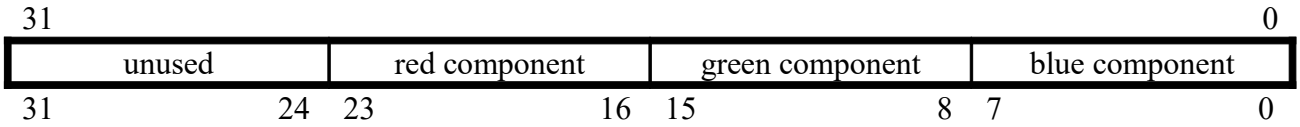
Summary: Many computer vision applications require the comparison of the color of image pixels, for example, in tracking based on object appearance. This assignment explores the function and implementation of several basic conditional execution, nested iteration, and memory access mechanisms to realize a color matching program. The program compares eight colors stored in memory in a packed RGB representation and selects the two that most closely match (as measured in total component difference). The total difference of the closest colors is also determined.

Background: Digital imaging devices often represent color pixel data as a triple of eight-bit integers representing the intensity of a pixels red, green, and blue components (RGB format). In a computer employing 32-bit words, the color of a single pixel can be encoded in a packed representation:



In this assignment, eight such values are created and stored in memory. There are several commonly used metrics to assess color difference. In this case, color differences are measured as the total component difference, also known as the sum of absolute difference (SAD), defined as:

$$ColorDifference_{AB} = |A_r - B_r| + |A_g - B_g| + |A_b - B_b|$$

where each subscript indicates the red (r), green (g), or blue (b) component of color A or B.

To compute the minimum color difference, all combinations of different colors must be compared.

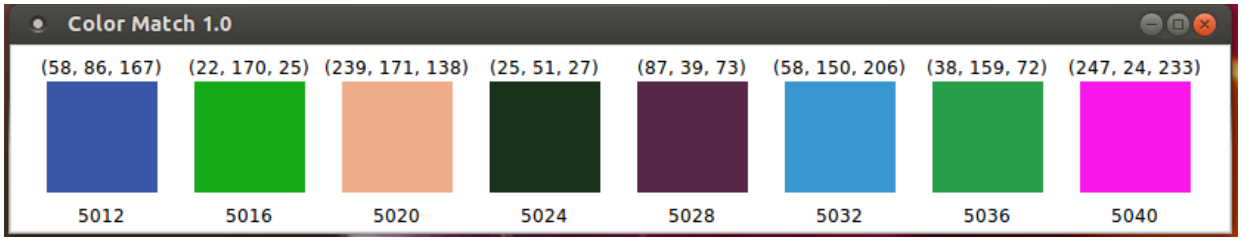


Figure 1: Color Palette generated by swi 500, with (R, G, B) values listed above, memory addresses below.

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. Note that Misasim uses little endian byte ordering: the least significant byte is located at the word address.

2. Compute the color difference (SAD) between two given unpacked colors using the formula 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 SAD.

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 Canvas site as a starting point for your programs (Canvas > Modules > _Module_Name_ > Code Examples).

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

HW2-1: For this part, design and implement a C version of the color match program. As a starting point, use the file `HW2-1-shell.c`. The program should employ a reasonable algorithm that compares all pairings of colors in the palette exactly once. A color should not be compared to itself. Nor should two colors be compared to each other more than once. Your C program should print out the total component color difference for the closest pair using the print string provided in the shell code. 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 (`test119.txt, test30.txt, test111.txt`) are provided – the number in the name of each file indicates the correct answer for the min component difference.

You should compile and run your program using the Linux command lines (this is what the autograder will use):

```
> gcc HW2-1.c -g -Wall -o HW2-1
> ./HW2-1 test119.txt
```

(Replace "test119.txt" with other test file names to run more tests.)

You can create additional test files for your C program using MiSaSiM in this way: 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, as described below) in the name of the file.

Name the file **`HW2-1.C`** and upload it to Canvas by the scheduled due date.

HW2-2: Design and implement a MIPS version of the color match program. 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 total component 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.

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

SWI 500: Create Palette: This routine generates and displays eight random colors as shown in the example in Figure 1. It initializes memory with the 8 packed colors 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.

Hint: 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 total component difference you computed across all pairs of colors, **$11** and **$12** should contain the memory addresses of the two closest colors (and may be provided in either order).
>
> OUTPUTS: **$13** gives the correct minimum total component difference, **$14** and **$15** will contain the correct memory addresses of the two closest colors.

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