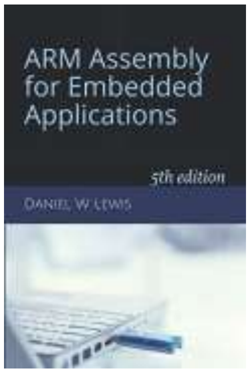


Programming Lab 9D

Extending Precision

Topics: Floating-pt. arithmetic, series approximations, Kahan summation, double floats

Click to download Lab9D-Main.c

Prerequisite Reading: Chapters 1-9
Revised: February 15, 2021

Background: The 24-bit significand of single-precision floats supports 6 to 9 significant decimal digits of precision. What this means is (1) if a decimal string with at most 6 significant digits is stored in a float, and then converted back to a decimal string with the same number of digits, the result should match the original string, and (2) if a float is converted to a decimal string with at least 9 significant digits, and then stored back in a float, the result must match the original number[1].

Unfortunately, the result of a sequence of arithmetic operations may have even fewer significant digits. That's because the representation error due to limited precision can accumulate. For example, simply summing n numbers in sequence has a worst-case error that grows proportional to n. To prevent this, Kahan[2] developed a novel summation technique that keeps a separate running compensation variable, effectively eliminating the cumulative effect.

Our processor only provides floating-point instructions for single-precision arithmetic; double-precision may be used when even more precision is required, but arithmetic operations are then emulated in software, thus increasing execution time by one to two orders of magnitude. When hardware support for single-precision floating-point is available, a much faster alternative known as "double float"[3] represents values using a *pair* of single-precision floats and achieves near double precision. The main program of this lab computes Nilakantha's infinite series approximation[4] to pi, comparing the accuracy and speed of using Kahan's technique with floats and doubles to that of a normal series approximation using double floats:

$$\pi = 3 + \frac{4}{2\times3\times4} - \frac{4}{4\times5\times6} + \frac{4}{6\times7\times8} - \frac{4}{8\times9\times10} + \cdots$$

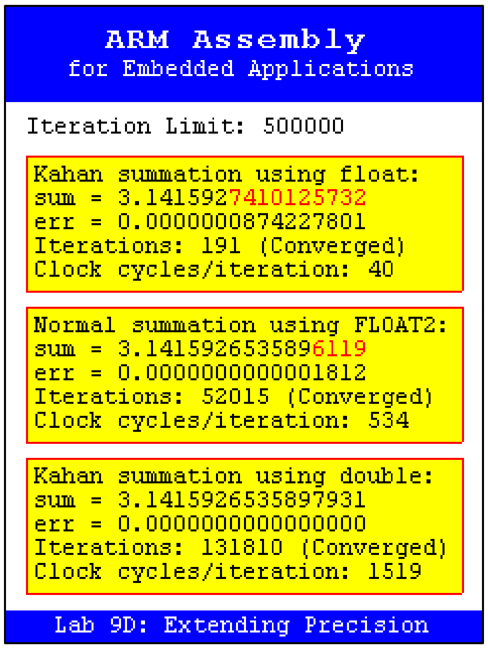
Assignment: The main program may be compiled and executed without writing any assembly. However, your task is to create faster assembly language replacements for the three functions listed below using their C versions to guide your implementation. These are low-level support functions that are used to implement the basic arithmetic operations on double floats. The original C versions are defined as "weak", so that the linker will automatically replace them in the executable image by those you create in assembly; you do not need to remove the C versions.

```
FLOAT2 QuickTwoSum(float a32, float b32) ;
FLOAT2 TwoSum(float a32, float b32) ;
FLOAT2 Split(float a32) ;
```

At the end of each function there is a return statement similar to:

```
return (FLOAT2) {x, y} ;
```

This statement returns a struct consisting of a pair of single-precision floats, with `x` in register `S0` and `y` in register `S1`. If your code is correct, the display should look similar to the image at right showing each function's accuracy and execution time as clock cycles per iteration. The displayed value of the sum is the function's series approximation to pi, with incorrect digits highlighted in red.



```
ARM Assembly
for Embedded Applications

Iteration Limit: 500000

Kahan summation using float:
sum = 3.1415927410125732
err = 0.0000000874227801
Iterations: 191 (Converged)
Clock cycles/iteration: 40

Normal summation using FLOAT2:
sum = 3.1415926535896119
err = 0.0000000000001812
Iterations: 52015 (Converged)
Clock cycles/iteration: 534

Kahan summation using double:
sum = 3.1415926535897931
err = 0.0000000000000000
Iterations: 131810 (Converged)
Clock cycles/iteration: 1519

Lab 9D: Extending Precision
```

[1] https://people.eecs.berkeley.edu/~wkahan/ieee754status/IEEE754.PDF
[2] https://en.wikipedia.org/wiki/Kahan_summation_algorithm
[3] http://andrewthall.org/papers/df64_qf128.pdf
[4] https://en.wikipedia.org/wiki/Pi