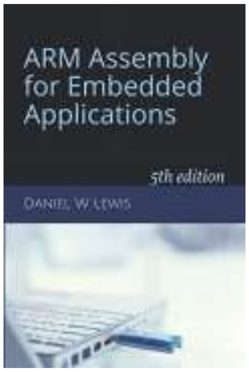


Programming Lab #8g

Q16 Fixed-Point Reciprocal

Topics: Q16 fixed-point arithmetic, reciprocal division, Newton's method to compute 1/d.

Prerequisite Reading: Chapters 1-11

Revised: December 15, 2020

Background: [1] Even though Q16 fixed-point reals are actually 32-bit integers, implementing fast Q16 fixed-point division on the Cortex-M4 processor is somewhat of a challenge. It cannot be done by simply using the integer divide instruction because the 32-bit Q16 dividend must be sign-extended to 64-bits and shifted left by 16 bits so that the imaginary binary point will be in the middle of the resulting quotient. Unfortunately, the `SDIV` instruction only supports a 32-bit dividend. Common solutions such as restoring, non-restoring or SRT algorithms tend to be slow, requiring loops with as many as 32 iterations – one for every bit in the divisor. Division may also be implemented using multiplication, but requires knowing the reciprocal of the divisor. When the divisor d is a constant, the reciprocal $1/d$ will also be a constant that may be precomputed by hand and inserted into the source code. However, when d is a variable, the reciprocal must be computed during execution. This can be done rather efficiently using Newton's method as described in the footnote reference[1].

To do: The main program (downloaded from here) uses Newton's method to compute and plot the reciprocal for $-1 \le d \le +1$. Rather than designed to reduce execution time, the implementation has been partitioned into several small functions to simplify your task, which is to replace each of the following C functions with assembly:

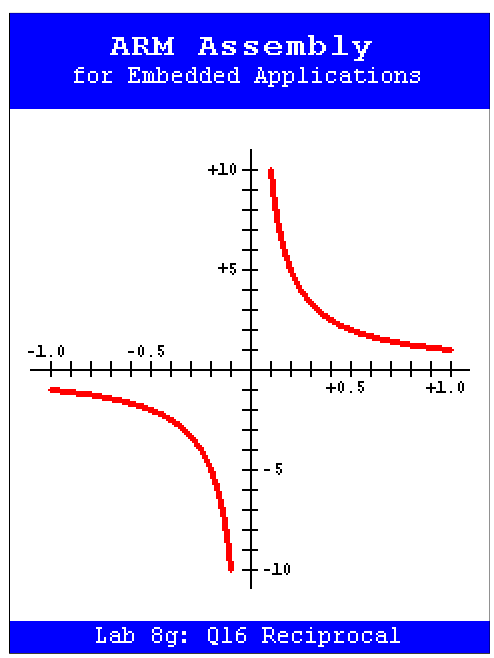
```
typedef uint32_t Q16 ;                          // only working here with d >= 0

Q16 Normalize(Q16 divisor, int zeros) ;         // normalizes the divisor to 0.5-1.0
Q16 Denormalize(Q16 estimate, int zeros) ;      // denormalizes Normalized(estimate)
Q16 NormalizedEstimate(Q16 divisor) ;           // initial estimate of 1/Normalized(d)
Q16 InitialEstimate(Q16 divisor) ;              // initial estimate of 1/d
Q16 Reciprocal(Q16 divisor) ;                   // computes the reciprocal: 1/divisor
```

Note: The C versions of these functions are defined as "weak" functions, which causes the linker to ignore the C version if you provide an assembly language function of the same name. E.g.,

```
Q16 __attribute__((weak)) Reciprocal(Q16 divisor)
    {
    . . .
    }
```

These five functions make use of two other functions that are not needed in assembly: (1) every call to `LeadingZeros` may be replaced by a single `CLZ` instruction, and (2) every call to `Q16Product` may be replaced by a simple `SMULL`, `LSR`, `ORR` instruction sequence. These are also weak functions, which causes the linker to eliminate them if not called.

Suggestion: Code and test your replacement functions one at a time in the order shown above. If your code is correct, the display should look like the figure shown on the right.



[1] https://en.wikipedia.org/wiki/Division_algorithm