Lab 4: How colorful is Your Name

Victor Yuan

Microcontrollers and Embedded Systems

March 1, 2017

Introduction:

The Goal of this lab was to write a program in assembly read in a line of ASCII characters add each asciz character into a array, then convert every 3 asciz characters into the grey scale

Procedure:

I: create rgb array

First create an array with the length of however long your name is rounded up to the nearest multiple of three, with the width of ten rows. Multiple length and width and store this value in r6 then left shifting by 1 and adding the value back into itself, this will have the same result as multiplying r6 by three. Branch Link to malloc to allocate the array in memory. At this point the array pointer is stored in r0 mov this pointer to register r8.

II: initialize array

Right now r8 contains an array of zeros. Initialize the array with the asciz numbers of your name.

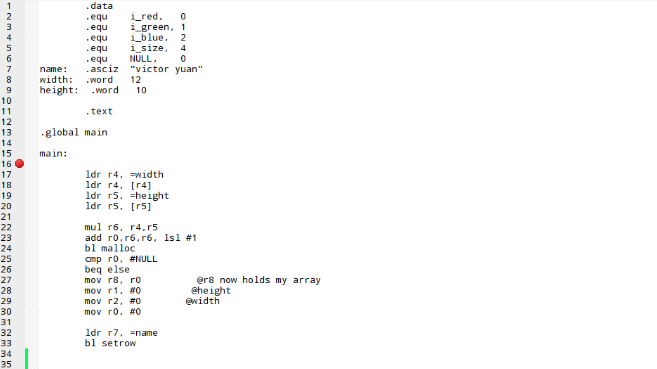
Load your name to r7. Use ldrb to load a single byte to a temporary register r0 for example, load from r7, the register holding the pointer to your name variable. Offset r7 so that the next character can be accessed later. Use strb to store the single asciz number stored in r0 into the first value of [r8] offset this by 1 so that the next empty slot can be accessed later. Using r2 as a counter for how many characters of your name you have entered add 1 onto r2 every time you have initialized a portion of the array. Depending on how long your name is u will need to calculate how many times you will need your loop to loop for. For my name there were 11 characters so I compared r2 to #11 if the condition were set to equal then the program would load my name back into r7 resetting the pointer to the first position it would add a extra offset to the pointer in my array so that it would skip a 0 therefore effectively making my name in the array 12 characters, 11 with a value and the last one 0. And lastly reset the counter back to 0.

I recorded every time the first mini-loop which was used to load a single character in r1 when r1 reached 110, it signified that I had performed the loop 10 times and that 10 rows of my array had been initialized. Move the program counter back to the link register to finish part1 of the lab.

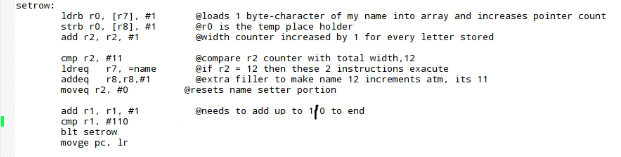
III: Grey scale

Part2 of the lab was converting to grey scale. For this section set r1, 2, and 3 to the values 54, 184, 18. Move the pointer in r8 back 120 spaces so that it begins reading at the start of the array. Use r11 as a tracker. Call your subroutine that converts your array to grey scale. Load 1 byte into r4 the next into r5 and the next in r6 offset for the first 2 and leave the 3rd where it is for now. Multiply r4 with r1(54), r5 with r2(184), r6 with r3(18). Store these values in r7, 8, and 9. Add r7 and r8 and store in r7. Add r7 and r9 and store in r7. Right shift this number by 8 which is the same as diving by 2^8 , 256, this will result in r7 to be the grey scale value. Store this value back into r8 and shift it by 1. Repeat this 40 times for my case. It will be the size of your name rounded up to the nearest multiple of 3 times 10 divided by 3. Spot check this in the memory address of r8.

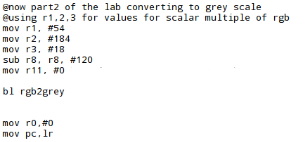
Code:



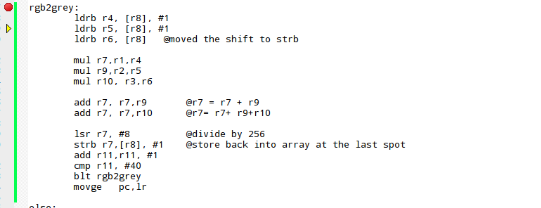
Setting up the array putting name into r7 and calling subroutine setrow, part1 of the lab



Set row subroutine

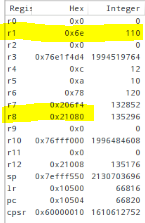


Sets up r1, 2, and 3 as constant values, calls grey scale conversion function

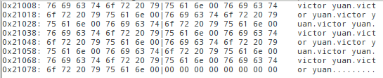


Rgb2grey subroutine

Results:

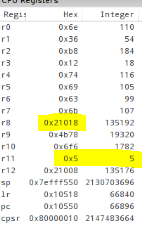
counter r1 is full r2 just finished its cycle

Current state of the array

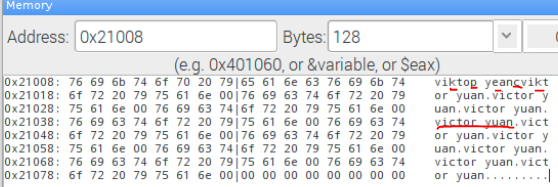


“victor\_yuan “ was repeated 10 times.

Part 1 was successful



The counter r11 shows that it’s in this 5th loop therefore there should be 5 changes



5 letters where changed c-> k , r->p, u-> e, “ ” -> c, and repeats at c->k again.

Part 2 was successful.

Conclusion:

To make a complete rbg2grey conversion I only have to replace the values two spaces before the every change to the same number in every 3rd spot. One thing I learned in this lab is conditional operators can help manipulate a loop very easily