Summary: Many computer vision applications require computing the similarity between pairs of image pixels, for example, in stereo vision and motion estimation. This assignment explores the function and implementation of several basic conditional execution, nested iteration, and memory access mechanisms to realize a pixel comparison program. The program compares the intensities of eight color pixels that are stored in memory in a packed 24-bit RGB representation and selects the two that most closely match in intensity. The absolute difference in intensity of these two pixels is also determined.

Background: Most digital imaging devices represent color pixel data as a triple of typically eight bit integers representing the intensity of its red, green, and blue components (RGB format). The color of a single pixel can be encoded in a packed representation in a 32-bit word:

31			0
unused	red component	green component	blue component
31 — 24	23 — 16	15 — 8	7 — 0

In this assignment, a palette of eight such values are created and stored in memory. The intensity of a pixel is defined as:

$$I = \frac{(R+G+B)}{3}$$

where R, G, and B are the red, green, and blue component values of the color pixel. Note that R, G, and B are all unsigned integers. Integer division (a/b) yields an integer which is the "whole" part of the result of the division (the remainder is ignored). So, for example, if R=1, G=2, B=2, the intensity is 5/3 = 1.

Your program must find the pair of pixels in the palette of eight colors that are most similar in intensity (i.e., they have the smallest absolute difference in intensity). To compute the minimum intensity difference, all pairs of different colors in the palette must be compared.

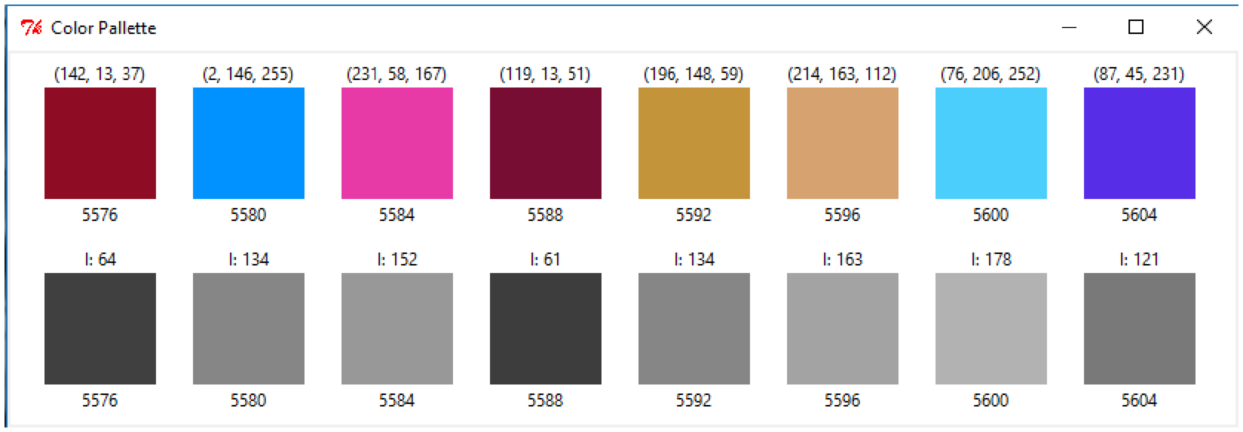

Figure 1: Color Palette generated by swi 589, with (R, G, B) values listed above first row, memory addresses below and the corresponding intensity values for each color in the second row.

Objective: For this assignment, you will write two programs, one in C and one in MIPS. There are three key tasks each color matching program needs to be able to do:

1. Unpack a color by pulling the individual unsigned 8-bit red, green, and blue component values from the lower 24 bits of a 32-bit integer. Hint: in C, the shift (<<, >>) and mask

(&) operators may be useful; in MIPS, the load byte instructions may be helpful (http://www.ece.gatech.edu/academic/courses/ece2035/assignments/Load-Store-Byte-Insts.pdf). Note that the Misasim memory implementation is little endian (the least significant byte is located at the word address).

2. Compute the difference in intensity between two given unpacked colors using the definition of intensity given above.
3. Use nested loops and conditionals to orchestrate the comparison of all pairs of unpacked colors to find the pair with the minimum absolute difference in intensity.

Honor Policy: In all programming assignments, you should design, implement, and test your own code. **Any submitted assignment containing non-shell code that is not fully created and debugged by the student constitutes academic misconduct.** The only exception to this is that you may use code provided in tutorial-0.zip or in the examples on the course website/Canvas as a starting point for your programs (http://ece2035.ece.gatech.edu/examples/index.html or Canvas>Files>Lecture Slides>Code Examples).

All code (MIPS and C) must be documented for full credit.

HW2-1: Design and implement a C version of the intensity match program. As a starting point, use the file `HW2-1-shell.c`. Your C program should print out the difference for the closest pair using the print string provided in the shell file. Name the file **`HW2-1.C`** and upload it to Canvas by the scheduled due date.

The shell program `HW2-1-shell.c` includes a reader function `Load_Mem()` that loads the values from a text file. You should use `gcc` under Ubuntu to develop your program. Sample test files (`test0.txt, test1.txt, etc.`) are provided – the number in the name of each file indicates the correct answer for the min absolute intensity difference. You should compile and run your program using the Linux command lines:

```
> gcc HW2-1.c -g -Wall -o HW2-1
> ./HW2-1 test0.txt
```

You can create additional test files using MiSaSiM to run `HW2-2-shell.asm`, go to the end of the trace, and use the "Dump" memory menu button to save the memory to a text file with the correct answer (the value in $13) in the name of the file.

HW2-2: Design and implement an *efficient* MIPS version of the color match program. How performance is measured and a description of the MIPS library routines you need is given below. Use the file `HW2-2-shell.asm` as a starting point. The results should be returned by your program in **$10** (minimum intensity difference), **$11** (memory address of closest color A), and **$12** (memory address of closest color B). **You must use these register conventions or the automated grader will score your program incorrectly**. Memory should only be used for input to your program. Your program must return to the operating system via the **`jr`** instruction. *Programs that include infinite loops or produce simulator warnings or errors will receive zero credit*. Name the file **`HW2-2.asm`** and upload it to Canvas by the scheduled due date.

Grading for efficiency: The assessment of the MIPS version of your code (HW2-2) will include functional accuracy during 40 trials and performance and efficiency. The code size, dynamic execution length, and operand storage requirements are scored empirically, relative to a baseline solution. The baseline numbers for this project are static code size: **40** instructions, dynamic instruction length: **580** instructions (avg.), total storage required for registers, static memory, and stack memory: **18** words (not including dedicated registers $0, $31, and not including the 8 words for the packed color data). *As a safety net, the dynamic instruction length metric is the maximum of the baseline metric and the average dynamic instruction length of the five fastest student submissions.*

Your score will be determined through the following equation:

$$PercentCredit = 2 - \frac{Metric_{YourProgram}}{Metric_{BaselineProgram}}$$

Percent Credit is then used to scale the number of points for the corresponding points category. For example, if your program uses half as much storage as the baseline, then PercentCredit is 1.5 and the number of points for the storage category (see Project Grading table below) is 5 scaled by 1.5 = 5*1.5 = 7.5. Important note: while the total score for each part can exceed 100%, especially bad performance can earn *negative credit*. The sum of the combined efficiency metrics scores (static code size, dynamic execution length, and storage) will be capped at zero; the sum of that portion of the grade will not be less than zero points. **Finally, these efficiency scores will be reduced by 10% for each incorrect trial (out of 40 trials). You cannot earn points for efficiency metrics if your implementation fails ten or more of the 40 trials**.

MIPS Library Routines: There are two library routines (accessible via the `swi` instruction).

SWI 589: Create Color Palette: This routine generates and displays eight random colors as shown in the example in Figure 1. It also displays the intensity below each of the eight colors in gray scale. It initializes memory with the 8 packed colors in RGB format beginning at the specified base address (e.g., `Array`). INPUTS: $1 should contain the base address of the 8 words (32 bytes) already allocated in memory. OUTPUTS: the 8 words allocated in memory contain the 8 packed RGB color values to be used as input data.

To more easily view the RGB color components in the memory, you can select “hexadecimal” for the display base in MiSaSiM's menu button “Options” (the wrench). For example, for a color whose RGB value is `(60, 5, 13)`, the packed value is `0x3C050D`.

SWI 581: Report Closest Colors: This routine allows you to report your answer and an oracle provides the correct answer so that you can validate your code during testing.

INPUTS: **$10** should contain the minimum absolute intensity difference your program computed across all pairs of colors, **$11** and **$12** should contain the memory addresses of the two closest colors in intensity (and these may be provided in either order).

OUTPUTS: **$13** gives the correct minimum absolute intensity difference, **$14** and **$15** will contain the correct memory addresses of the two closest colors. (These are the oracle’s answers.)

If you call **swi 581** more than once in your code, only the first answer that you provide will be recorded.

On rare occasions, there may be more than one pair of colors with the same minimum intensity difference. In these cases, the autograder will count as correct any of the pairs that have the same minimum intensity difference (even though the oracle will report only one of the pairs).

Grading: The assignment grade will be determined as follows:

part	*description*	*percent*
HW2-1	Intensity Matcher (C code)	
	correct operation	40
	compiles w/out errors or warnings	5
	proper commenting	5
HW2-2	Intensity Matcher (MIPS assembly)	
	correct operation	15
	efficiency: static code size	5
	efficiency: dynamic execution length	15
	efficiency: operand storage requirements	5
	simulates w/out errors or warnings	5
	proper commenting	5
	Total	*100*