Daniel Christiansen

ECE 371

Project 1 Design Log

**11/08/16 - 15:30**

The Fahrenheit\_Temps array will have 24/1.5 = 16 entries, each one byte. So the array will look like FAHRENHEIT\_TEMPS: .byte 0x00, 0x41, 0x11, etc. The values will fall within the range 0x00 through 0x7D. The Celsius values will range from -18 through 52 in decimal, or 0xEE through 0x34.

**11/08/16 - 16:00**

Created a project file in CCS with a test array of Fahrenheit values that should cover most cases:

0x00, 0x02, 0x04, 0x10, 0x1D, 0x1F, 0x20, 0x21, 0x2C, 0x32, 0x3E, 0x49, 0x54, 0x5F, 0x6A, 0x7D in hex or…

0, 2, 4, 16, 29, 31, 32, 33, 44, 50, 62, 73, 84, 95, 106, 125, in decimal.

This should convert to (after rounding):

-18, -17, -16, -9, -2, -1, 0, 1, 7, 10, 17, 23, 29, 35, 41, 52 in decimal, or…

0xEE, 0xEF, 0xF0, 0xF7, 0xFE, 0xFF, 0x00, 0x01, 0x07, 0x0A, 0x11, 0x17, 0x1D, 0x23, 0x29, 0x34 in hex.

**11/08/16 - 16:30**

Wrote algorithm for conversion of single value from Fahrenheit to Celsius. There are two possibilities:

1)

Subtract 32 from F, then multiply by 5.

Then,

If F < 0, set an ‘increment’ value to -1 and set F = |F|

Else ‘increment’ = 1

Then, count how many times I subtract 9 from F until F < 9

2)

Subtract 32 from F, then multiply by 5.

Then, while F <= -9 or F >= 9,

If F <= 9, F += 9, --count

If F >= 9, F-= 9, ++count

I’m going to see which of these is simpler in assembly.

**11/09/16 - 17:00**

Chose to use the first algorithm for now. I’ll test the other one/optimize later. The first algorithm works for a single value…it seems like way too much code for what it’s doing, I definitely want to rework it when I have a chance.

**11/09/16 - 18:00**

Wrote the algorithm for converting the whole array. It works on the values I created in my array.

**11/09/16 - 20:00**

Added stack to program, even though it’s not really needed.

The program now loads pointers to the two temperature arrays into R1, R2, and an index value (16) into R3. It then calls the conversion procedure, which pushes registers to the stack, does its thing, then pops them off the stack and returns to the mainline program. This is mainly so that the pointers can be incremented as the conversion procedure moves through the array, then have them pointed to the beginning of the array again, in preparation for the averaging.

**11/09/16 - 22:00**

Wrote averaging procedure. The main program sets R0 as a pointer to the array it’s supposed to average, then calls the procedure. It pushes registers to the stack, totals the array, then uses repeated subtraction to find the average. It stores that value in R0, restores the registers and returns. The main program then stores R0 in the appropriate place. I call the averaging procedure twice, once with R0 pointing to the Fahrenheit array, and a second time pointing to the Celsius array. After only minor debugging, the program program works with the initial array of temperatures I supplied. I intend to use a second set to verify that it actually works. I want to make sure that the average works with all negative values, also.

**11/12/16 - 22:00**

Successfully tested program with two more sets of values.

I am realizing now that the program used the repeated-subtraction method of division in two separate places, and I could have made that its own procedure.