cdots)$

Reciprocal square root :$\frac{1}{\sqrt{b}} = z_0(1 + \frac{1}{2}d + \frac{3}{8}d^2 + \frac{5}{16}d^3 + \frac{35}{128}d^4 + \cdots)$

A general purpose unit



The unit calculates the powers of $(1 - bx_0)$ in parallel.

Elementary functions expansions

$$
\begin{aligned}
e^x &= 1 + x + \frac{1}{2}x^2 + \frac{1}{6}x^3 + \cdots \\
\ln(1+x) &= x - \frac{1}{2}x^2 + \frac{1}{3}x^3 - \frac{1}{4}x^4 + \cdots \\
\cos(x) &= 1 - \frac{1}{2}x^2 + \frac{1}{24}x^4 - \cdots \\
\sin(x) &= x - \frac{1}{6}x^3 + \frac{1}{120}x^5 - \cdots
\end{aligned}
$$

With parallel powering units, it is possible to build a fast and accurate general unit.

A parallel squaring unit

					a_5	a_4	a_3	a_2	a_1	a_0
×					a_5	a_4	a_3	a_2	a_1	a_0
					a_5a_0	a_4a_0	a_3a_0	a_2a_0	a_1a_0	a_0
				a_5a_1	a_4a_1	a_3a_1	a_2a_1	a_1	a_0a_1	
			a_5a_2	a_4a_2	a_3a_2	a_2	a_1a_2	a_0a_2		
		a_5a_3	a_4a_3	a_3	a_2a_3	a_1a_3	a_0a_3			
	a_5a_4	a_4	a_3a_4	a_2a_4	a_1a_4	a_0a_4				
a_5	a_4a_5	a_3a_5	a_2a_5	a_1a_5	a_0a_5					
a_5a_4	a_5a_3	a_5a_2	a_5a_1	a_5a_0	a_4a_0	a_3a_0	a_2a_0	a_1a_0		a_0
a_5		a_4a_3	a_4a_2	a_4a_1	a_3a_1	a_2a_1		a_1		
		a_4		a_3a_2		a_2				
				a_3						

With bit manipulations, we reach a unit much smaller than a direct multiplier.

A parallel cubing unit

							a_3	a_2	a_1	a_0
×							a_3	a_2	a_1	a_0
×							a_3	a_2	a_1	a_0
							$a_3a_0a_0$	$a_2a_0a_0$	$a_1a_0a_0$	$a_0a_0a_0$
						$a_3a_0a_1$	$a_2a_0a_1$	$a_1a_0a_1$	$a_0a_0a_1$	
						$a_3a_1a_0$	$a_2a_1a_0$	$a_1a_1a_0$	$a_0a_1a_0$	
					$a_3a_2a_0$	$a_2a_2a_0$	$a_1a_2a_0$	$a_0a_2a_0$		
					$a_3a_1a_1$	$a_2a_1a_1$	$a_1a_1a_1$	$a_0a_1a_1$		
					$a_3a_0a_2$	$a_2a_0a_2$	$a_1a_0a_2$	$a_0a_0a_2$		
				$a_3a_3a_0$	$a_2a_3a_0$	$a_1a_3a_0$	$a_0a_3a_0$			
				$a_3a_2a_1$	$a_2a_2a_1$	$a_1a_2a_1$	$a_0a_2a_1$			
				$a_3a_1a_2$	$a_2a_1a_2$	$a_1a_1a_2$	$a_0a_1a_2$			
				$a_3a_0a_3$	$a_2a_0a_3$	$a_1a_0a_3$	$a_0a_0a_3$			
			$a_3a_3a_1$	$a_2a_3a_1$	$a_1a_3a_1$	$a_0a_3a_1$				
			$a_3a_2a_2$	$a_2a_2a_2$	$a_1a_2a_2$	$a_0a_2a_2$				
			$a_3a_1a_3$	$a_2a_1a_3$	$a_1a_1a_3$	$a_0a_1a_3$				
		$a_3a_3a_2$	$a_2a_3a_2$	$a_1a_3a_2$	$a_0a_3a_2$					
		$a_3a_2a_3$	$a_2a_2a_3$	$a_1a_2a_3$	$a_0a_2a_3$					
	$a_3a_3a_3$	$a_2a_3a_3$	$a_1a_3a_3$	$a_0a_3a_3$						
1×	a_3			a_2			a_1			a_0
3×		a_3a_2	a_3a_1	a_3a_0	a_3a_1	a_2a_0	a_3a_0	a_2a_0	a_1a_0	
			a_3a_2	$a_3a_2a_0$	a_2a_1	a_2a_1		a_1a_0		
3×			$a_3a_2a_1$		$a_3a_1a_0$	$a_2a_1a_0$				

The unit sums the 3× terms together and reduces them to a carry and sum vectors. Then it reduces those with the 1× terms and a final CPA gives the result.

Truncation

In the series, each higher order power is multiplied by a smaller constant.

- Only the most significant part of the square, cube, or higher power is needed.
- For a single precision, the needed part of the cube PPA is 8 bits wide and 12 bits high. This is less than 10% of a direct multiply!
- The squaring unit can be truncated too.
- A detailed analysis tells you how much to truncate from each power term to keep the total error term within the accepted bounds.

Conclusions about division and elementary functions

- For a high speed and high accuracy double precision, the required time delay is that of a lookup table, two multiplications, and one addition.
- Such a unit may be pipelined into just four cycles.
- The hardware cost of such a unit is not very large.