CSCI 260 – Project 1

Due: Oct 16 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

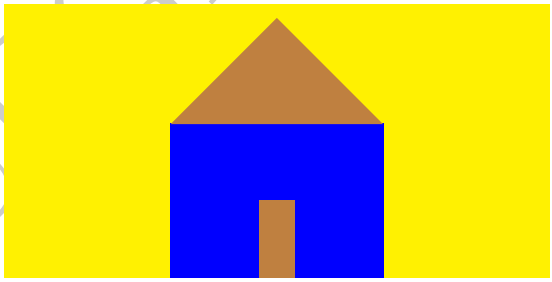
The input will be the assembly language version of the following C structure (details in Section 3):

```
struct projInput {
  unsigned w, h, d;       // size parameters
  unsigned chr, chg, chb; // red-green-blue components for color ch
  unsigned cdr, cdg, cdb; // ... color cd
}
```

You may assume that your implementation of C uses 32-bit ints (so there are 9 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 'house' on the bitmap display supplied as part of MARS:



More information about the picture:

- The background is always yellow (do not worry about the exact shade of yellow).
- The body of the house has width w and height h, and is horizontally centered in the bitmap. It has color ch.
- The door of the house has width d and is exactly half the height of the house body. It is horizontally centered and has color cd.
- The roof is an isosceles triangle with color cd and edges of slope 1 (*i.e.*,perfectly diagonal). It is OK for the top row to be 2 pixels wide (instead of a point).

The above figure is shown for certain colors and sizes, but your code should of course use whatever values are in `projInput`.

Your program should 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 roof is not perfectly diagonal), but that is acceptable and expected.

You do not need to use functions for this project (and I don't suggest it). But if you do, you need to clearly state the arguments in comments and follow *our* calling conventions (including frame pointers).

2 Submission Instructions

Please rename your file to `surnameName.txt` (*e.g.,*doeJohn.txt) and submit it using blackboard (under the Homework menu). Note that a text file is requested since blackboard does not support `.asm` files. Make sure to follow the formatting rules stated in Section 3. Do **not** include your alias in the file.

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 multiplication and 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

Accessing the Bitmap Display

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

```
.data
# DONOTMODIFYTHISLINE
frameBuffer:                    .space  0x80000  # 512 wide X 256 high pixels
w:                              .word   --
h:                              .word   --
d:                              .word   --
chr:                            .word   --
chg:                            .word   --
chb:                            .word   --
cdr:                            .word   --
cdg:                            .word   --
cdb:                            .word   --
# DONOTMODIFYTHISLINE
# Your 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 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→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
```

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`