COE 0147 Lab 4

Lab 4: LED Simulator

## Important:

* **For this lab, we require a modified version of Mars, which is available on courseweb**.
* **All of your code will be entered in the template, also available on courseweb.**
* **Please make sure to change the part value in the data field depending on what part you are working on.** **e.g if you are working on part 2, the value should be 2.**

**A note about how the LED simulator works:** You must enable the LED simulator by selecting “Keypad and LED Simulator” from the **tools** menu. Once it has been enabled, click “connect to MIPS”. You can draw to (read from) the LED display by writing to (reading from) its memory, which begins at address 0xFFFF0008.

The first byte (8 bits) of LED memory corresponds to the first 4 dots in the display (2 bits per dot). The 7th and 6th bits of the byte control the left most dot (of the first row), the 5th and 4th bits control the second dot from the left, and so on. Since there are two bits per dot, each dot can be set in only 4 colors (including the ‘off’ color).

# drawPattern()

Write a function *drawPattern (int\* address, int bitPattern)* that stores a word-sized *bitPattern* in the memory location pointed to by the *address*. You may assume that the *address* is word-aligned. In the function definition, *int* is the size of a word and *int\** is a pointer to a word, i.e., address of a word. Use the following code to test your function.

If your *drawPattern()* function is correct, the test code in the template and your function should light up the first 16 LEDs (starting from the top left) of the LED display. The output pattern for this part is shown in Fig. 1. (Here you could guess and try to know the LED color-value mappings.)

## Note: Please make sure this part works! The later sections require this to function correctly.



Figure 1: Output Pattern for part 1

# getPattern()

Write a function *int getPattern (int\* address)* that returns the word-sized bit pattern currently stored in the memory location pointed to by the *address*. There is test code in the template.

## Note: Please make sure this part works! The next two sections require this to function correctly. To test, change the part value in the .data section to 2.

## If you code part 2 correctly, you will see the string "The LED pattern matches.”

## If you see “The LED pattern doesn’t match!” you have an error.

1. **disruptPattern()**

Write a function *void disruptPattern (int\* address)* that reads the word-sized bit pattern stored in the memory location pointed to by the *address*, XORs that word with the value 0x???????? to **get the pattern as shown in Fig.2.** (You should be able to figure out this value by your guess and trial)

You should **start with the existing pattern in part 1 (this is already done if you follow the template),** get the pattern, **derive the XOR value you need** and store the XOR result back at the same location in the LED memory. You may assume that this address is word-aligned.

The above steps will alter the LED lights at that address. Your function must use the functions defined in the previous two parts, i.e., your *disruptPattern()* function must make use of your *getPattern()* and *drawPattern()* functions. The output for this part is shown in Fig. 2.

**To test, make sure you change the part value in the data section to 3! You should see the pattern change after 1 second.**

**After disruptPattern runs, it should look like figure 2.**



Figure 2: Output Pattern for part 3

# drawShape()

Write a function *drawShape* to draw the pattern shown in Fig. 3 on the LED display. Your *drawShape()* function must use your *drawPattern()* function in a loop to generate the desired pattern. For each iteration of the loop, turn on (light up) a section of LED memory; the section that you turn on will be immediately below the previous section. Note that each row consists of 128 LEDs, and each LED requires 2 bits, therefore each row is 256 bits wide (8 words). You will need to figure out the values to pass to the drawPattern function. This can be determined via trial and error.

**To test, make sure you change the part value in the data section to 4!**



Figure 3: Output Pattern for part 4

**For part 4, you need to enter code in two places:**

**You need to code the drawShape function near the bottom of the file.**

**You also need to code the test to call the function. Code in under the location shown below:**

# Add test code here to test part 4

part4:

Once parts 3 and 4 work, you are done. Submit the file to courseweb.

Name it: **lab4\_username.asm**