CS/CoE 0447 Spring 2017
Lab 5: Recursive Functions and LED Display

Released: Thursday 9 February 2017, 8am EST
Due: Monday 20 February 2017, Midnight EST

Each of you should submit your own solution to CourseWeb. If you choose to work with a neighbor/partner, put your partner's name on your submitted copy of the lab. Submission timestamps will be checked. Late submissions will not be accepted.

For each of the **two parts** in this lab, you must name your files in a specific format. Each part will tell you the name to use. Follow instructions so that your program produces output in the correct format.

Part 1: Recursive Functions

Consider a function defined for two non-negative integers n and k as follows[1]:

$$C(n,k) = \begin{cases} C(n-1,k-1) + C(n-1,k) & \text{if } 0 < k < n \\ 0 & \text{if } n < k \\ 1 & \text{if } n = k \\ 1 & \text{if } k = 0 \end{cases}$$

This function can be expressed recursively, as shown in the following pseudo-code:

```
int choose(int n, int k) {
      if (k == 0) {
            return 1;
      } else if (n == k){
            return 1;
      } else if (n < k){
            return 0;
      } else {
            return C(n-1, k-1) + C(n-1, k);
      }
}
```

Write MIPS assembly code that implements the '***choose***' function. Base your solution on the pseudo-code above. **Your code must call itself recursively (TAs will explicitly check for 'jal' instructions inside the function).** Place the arguments in the proper argument registers and the result into the proper return value register. Your function must save/restore any $s registers that it uses as well as the $ra register. To do this, use a prologue and epilogue.

Now, write a MIPS program that prompts the user for two numbers, calls the '***choose***' function with

[1] In case you are curious, the function computes the binomial coefficient $\binom{n}{k}$.

the two numbers as arguments and prints the value returned by the function.

Here is sample output from the program:

```
Please enter a number n:
5 <--- this is the input
Please enter another number k:
2 <--- this is the input
The chosen value is 10
```

Please make sure your output line contains the string **"`The chosen value is `"** as shown in the sample output.

Note: Anything after a # is ignored by MARS.

Question 1: Submit your program "**lab05part1-<your Pitt Mail ID>.asm"**. For example, "**lab05part1-xyz12.asm" (**Instead of your PeopleSoft number, be sure to use your Pitt Mail ID.) Output format will be strictly checked.

Part 2: LED Display

In this part of lab, we will use the LED display that you will use for your class project. Your objective is to draw a square on the display by manipulating the memory of the LED display, to turn on and off some LEDs.

Download from CourseWeb/Lab5, Mars4_5-Pitt.jar. This version of Mars contains the LED keypad support. You may download and use the setLED and drawVerticalLine files as well.

Write a MIPS program that prompts the user for 3 numbers (x, y, and size) and a character representing the color ('g' for green, 'y' for yellow and 'r' for red) and draws a line using the '***drawVerticalLine***' function. The function '***drawVerticalLine(int x, int y, int size, int color)***' draws a vertical line between the LED at position (x, y) and the LED at position (x, y + size - 1).

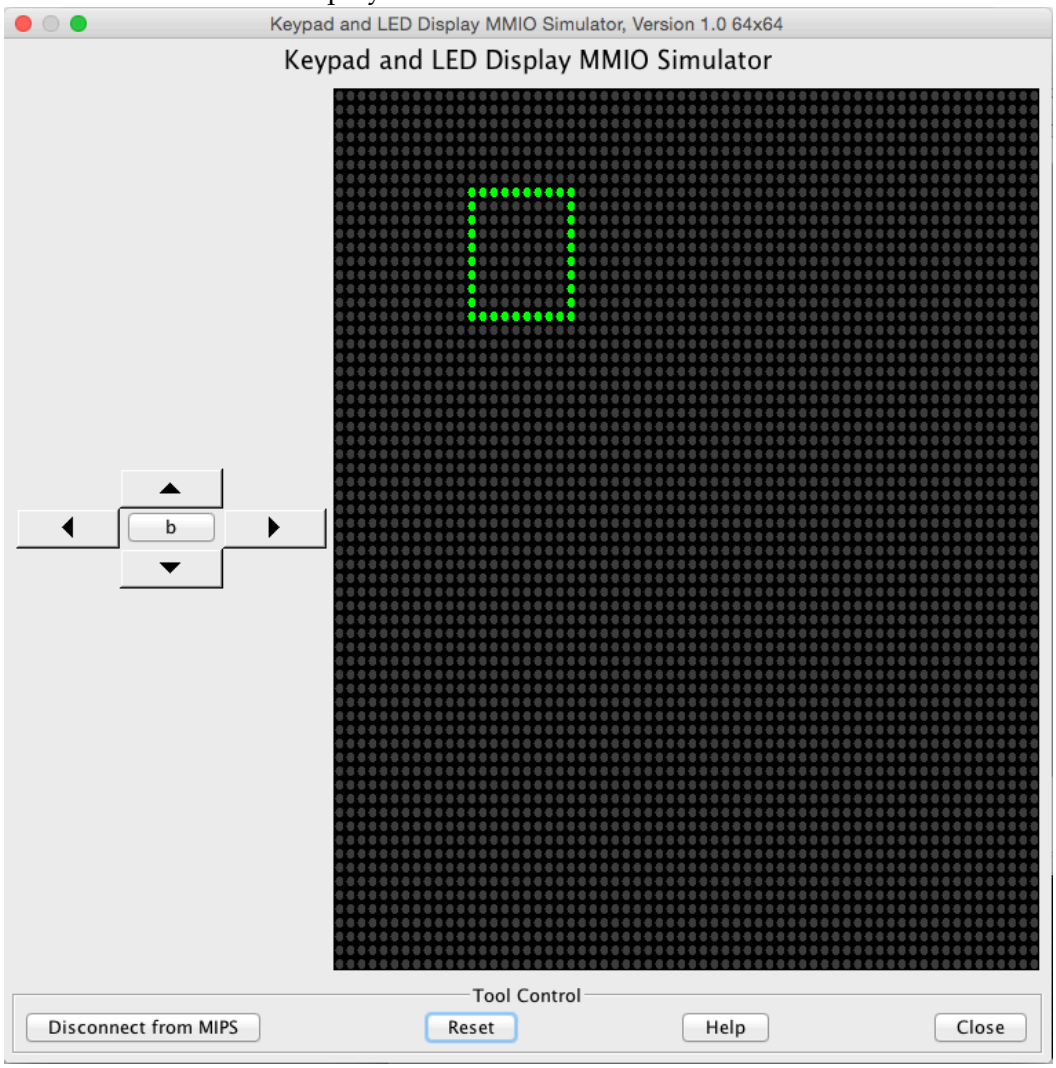
Write a function '***drawHorizontalLine(int x, int y, int size, int color)***' that draws a horizontal line between the LED at position (x, y) and the LED at position (x + size - 1, y). Use function `***setLED***` to turn on each LED that is part of the line. Your function must save the $ra register and any $s0 register that it uses. Test by modifying your main program to call '***drawHorizontalLine'.***

Now, write a function ***'drawSquare(int x, int y, int size, int color)'*** that draws a square starting at position (x, y) and ending at position (x + size - 1, y + size - 1). Use functions '***drawHorizontalLine***' and '***drawVerticalLine'*** to draw the sides of the square. Modify your main program to call '***drawSquare***'.

Here is sample output from the program:

```
Please enter the x coordinate of the position:
12 <--- this is the input
Please enter the y coordinate of the position:
7 <--- this is the input
Please enter the size:
10 <--- this is the input
Please enter the color ('g' -green, 'y' -yellow and 'r' -red):
g <--- this is the input
```

Here is how the LED Display looks like:



Question 2: Submit your program "**lab05part2-<your Pitt Mail ID>.asm"**. For example, "**lab05part2-xyz12.asm" (**Instead of your PeopleSoft number, be sure to use your Pitt Mail ID.) Output format will be strictly checked.

A note about how the LED simulator works: You have to enable the LED display simulator by choosing "Keypad and LED Display 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).

You may use the function '***setLED(int x, int y, int color)***', which sets the LED at position (x,y) to the given color. Possible values for color are 0 (off), 1 (red), 2 (yellow) and 3 (green). The following MIPS code shows an implementation of '***setLED***':

```
setLED:
      #arguments: $a0 is x, $a1 is y, $a2 is color
      # byte offset into display = y * 16 bytes + (x / 4)
      sll   $t0,$a1,4      # y * 16 bytes
      srl   $t1,$a0,2      # x / 4
      add   $t0,$t0,$t1    # byte offset into display
      li    $t2,0xffff0008   # base address of LED display
      add   $t0,$t2,$t0    # address of byte with the LED
      # now, compute led position in the byte and the mask for it
      andi  $t1,$a0,0x3    # remainder is led position in byte
      neg   $t1,$t1        # negate position for subtraction
      addi  $t1,$t1,3      # bit positions in reverse order
      sll   $t1,$t1,1      # led is 2 bits
      # compute two masks: one to clear field, one to set new color
      li    $t2,3
      sllv  $t2,$t2,$t1
      not   $t2,$t2        # bit mask for clearing current color
      sllv  $t1,$a2,$t1    # bit mask for setting color
      # get current LED value, set the new field, store it back to LED
      lbu   $t3,0($t0)     # read current LED value
      and   $t3,$t3,$t2    # clear the field for the color
      or    $t3,$t3,$t1    # set color field
      sb    $t3,0($t0)     # update display
      jr    $ra
```