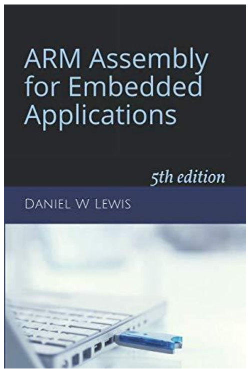


Programming Lab 11D

Q16 Fixed-Pt Reciprocal

Click to download Lab11D-Main.c

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

Prerequisite Reading: Chapters 1-11

Revised: March 5, 2021

Background: [1] Even though Q16 fixed-point reals are actually 32-bit integers, implementing a fast Q16 fixed-point division on the Cortex-M4 processor is somewhat of a challenge. It cannot be accomplished by simply using the integer divide instruction because the 32-bit Q16 dividend must be sign-extended to 64-bits and shifted left 16 bits so that the quotient's imaginary binary point will be in the middle of its 32-bit representation. Unfortunately, the `SDIV` instruction only supports a 32-bit dividend. Emulation functions based on the common 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. However, division may also be implemented by multiplying the dividend by the reciprocal of the divisor. When the divisor d is a constant, the reciprocal $1/d$ may be precomputed by hand and inserted into the source code. But when d is a variable, the reciprocal can only be computed during execution. Fortunately, this may be done rather efficiently using Newton's method as described in the footnote reference[1].

Assignment: The main program uses Newton's method to compute and plot the reciprocal for divisors in the range $-1 \le d \le +1$. The function `Reciprocal` listed below assumes the divisor is positive; for negative divisors, the program simply plots the value of $-(1/|d|)$. Rather than being implemented for maximum speed, the code has been partitioned into several small functions to simplify your task, which is to replace each of the following C functions with assembly.

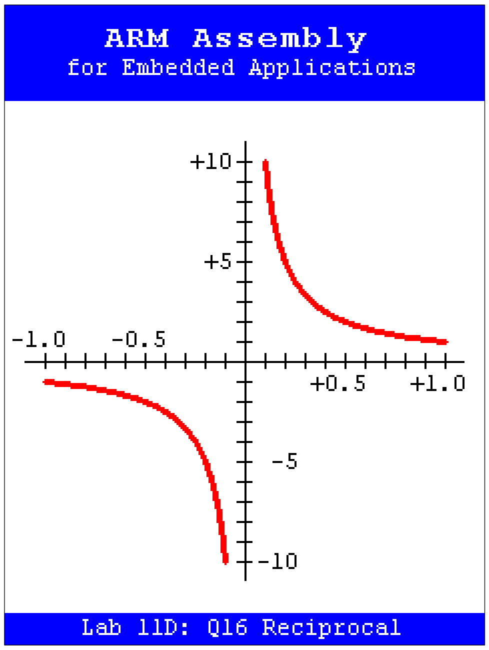
```
typedef uint32_t Q16 ;

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 reciprocal for divisor >= 0
```

Note: These functions use a macro called "`FIXED`" defined in the main program. This is *not* a function; do not implement it as a function in assembly. Each place where the macro is used, determine the constant it produces using a calculator and insert that constant into your code.

The main program includes C versions of these functions. You can compile and run the program as is without writing any assembly. However, your task is to create faster versions of these functions in assembly using the C versions to guide your implementation. The original C versions have been defined as "weak", so that the linker will automatically replace them in the executable image by those you create in assembly; there is no need to remove the C versions.

These five functions make use of two other functions that are not needed in assembly: (1) the 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 from the executable 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