Fast Evaluation of the Elementary Functions in Single Precision PhL 10.12.04

A 1995 paper by Wong and Goto (Singapore and Japan, IEEE Trans Computers).

The starting point is to represent a number X as shown in (1)

$$X = x_0 + \lambda x_1 + \lambda^2 x_2 + \lambda^3 x_3 \qquad (1)$$

where X will generally be regarded as the mantissa which is in the range 1.00000.. to 1.111111... or (1,2). The number λ is a constant $\lambda = 2^{-6}$.

How do we know you can expand ANY number in this manner? Well, you can think of x_0 as the first 6 bits, then x_1 as the next 6 bits, and so on, and we are thinking about 24 bit numbers as in single-precision FP. You can rewrite the above in this notation: [see also their block diagram!]

$$X = x_0 + (x_1 >> 6) + (x_2 >> 12) + (x_3 >> 18) \qquad \text{// so } x_0 \text{ is the MS 6-bits} \qquad (1')$$

For now, think of all four x_i as 6-bit numbers. Then (1) is just a statement that you can write a 24-bit number as 4 sets of 6 bits each, so no need to "prove" expansion (1). For now I will think of all 24 bits as being after the binary point.

Formula (2) is a Taylor series around the point $X_0 \equiv x_0 + (x_1 >> 6)$. I accept that the result can be written in the form (4) to single-precision accuracy (ie, I could verify it if I wanted).

Now let's jump to the hardware block diagram on page 456. They propose look up ROM's with 12 bits of input, which means you can use this as your input:

$$X_0 = x_0 + \lambda x_1 = x_0 + (x_1 >> 6)$$

In other words, you just use the MS 12 bits of X for your lookup in the top table. So the top line produces the first term in (4). I guess the 30 bits output is needed so you have 24 bits of accuracy after all the additions are done.

To get the *second* term, you have to add and subtract λx_2 from X_0 which is the role of the + and - adder boxes in the picture, and now we have 13 bits instead of 12 coming out, as usual. Then the two tables (row2 and row3) compute the two terms needed. These two terms are not immediately subtracted but are fed along with the first term into a Wallace Tree adder that is just a 6-input adder which seems to have a 30 bit input width. I think the shifts shown as 2^{-5} should really be 2^{-7} , a possible paper mistake, looking at (4).

A similar thing is done to get the next two terms in (4), and I would again argue for 2^{-13} in the final shifters, not 2^{-11}. Notice that so far, all 5 lookup tables are the same table! But for parallel computation, you need 5 of these identical tables.

The last term in (4) is a little different. You program this table with the functional combination shown, and you see that it is a function of the two 6-bit numbers x_0 and x_2. So we a 6th table with the same number 12 bits of input. Since the output here is shifted by $\lambda^4 = (>>24)$, this table only needs to put out 6 bits. Similarly, the other tables need less than 30 bits for this same reason. But I think many of these numbers are off based on comments above.

OK, probably my 1/2 errors have to do with what X represents, where the binary point is, etc. The main point is the block diagram and (4). You have four adder/subtractors in the first column, then you have 6 tables in the second, and you end with adders, so they call it their ATA method, fine. The first

table is a RAM with 12 address lines and 30 bits wide, so $30*2^{12} = 30*4K = 120K$ bits, and 6 of these is about 720K bits. If you want to have set-in tables for all the elementary functions at once time, they estimate about 14M bits.

The motivation for the paper is that memory was becoming cheaper in 1995 and it seemed a good idea to use it. I have no idea whether this ATA method has been used by anyone! I also have no idea what the "usual" method is for computing elementary functions. Probably a Taylor series around a normalized expansion point, where you do lots of FP multiplies and adds. You need this complexity even if you have a FP hardware unit of course, which would be IEEE.