CSCI 260 – Project 1

Due: April 01 11^{59} PM (Extended)

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

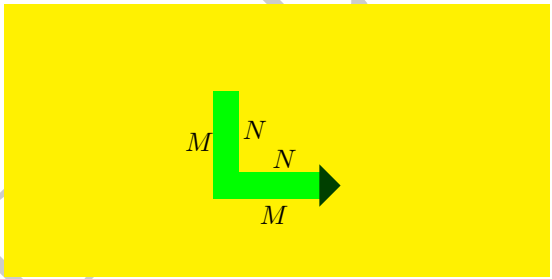
The input will be the assembly language version of the following C structure (details in Section 3):

```
struct projInput {
  unsigned M, N;          // size parameters
  unsigned cr, cg, cb;    // red-green-blue components for arrowhead 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 (**read carefully**):

- The background is always yellow (do not worry about the exact shade of yellow).
- The dimensions M and N refer to the outer and inner lengths of the rectangles *excluding* the arrowhead. The dimension letters are only for informational purposes, and not to be drawn.
- The arrowhead has an overhang of 8 pixels on each side, and is a 45-45-90 triangle. For example if the horizontal part of the arrow has height 24, the arrowhead would start at 40 pixels high, and its consecutive columns would have heights $38, 36, \ldots, 2$ (yes, its a little stubby).
- The arrowhead has color given by the input `cr-cg-cb` (see inputs above). Each RGB component of the arrow color (bright green in example) is exactly four times the corresponding RGB component of the arrowhead color. If a color component is not legal due to overflow, you should use the largest possible value for that component.
- The entire picture (including arrow and arrowhead) should be **perfectly centered** in the display.

The above figure is shown for $M = 100$ and $N = 76$ (total width = 120, total height = ~~100~~108) and a full green arrowhead, but your code should of course use whatever values are in `projInput`. Thus, the entire 120X~~100~~108 box (in this example) is centered in the display.

Your program should first check the values given in `projInput`. If it is not possible to draw the figure for these values, your program should draw just the yellow background. Note that a figure that is not exactly centerable (even if it is just half a pixel off) counts as 'not possible'. 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.

You are **not** allowed to use functions for this project, and any code that uses functions will not get credit.

2 Submission Instructions

All of you have been sent email with an account on gradescope. If you submit more than once, only your most recent submission will be graded (and with relevant late penalties). **Gradescope is not an assembler/compiler**, and your program will not be run automatically – making good test cases accounting for error and edge cases is a major part of the project.

The file should follow normal MIPS conventions including comments. 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 functions and multiplication/division (you're probably thinking the wrong way if you think you need these). Note that MARS may treat some seeming 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:

00000000	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
M:                                      .word   --
N:                                      .word   --
cr:                                     .word   --
cg:                                     .word   --
cb:                                     .word   --
# 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×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 the brightest possible 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→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       $t3,15340($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
```

4 Additional Notes

I may update this section with assorted notes, based on student questions.

1. You may use any $s or $t registers for your program if you want.
2. Some earlier editions of this had incorrect colors on the picture (swapping the arrow and arrowhead color). You should follow the text, not the picture; however, we accepted both versions as corect.