EE333 Microcontroller Engineering
Oregon Tech Portland, Fall 2013
Lab Assignment #3 - *Eliminating Display Jitter*
Due November 7

Objectives:

The student will learn how to use low-pass filtering to stabilize a display.

Equipment Required:

Everything from lab assignment 2, including your program.

Background Information:

The program from assignment 2 would have a jittering display, showing two temperature values in what appeared to be simultaneously, whenever the temperature was right on the boundary between two values. In addition, the illumination level LEDs are difficult to read because of the jitter. In an ideal world, this would not occur, however noise in the electronics, and slightly varying temperatures/illumination cause the measurements to be constantly changing.

The way to smooth any changes is to implement a low pass filter. We could do this electronically (with an RC filter, for instance) however we don't really have a way to modify the Dragon12. I don't recommend performing board modifications with an Xacto knife.

One simple technique is to reduce the measuring rate. Measurements are currently taken (by your code) every 1.024 milliseconds. That's even faster than they can be displayed or the display can be seen with the eyes. You could modify the code so that measurements are made at a much slower rate. That way the display could only change once a second and would be easier to read.

Another approach is to average a number of readings to cancel the noise. You could keep track of, say, the last 100 readings, sum them, an divide by 100. But this would be a time consuming task, as well as use up a lot of RAM to hold all of those values. There is a better way that will give an almost identical result which averages all measurement but older ones with ever decreasing significance.

Let's say that the illumination just measured is *L*, and we have another variable that holds the sum of 256 measurements called *S*. This also represents the average of the measurements if we divide it by 256. Each time we take a new measurement we store a new value of S calculated as $\frac{255S}{256} + L$. Each new L becomes 1/256th the value of the new S. Since we are making 976 measurements a second, the value of S will quickly reach the limit where it is 256 times the average luminance. How quickly? Well the "filter" has a time constant of 256 samples, the reading will be

within 1 count in about 1.25 seconds. That should be fast enough. Now the displayed value is calculated from S/256 and the jitter is reduced.

Assignment:

In this assignment you will modify your Lab 2 assignment so that it has a stable display by using averaging and the reduced display rate. I suggest doing this in three steps, verifying operation after each step.

Assignment Step 1:

The first step is to do averaging for the luminance. There is a reason to use the value 256 in the explanation, above. If we have a word variable, the upper byte represents the value divided by 256, so we get a "free" division operation.

I added a word variable named lumavg to hold the sum or averaged value. In the main program loop, where you read from ADR04H, change the code so that instead the most significant byte of lumavg is copied into lumavg.

In the later section (which executes every 1.024 milliseconds) you will read the value in ADR04H and divide by 4, just as in lab 2. Then use it to calculate the new lumavg value. The code to calculate lumavg is a bit obscure. It relies on $\frac{255S}{256} + L$ being the same as $L + S - \frac{S}{256}$. The code to do this is compact but a bit obscure, so I'll show it here:

```
  ; start out with the luminance value in D
addd    lumavg    ; add lumavg
subb    lumavg    ; subtract lumavg/256
sbca    #0        ; handle borrow from upper byte
std     lumavg    ; store result.
```

Verify operation at this point before proceeding. Note how the displayed value is more stable, but still is a bit difficult to read.

Assignment Step 2:

Repeat what you did in Step 1, but for the temperature measurement. The sum should be 256 times the temperature in degrees Celsius.

Assignment Step 3:

Modify the program from Step 2 so that the displays update roughly four times a second instead of what appears to be continuously. This should really improve the ability to read the luminance and less obviously the temperature.

To do this, make use of the section of code that executes once every 1.024 milliseconds. Add a byte variable that is incremented here. Whenever the value becomes zero (which will happen every 256 interrupts, roughly four times a second) set a second byte variable (a "flag" variable) to be non-zero. This means that the flag variable will be set roughly four times a second.

Now in the main program loop (where you write to the display variables) add a test to see if the flag variable is non-zero. If it is non-zero, clear the variable (make it zero) and then write to the display. If it is zero, branch around the code that writes to the display. This way the display will be updated only four times a second.

Run the program and verify operation. Much better, isn't it?

To turn in:

- Documented program listing.
- Description of how you tested the program.
- Discussion of any problems you had

This should all be placed in a single file (PDF format preferred, Word or Open Office formats also acceptable).