Project 1: Gradient Diamond Drawing in MIPS

Due: July 7th

The goal of this homework is to write a non-trivial MIPS program, and execute it using a MIPS simulator (MARS). This is an **individual** assignment, and collaboration on code is not permitted beyond general discussions.

1. Assignment

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

Inputs

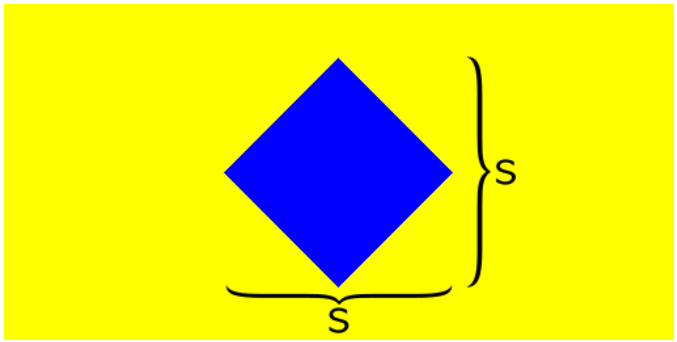
The input will be the assembly language version of the following C structure (details in Section 3):

```
struct projInput {
    unsigned s;          // Diagonal length of the diamond (must be odd)
    unsigned x, y;       // (X, Y) coordinates of the diamond center
    unsigned cr, cg, cb; // RGB components for the diamond color
}
```

You may assume that your implementation of C uses 32-bit ints (so there are 6 words in the above definition). You may also assume that the upper 24 bits of each color component above are 0s.

Task

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



More information about the picture:

- The background color is yellow at full brightness, i.e., `0x00FFFF00`.
- The diamond is centered **exactly** at coordinates (x, y), with a horizontal and vertical size of **s** pixels (where **s must** be odd).
- The initial color of the diamond is specified by the RGB components `cr-cg-cb`.
- The diamond is symmetrical, with diagonal lines that increase and/or decrease by one pixel in both the x and y directions (slope of 1).
- If the input parameters are invalid or the diamond cannot be centered exactly, only the yellow background should be drawn.
- Implement a color gradient, such that:

- **Left of x-coordinate**: Red component decreases by 1 per pixel left of the given x-coordinate.
- **Right of x-coordinate**: Green component decreases by 1 per pixel right of the given x-coordinate.
- Color components do not decrease past 0.

The above figure is shown for certain colors and sizes, but your code should be able to run for arbitrarily given values in projInput. Your program should first check the values given in projInput.

If it is not possible to draw the figure (e.g., the diamond exceeds the 512×256 bitmap dimensions or has an even diagonal length), your program should only draw the yellow background.

You will need to determine other error cases too. If your display pixels are not squares, your picture may be a bit elongated (so the diagonals are not at a 45-degree angle), but that is acceptable and expected.

2. Submission Instructions

Submit as one file named `CSCI260project.asm` to gradescope. Make sure to follow the formatting rules stated in Section 3.

The file should follow normal MIPS conventions including comments. Your program will be tested on multiple different values, so you should test your program adequately. You may use any of the instructions and pseudo-instructions we covered in class excluding multiplication/division (you're probably thinking the wrong way if you think you need these). Note that MARS may treat what appear as real instructions as pseudoinstructions (for example something with a 32-bit immediate) and this is allowed; however, you may regret using these when debugging.

Your program will be graded on both correctness (on all test cases) and style (meaningful comments on every line, register conventions, etc.). Although you don't need to have the most efficient implementation, it should be reasonable. See syllabus for late policy.

3. Writing The Program

Colors

Recall that a standard RGB color is one word with the following format:

0000000	red	green	blue

where each field is one byte.

Accessing the Bitmap Display

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

```
.data
# DONOTMODIFYTHISLINE
frameBuffer:    .space  0x80000  # 512 wide X 256 high pixels
s:              .word   --       # Size of diamond
x:              .word   --       # X-coordinate
y:              .word   --       # Y-coordinate
cr:             .word   --       # Red component
cg:             .word   --       # Green component
cb:             .word   --       # Blue component
# DONOTMODIFYTHISLINE
# Your other variables go BELOW here only
```

You **must** have the exact names/format given above as your program will not pass our tests otherwise. It also needs to include the two DONOTMODIFY lines exactly as given (1 space after #, no other spaces), and that section should contain only those variables.

The `framebuffer` is essentially memory-mapped I/O storing a 512-pixel × 256-pixel × 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 (Course Materials) 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` → `BitmapDisplay`
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 $t3,15340($t1)
    sw $t3,15344($t1)
    sw $t3,15348($t1)
    sw $t3,15352($t1)
    sw $t3,15356($t1)
    sw $t3,15360($t1)
    sw $t3,15364($t1)
    sw $t3,15368($t1)
    sw $t3,15372($t1)
    sw $t3,15376($t1)
    li $v0,10                          # exit code
    syscall                            # exit to OS
```

Two important pseudoinstructions, which are necessary for this project:

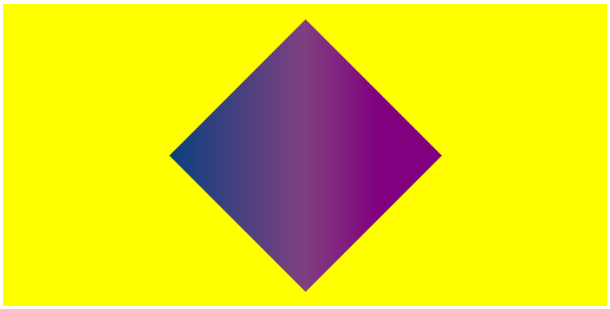
- la reg,Label: load the address corresponding to Label into register reg
- li reg,imm32: load the 32-bit immediate imm32 into register reg

Additional Resources

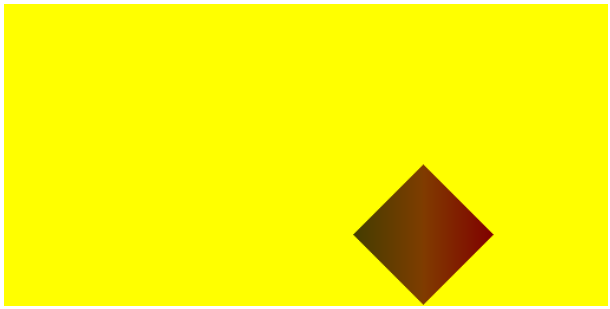
Refer to the MARS documentation and the bitmap display guide on Blackboard for more information on using the simulator and accessing the bitmap display. Posted on blackboard is the full example to draw a line. Below are several examples of different inputs, and the output for each.

Example Outputs

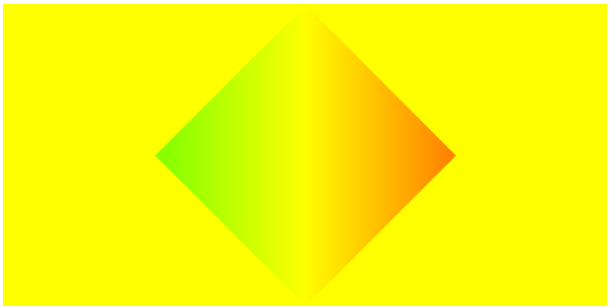

```
s:  .word 91
x:  .word 100
y:  .word 125
cr: .word 255
cg: .word 255
cb: .word 150
```



```
s:  .word 231
x:  .word 256
y:  .word 128
cr: .word 128
cg: .word 64
cb: .word 128
```



```
s:  .word 119
x:  .word 355
y:  .word 195
cr: .word 128
cg: .word 61
cb: .word 0
```



```
s:  .word 255
x:  .word 256
y:  .word 128
cr: .word 255
cg: .word 255
cb: .word 0
```