CSCI 260 – Project 1

Due: Oct 15 11:59 PM

The goal of this homework is to write a non-trivial MIPS program, and execute it using a MIPS simulator (MARS). It is an **individual** assignment and no collaboration on coding is allowed (other than general discussions).

1 Assignment

For this assignment, you will draw a shape on a 2D bitmap display.

Inputs

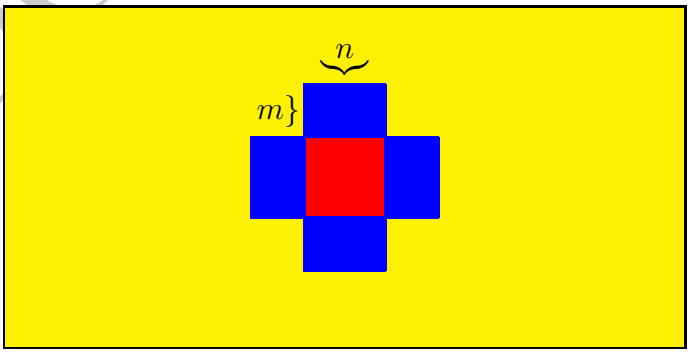
The input will be the assembly language version of the following C structure (details in Section 3):

```
struct projInput {
  unsigned m, n;          // size parameters
  unsigned c1r, c1g, c1b; // red-green-blue for color 1
  unsigned c2r, c2g, c2b; // ... color 2
  unsigned c3r, c3g, c3b; // ... color 3
}
```

You may assume that your implementation of C uses 32-bit ints (so there are 11 words in the above definition). You may also assume that the upper 24 bits of each color component are 0s.

Task

Write MIPS code to draw the following shape on the bitmap display supplied as part of MARS:



The above figure is shown for c1/c2/c3 being yellow/blue/red respectively, but your code should of course use whichever colors are input. The figure (in c2/c3) should be centered in the frame (in c1). The variables

m and n refer to the length of the corresponding dimensions and should **not** be drawn. The figure is horizontally and vertically symmetric, and the center box (in c3) is a square. Additional notes:

- If n is odd, you should increment it to the next even number, so that a perfect centering is possible.
- If the parameters are not realizable (*e.g.,*the combination of m and n is too big), your code should display nothing.

2 Submission Instructions

Please submit a single text file using blackboard (under the Projects menu). The file should contain your name in header comments, and follow normal MIPS conventions. Your program will be tested on multiple values of the inputs, so you should test your program adequately. You may use any of the instructions and pseudo-instructions we covered in class excluding multiplication and division (you're probably thinking the wrong way if you think you need these).

Your program will be graded on both correctness (on all test cases) and style (meaningful comments on every line, register conventions, etc.).

See syllabus for late policy.

3 Writing The Program

Accessing the Bitmap Display

In your program, your data segment should start with the following (replace `--` with appropriate values):

```
.data
frameBuffer:  .space  0x80000  # 512 wide X 256 high pixels
m:            .word   --
n:            .word   --
c1r:          .word   --
c1g:          .word   --
c1b:          .word   --
c2r:          .word   --
c2g:          .word   --
c2b:          .word   --
c3r:          .word   --
c3g:          .word   --
c3b:          .word   --
```

If you need additional variables, please place them *after* the above.

The framebuffer is essentially memory-mapped I/O storing a 512-pixel×256-pixel×32-bit image in row-major order. For example, MEM[frameBuffer+4] would contain the pixel at row 0, column 1.

Each **pixel** is a 32-bit value consisting of 8-bits each for the red, green, and blue components in bits 23:16, 15:8, and 7:0 respectively (the upper 8 bits are ignored). For example, the value 0x0000FF00 would correspond to bright green. Since our color components are words, you may assume that their upper 24 bits are zeros.

Using the MARS Bitmap Display

Please see the other guide posted on bb for how to use MARS. After reading that, you will need to do a few additional things to use the bitmap display as follows:

1. Select `Tools→Bitmap Display`
2. Click the `Connect to MIPS` button
3. Follow the instructions on the MARS guide to assemble and run your program.

The data segment defaults to starting at 0x10010000.

Sample code: The following snippet (at the beginning of the text segment) draws a 10-pixel green line segment somewhere near the top center of the display (assuming you have the data segment from above):

```
.text
drawLine:  la       $t1,frameBuffer
           li       $t3,0x0000FF00   # $t3 ← green
           sw       $t3,56300($t1)
           sw       $t3,56304($t1)
           sw       $t3,56308($t1)
           sw       $t3,56312($t1)
           sw       $t3,56316($t1)
           sw       $t3,56320($t1)
           sw       $t3,56324($t1)
           sw       $t3,56328($t1)
           sw       $t3,56332($t1)
           sw       $t3,56336($t1)
           li       $v0,10           # exit code
           syscall
```

Two pseudo-instructions we haven't covered yet are:

- `la reg,Label`: load the address corresponding to `Label` into register `reg`
- `li reg,imm32`: load the 32-bit immediate `imm32` into register `reg`