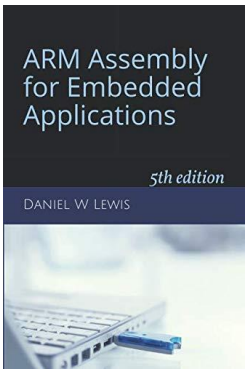

Programming Lab #4c

Linear Interpolation

Topics: Integer arithmetic, linear interpolation using integers, rounding.

Prerequisite Reading: Chapters 1-5
Revised: March 22, 2020

Background: The STM32F429ZI MCU includes an analog sensor whose output voltage changes linearly with the temperature of the chip and an internal analog-to-digital (A/D) converter that can be used to convert that voltage into a digital value. That value differs from one chip to another due to variations in the manufacturing process that affect: (1) an offset in the sensor output and (2) the value of an internal reference voltage (V_{ref}) that determines the gain of the A/D converter. (Note: A/D *gain* refers to the change in the digital output for a given change in input voltage.) To improve the accuracy of the temperature sensor (especially for absolute temperature measurement), calibration values are individually measured for each part during production test and stored in the system memory area. First, the reference voltage V_{ref} is set to 3.3 volts, measured by the A/D converter and recorded. Then with V_{ref} held at 3.3 volts, temperature measurements are taken at 30° and 110° Celcius, converted by the A/D converter and recorded.

When we run our program V_{ref} will not be 3.3 volts, but we can use the A/D converter to measure the actual value of V_{ref}. The ratio of the actual measurement to the calibrated measurement is a multiplier that we can use to scale the two calibrated temperature measurements into values that correspond to the *actual* reference voltage. With scaled A/D measurements of 30° and 110° C, we can calculate the slope (m) and offset (b) of the linear relationship ($y = mx + b$) between temperature and A/D measurements.

Assignment: Create a single ARM Cortex-M4 assembly source code file containing the linear interpolation function given below. This function is called several places in the main program (download from here) that displays the calibration values, the scaled calibration values, and the chip temperature as both raw A/D output and Celcius values. The program also displays a moving plot of temperature as shown.

```
int32_t MxPlusB(int32_t x, int32_t mtop, int32_t mbtm, int32_t b) ;
```

This function should return the value of the familiar linear function, $y = mx + b$. Except for the *displayed* value of the scale factor, all of the variables in the program are integers because they are derived from integer values returned by the A/D converter. This requires that the value of the slope (m) be represented as the *ratio* of two integers, `mtop` divided by `mbtm`, and thus your function must include an integer division. (Be careful with the order of evaluation or the result will be incorrect.)

Since integer division discards (truncates) the fractional part of the quotient, you will get better accuracy if the quotient is rounded. A common approach (for positive operands) is to add 0.5 to the quotient produced by real division and then copy the result into an integer (discarding the fractional part of the sum), as in:

```
int q = (int) (15.0/4.0 + 0.5) ; // 15/4 = 3.75 rounds to 4
```

However, your function will be using integer arithmetic, so we must rearrange the operands so that the addition occurs *before* the division, by adding one-half of the divisor to the dividend as in:

```
int quotient = (15 + 4/2)/4 ;      // (15 + 4/2)/4 → 17/4 → 4
```

Unfortunately, this code rounds in the wrong direction when the quotient is negative! Use the following solution that always rounds the integer quotient `a/b` correctly:

```
rounding = (((( (dvnd*dvsr) >> 31) * dvsr) << 1) + dvsr) / 2 ;
quotient = (dvnd + rounding) / dvsr ;
```

The "`>>`" shift operation must be implemented with an *arithmetic* shift right (ASR) instruction; this causes the value of the expression `(dvnd*dvsr) >> 31` to be `0` when the signs of `dvnd` and `dvsr` are identical, and `-1` when they differ. The magnitude of `rounding` will be `|dvsr/2|`, but its sign will be the same as that of `dvnd`. Adding it to `dvnd` thus increases the *magnitude* of `dvnd` before the integer division by `dvsr` truncates the quotient.