ENSC 254

Summer 2018

Lab 5

Towards the stars

Deadline:June 28th, 2018
Last revision: June 23, 2018

Karol Swietlicki
Simon Fraser University

1 Towards a synthesizer

We are continuing to make progress towards creating a large program in assembly language. This lab is mostly intended as practice of working with memory and large programs.

2 Caveats

You are likely to need more than 16 data items to perform the tasks within this lab. Use memory locations to store data. At the start of the program the register R4 contains the address of one megabyte of memory. Divide it up and do with it whatever you wish.

Keep in mind that memory accesses are very, very slow compared to operations which only work on registers. If you use memory reads in your delay loops you may lose accuracy if you don't recalibrate your timing.

This lab is divided into several sections. Only the sixth section is going to be marked. The other sections exist to help guide you towards the completion of the lab. Sections seven and eight will be a part of a future lab. You can submit those in lieu of section six, if that is more convenient for you.

3 Prelude

The main point of this lab is for you to both work with a large program, as well to practice doing some research on your own.

I often use the word "voice" in relation to LEDs. This is there for a reason, as the lab will be used as a base of a sound synthesizer. For now you will only work with light.

The LED numbers are printed on the circuit board. 7 is the leftmost LED and 0 is the rightmost one.

4 Voice allocation protocol

Whenever the center button on the directional/cross set of buttons is pressed, light up the next LED in order, from left to right. For example, if the previously lit LED was 5, light up the LED with the number 4. Wrap around once you exhaust the possibility of motion towards the right. When the center button is released, turn off the LED that it lit, making sure that no LED is lit.

You don't need to handle multiple buttons, but you need to handle button bounce. One button press must equal to one space worth of motion, not more. You can look up how to debounce buttons in software online.

5 Restricted voice allocation protocol

Repeat the previous section, but this time if a slider next to an LED is in the down position the LED needs to be skipped over.

For example, if the last lit LED was 4 and the only slider currently up is 6, then 6 is the LED that must light up, as all other LEDs are supposed to be skipped. Note that the slider switches should have no impact on already lit LEDs.

6 Final voice allocation protocol

Repeat the previous section, this time using all five of the cross/directional buttons. You can ignore the sliders for this one.

When a button is pressed, light whichever LED is the next one in order to be lit. Keep track of which LED is lit up by which button. When a button is released, turn off the LED that this button lit. You need to remember which button triggered which LED to facilitate this.

If an LED that is next in turn to be lit is already taken, skip over it and light the next free one, just as you did with the sliders in the down position that made LEDs to be skipped in the previous section.

An alternative description for the same behavior is as follows: There is one, global, insertion point. When a button is pressed, check if that insertion point contains an already lit LED. If so, move your insertion point one unit to the right and check for vacancy again. Otherwise, if the LED at the insertion point is not lit, light it and move the insertion point one space to the right. Wrap the insertion point all the way to the left should it scroll off the right side of our LED strip. The insertion point does not reposition on button releases.

Submit your code to CourSys.

7 LED brightness

Repeat the previous section. You will still light up LEDs when a button is pressed, but this time you won't light up the LEDs to the maximum possible brightness. Make the apparent light intensity be $(X+1)/8$, where X is the number of the LED in question. Different LED, different brightness.

Make sure that you are using values which correspond to human perception, not to the raw duty cycles. The LEDs must appear, to the human eye, about even in their differences of light intensity.

8 Per aspera ad astra

Repeat the previous section.

Instead of a fixed brightness, make each lit LED shimmer, periodically dim and twinkle like a star. Each LED should be completely independent of the others. One LED twinkling or dimming should not impact how the others act. The effect must be as unpredictable as possible. Make sure no discernible repeating pattern in your effect is apparent. Your effect must not rely in entirety on a very slow update period to cause constant flickering. This can be a part of the effect, but it cannot be the whole thing.

You can accomplish this in many ways. I would suggest using a very large table of constants generated through any means of your choosing and hardcoded into your assembly file, with linear interpolation between the adjacent values.

9 All in one

In case the last section isn't clear, here is exactly what is expected from you. This section is just a rewording of the previous one.

Use the "final voice allocation protocol" to turn on LEDs when directional buttons are pressed and to turn them off when they are released. LEDs must be allocated in a round-robin fashion, with the already lit LEDs being skipped over. Each lit LED must gently shimmer, dim or twinkle independently of the others. You can accomplish that with a duty cycle that varies through time. A casual observer can't look at the LEDs that are lit and observe a repeating pattern in the change in the brightness of the LEDs. That's it. You can ignore the sliders.

10 Bonus possibilities

A bonus point will be awarded to any group which implements either Perlin, Simplex, OpenSimplex, Value or Octave noise to drive the LED shimmer. This is a very difficult task.