CS2630: Computer Organization
Homework 3
MIPS procedure calls, I/O, data structures
Due July 10, 2017, 11:59pm

Instructions: You must submit your files to Assignments > Homework 3 on ICON. Make sure your programs run successfully and you double check your uploaded files.

The files to submit are:
- draw.asm
- listnode.asm
- (optional) extra.asm

All procedures you write in this assignment should work if called on other inputs besides the ones given in the starter code.

Goals
- Write MIPS programs that define and call procedures using standard calling convention
- Write recursive procedures in MIPS
- Add methods to a data structure class written in MIPS
- Write programs with two kinds of I/O: memory mapping and syscalls

Part 1: Drawing in a Bitmap Display

In this problem, you will write a series of procedures in MIPS to draw to a bitmap display.

a) Part 1a only involves setup and explanation. You can run the starter code with the following steps.
   i. Open draw.asm in MARS
   ii. Assemble the file
   iii. Open the Bitmap Display under Tools
iv. This tool is an emulation of a computer screen. Leave all the dropdown options in their default values.

v. Click "connect to MIPS". This button will attach the screen to your MIPS program.

vi. Now you can run the drawing program. Press run as usual.

vii. All draw.asm does is put 3 tiny red pixels in the center of the screen. Your Bitmap Display should now look like this.

If it is hard to see, you can try using the screen magnifier under Tools.

How does the display work? It uses a method of I/O (input/output) called memory mapped I/O (or, MMIO). The top left pixel (0,0) is mapped to the base address
0x10010000. Each pixel is 4 bytes and the color is given by a 24-bit "RGB value" stored to the least significant 3 bytes.

Let's look at an example of a width=512, height=256 screen zoomed in to look at just the 9 pixels in the top left corner. If we want to draw

```
  ●  ●  ●
  ●  ●  ●
  ●  ●  ●
  ●  ●  ●
```

Then we would store the following pixel **VALUES**. The memory is initialized to 0x00000000 (black) for every pixel, so we only have to store the non-zeros.

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0x00FF0000</td>
<td>0x00000000</td>
<td>0x00000000</td>
</tr>
<tr>
<td>1</td>
<td>0x00000000</td>
<td>0x00FF0000</td>
<td>0x00000000</td>
</tr>
<tr>
<td>2</td>
<td>0x00000000</td>
<td>0x00000000</td>
<td>0x000000FF</td>
</tr>
</tbody>
</table>

The addresses of the pixels increase linearly from left to right and top to bottom. Here's the **ADDRESSES** of those 9 pixels in the width=512, height=256 screen.

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0x10010000</td>
<td>0x10010004</td>
<td>0x10010008</td>
</tr>
<tr>
<td>1</td>
<td>0x10010000</td>
<td>0x10010004</td>
<td>0x10010008</td>
</tr>
<tr>
<td>2</td>
<td>0x10010000</td>
<td>0x10010004</td>
<td>0x10010004</td>
</tr>
</tbody>
</table>

b) Write the drawhorizontal procedure. You must fill in the `drawhorizontal` procedure in `draw.asm`. This procedure takes 4 arguments: a starting x, a starting y, length, and RGB color and draws a line on the Bitmap Display by storing values to memory.

Requirements:
- `drawhorizontal` must use the standard MIPS procedure conventions that were shown in class
- `drawhorizontal` must be implemented using recursion
- You may assume you are not given arguments that would draw off-screen

Hints:
- For an example of how to write to the screen, see the provided procedure `draw3pixels`.
- We've provided an `addressOfPixel` procedure that may be useful to you. See the code `draw3pixels` for an example of using `addressOfPixel`.
- Feel free to write helper procedures.
• If you aren't seeing anything on the screen, double check the steps in part 1a. Alternatively to using the screen, you can always check the Data Segment to see if the values have been written.

When you finish this part, you should be able to run the program and see a green line in the top left corner.

c) Write the `drawvertical` procedure. It is similar to the previous procedure, except it draws a vertical line.

Requirements:
• `drawvertical` must use the standard MIPS procedure conventions that were shown in class
• `drawvertical` must be implemented using recursion
• You may assume you are not given arguments that would draw off-screen

When you are done, you should see something like this
d) Write the `drawsquare` procedure. This procedure takes 4 arguments: a starting x, a starting y, side length, and RGB color and draws a square on the Bitmap Display by storing values to memory.

\[(x,y)\] 

\[(x+\text{length},y+\text{length})\]

Requirements:
- `drawsquare` must use the standard MIPS procedure conventions that were shown in class
- `drawsquare` must call `drawvertical` and `drawhorizontal` to draw the sides.
- You may assume you are not given arguments that would draw off-screen

e) Fill in the code that calls the `drawsquare` procedure to draw a square of side length 100 and top left corner (80,80).

When you are done with this step, you should see the complete drawing.
Part 2: Adding a method to a linked list ListNode class

The starter file listnode.asm contains a Java class in the comments and the same program written in MIPS. This ListNode class is the same one discussed in the lecture on data structures. Your job is to fill in the missing printlist method.

a) Assemble and run the code. Examine the code and the register values to find where the linked list is stored. Remember, it is a linked list not an array. The test program copies an array in the .data section starting at 0x10010000 to a linked list by calling the append method in a loop.

b) Write the printlist method. It takes one argument: the address of the first ListNode. It prints the list with values separated by commas.

For example, if $a0 stores the address of the following list

```
   | 10 | ←--| 20 | ←--| / |
```

Then calling printlist will print to the console:

```
10,20,
```
Note that the trailing comma (,) is expected.

Requirements:
- `printlist` must use the standard MIPS procedure conventions that were shown in class
- `printlist` must be implemented using recursion
- If it helps, you may assume that `printlist` is never called with the argument NULL (i.e., empty list).

Hints:
- Just like we've provided the Java code for the constructor and append method, it may help you to write the Java version of `printlist` then translate the algorithm to MIPS.
- To print an integer to the console, use the "print integer" syscall. You can find information in MARS under Help > MIPS > Syscalls.
- To print an ASCII character to the console use the "print character" syscall. Note that the MARS provides this convenience: write a character in single quotes (e.g., ', ') instead of an integer and the assembler will convert it to ASCII encoding for you.

Extra credit

Draw something much more interesting than a square, using recursion. Submit your code in `extra.asm`. Also, post a screenshot of your artwork to https://uiowa.instructure.com/courses/56016/discussion_topics/337435.