COE 0147 Spring 2013
Lab 4: Simple Functions

Each of you should submit your own solution via email, according to the instructions on your TA's website. You may collaborate with your partner, but each person must turn in their own copy of the lab, with the name of their partner.

In this lab, we will write four functions that manipulate the memory of an LED display, in order to turn on and off some LEDs. This requires the use of a specially modified version of MARS, which is available at: http://www.pitt.edu/~dbd12/teaching/files/Mars-4.1-With-Keypad-LED128x8.jar

The first two parts are **not** directly submitted but are instead used in parts three and four.

You do not have to save any registers on the stack. You may need to make a backup of a register, but may do so by moving it to an unused register. You do not need to perform error checking (e.g., you may assume that the addresses passed to a function are valid addresses).

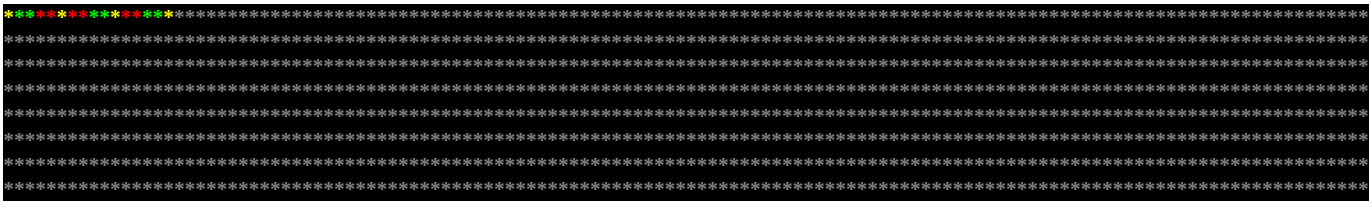
A note about how the LED simulator works: You have to enable the LED simulator by choosing "Keypad and LED Simulator" from the tools menu. Once it has been enabled (click, "Connect to MIPS"), you may draw to the display by writing to its memory. You may read from the display by reading from its memory. Its memory begins at address 0xFFFF0008.

The first byte (8 bits) of LED memory corresponds to the first 4 dots in the display. There are 2 bits per display dot. The 7th and 6th bits of the byte are the far left dot, the 5th and 4th bits of the byte are the second dot on the left, and so on. Since there are two bits in a dot, there are 4 color combinations total (this includes the 'off' color).

Part 1) Write a function *void drawPattern(int *address, **int bitPattern**)* that stores the **word-sized** bitPattern in the memory location pointed to by address. You may assume that address is word-aligned. In the previous definition, an int is the size of a word and int * is a pointer to a word (address of a word). You may consider using the code below to test your function:

```
.text
li $a0, 0xFFFF0008  #LED memory starts at this address
li $a1, 0x7EF965BD  #LEDs to turn on
jal drawPattern     #Jump and link to setLED
li $v0, 10          #Exit
syscall
```

If you have written your drawPattern function properly, the above test code should light up the first 16 LEDs (starting in the top left) of the LED display. The pattern that should be shown should look like:



Challenge: drawPattern can be done in two instructions.

Part 2) Write a function *int getPattern(int *address)* that returns the **word-sized** bit pattern currently stored in the memory location pointed to by address. You may assume that address is word-aligned. You may consider using the code below to test your function:

```
          .data
ok:       .asciiz   "The LED pattern matches."
not_ok:   .asciiz   "The LED pattern doesn't match!"

          .text
          li $a0, 0xFFFF0008  #LED memory starts at this address
          li $a1, 0x7EF965BD  #LEDs to turn on
          jal drawPattern     #Jump and link to drawPattern
          jal getPattern      #Jump and link to getPattern
          bne $a1, $v0, else  #Return values should be in $v0
          la $a0, ok          #Load ok string if equal
          j end
else:     la $a0, not_ok      #Load not_ok string if not equal
end:      li $v0, 4           #Print the string
          syscall
          li $v0, 10          #Exit
          syscall
```

If you're using the test code in part 2, it should print out that the LED pattern matches. Why? The first step is to draw a pattern to LED memory at a specific location. Then, the test code retrieves the pattern at that exact same location. The retrieved pattern should match what had just been drawn.

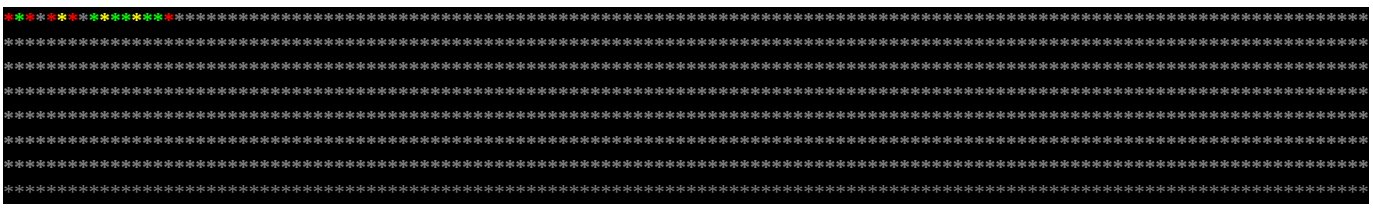
Challenge: getPattern can be done in two instructions.

Part 3) Write a function *void disruptPattern(int *address)* that reads the **word-sized** bit pattern stored in the memory location pointed to by address, XORs that word with the value 0xC31601C9, and stores it back to the same location in 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. That is, your disruptPattern function **must** make use of your getPattern and drawPattern functions. For this part of the lab, be sure to use the template at:
http://www.pitt.edu/~dbd12/teaching/files/lab4part1.asm

Question 1: Submit part 3 (be sure that you used and include the functions you created for parts 1 and 2). Be sure that your file is named **lab4part1.asm**.

Challenge: disruptPattern can be done in seven instructions (using pseudoinstructions).



Part 4) Write the function *void drawRepeatedPattern(int *address, **int bitPattern**, int num).* drawRepeatedPattern should store the **word** bitPattern num times vertically. For example, if bitPattern was a pattern that was all red (16 red LEDs) and num was 2, then the result would be an LED display with a red rectangle which is 2 LEDs high and 16 LEDs wide.

You may assume that address is word-aligned. Your drawRepeatedPattern function **must use** your drawPattern function. Your function should not change the LED memory if num is 0 (i.e., the user is telling the function to draw a pattern 0 times). For this part of the lab, be sure to use the template at:
http://www.pitt.edu/~dbd12/teaching/files/lab4part2.asm

Your drawRepeatedPattern function will require the use of a loop. Each time through the loop, you will want to draw a pattern to a section of LED memory. The section that you draw to will be a different section than before, and will be the section immediately below the previous section. Recall that each LED row consists of 128 LEDs. Each LED is 2 bits, which is 256 bits. 256 bits is 32 bytes (8 words).

Question 2: Submit program 4 (be sure that you used and included your drawPattern function that you created for program 1). Be sure that your file is named **lab4part2.asm**.

After running your program, the LED display should show: