**EC-340**

**Computer Organization and Architecture**



Utkarsh Mahajan 201EC164

R S MUTHUKUMAR 201EC149

Devvrat Ashtaputre 201EC118

**a) Assume that you have an array of 10 elements with base address in $s0. Write an assembly program to find the minimum value from the array and swap it with the last element in the array.**

-> We will make use of word directives to store array into memory with base address labelled as array. For debugging the code, We will use **la** to store the base address at **$s0** as asked to be assumed in the question. We will store index I in register **$t1** initializing it to 0. And then check if it reaches 10 using **slti**. Through each iteration, we will first multiply index(**$t1**) by 4 by left shifting by 2bits and store that offset value in **$t6**. Calculating respective index’s address by adding base + the offset. Then through address we will check if it’s the minimum value until now using **slt** and if it is then we will jump to **min** where we will store the value and its index in **$t7** and **$t0** respectively, we will increment at the end of the loop and then jump to loop label where the condition will be checked again.

Once the most minimum value has been found in the array after iterating through the array. Through **beq**, We will jump to **swapmin**. Here we will swap the minimum value with the last value of the array.

Later jumping to **printl.** Looping similarly, we will later print the updated array into the console using **syscall** with respective **$v0** and **$a0** values then jump to done which will jump to **$ra** (return address)

Assembly code:

        .data

*#  storing required data into memory*

array:  .word 67 43 3 7 2 35 9 62 4 8

        .text

        .globl main

main:

        li      $t1, 0              *# i (index) = 0*

        la      $s0, array          *# fetch base address*

*# initializing minimum =  storing a[0] value in minimum*

        lw      $t0, 0($s0)

        li      $t7, 0              *# index of minimum*

*# j (index) = 0 for printing updated array*

        li      $t9, 0

loop:   slti    $t3, $t1, 10         *# if i == 10 goto done*

        beq     $t3, $zero, swapmin

        sll     $t6, $t1, 2         *# offset = index \* 4*

        add     $t5, $s0, $t6       *# address = base\_address + offset;*

        lw      $t4, 0($t5)         *# t4= arr[i]*

        slt     $t2, $t4, $t0       *# setting less than in t2*

*# switching to min branch if a[i]<current\_min*

        bne     $t2, $zero, min

*# label b\_loop for returning after min has been updated*

b\_loop: addi    $t1, $t1, 1         *# i++*

        j       loop

*# min updates the minimum value and the index containing it*

min:    add     $t0, $zero, $t4     *# updating minimum value*

*# updating index of minimum so that we can swap values letter*

        add     $t7, $zero, $t1

        j      b\_loop

*# for printing the updated array*

printl: slti    $t3, $t9, 10         *# if i == 10 goto done*

        beq     $t3, $zero, done     *# jumping to done if all elements are printed*

        sll     $t6, $t9, 2         *# offset = index \* 4*

        add     $t2, $s0, $t6       *# address = base\_address + offset;*

        lw      $t4, 0($t2)        *# t4 = arr[i]*

        li      $v0, 1              *# for printing*

        move    $a0, $t4            *# print arr[i]*

        syscall

        li      $v0, 11             *# print space character*

        li      $a0, 32

        syscall

        addi    $t9, $t9, 1         *# i++*

*# jump back to printl to print all values of 2nd array*

        j       printl

*# for swapping the minimum and last array*

swapmin:

        lw      $t4,  36($s0)     *# saving the last value of the array*

        sll     $t6, $t7, 2        *# offset = min\_index \*4*

        sw      $t0, 36($s0)        *# storing min value in last position*

        add     $t5, $s0, $t6       *# min\_address = base + offset*

*# storing the value of last element in the position of minimum value*

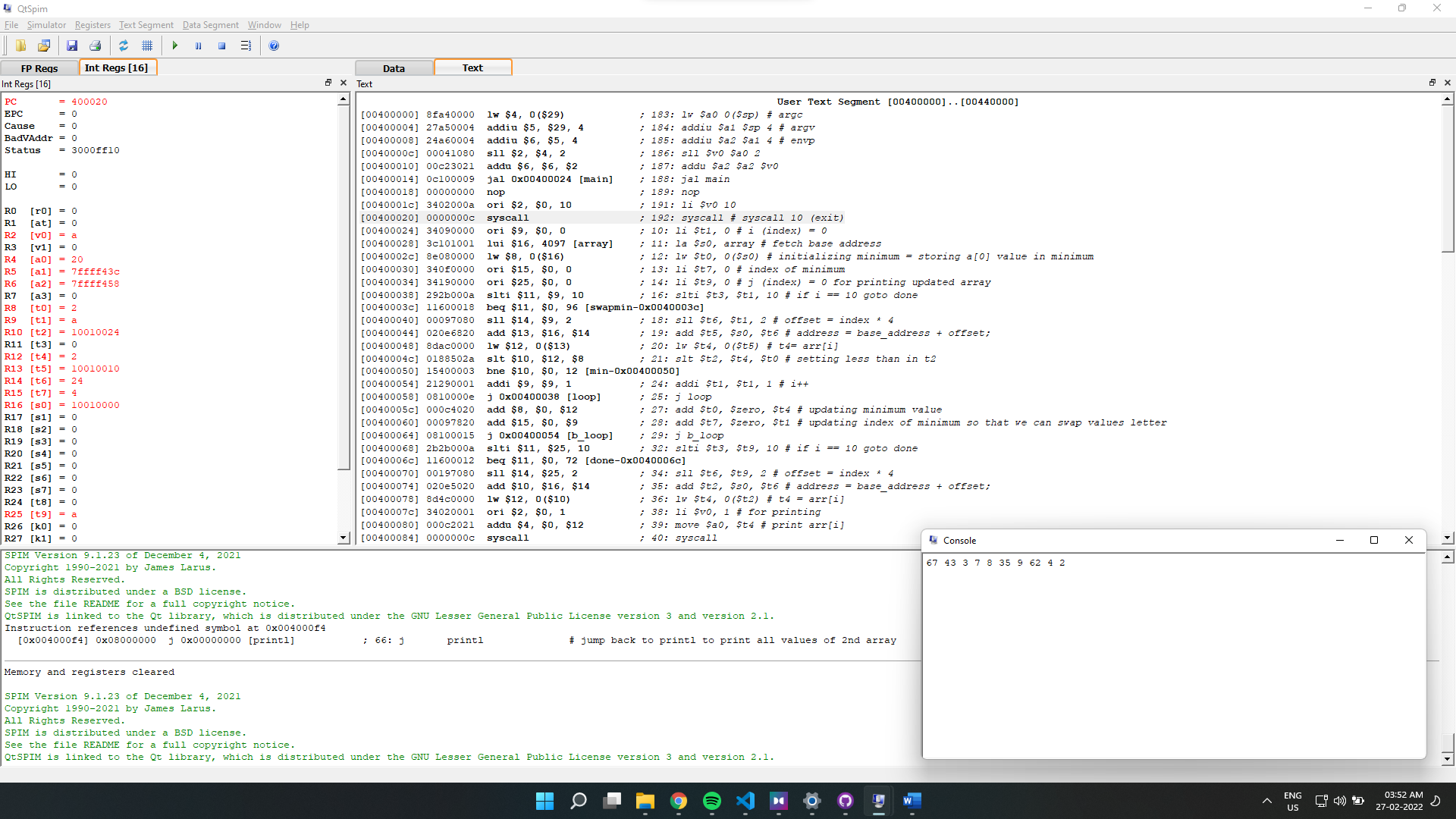
        sw      $t4, 0($t5)

        j       printl          *# to print the updated array*

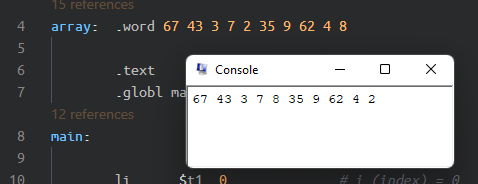
done:

        jr $ra                      *# return from main*

Output:



Console output and input array value:



**b)** **Assume that you have an array of 10 elements with base address in $s0. Assume that the base address of a second array is in $t0. Write an assembly program to copy the elements from the first array to the second.**

->

We will make use of word directives to store first and second array labelling them as **firstarray** and **secondarray** respectively. For debugging the code, We will use **la** to store the base addresses at **$s0** and **$t0** respectively as asked to be assumed in the question. We will store index i in register **$t1** initializing it to 0. And then check if it reaches 10 using **slti**.

Through each iteration, we will first multiply index(**$t1**) by 4 by left shifting by 2bits and store that offset value in **$t6**. Calculating respective index’s address by adding base + the offset. We will store the element from first array to second array by loading a element with **lw** and storing into second array one with **sw**. we will increment index at the end of the loop and then jump to loop label where the condition will be checked again.

Once the most minimum value has been found in the array after iterating through the array. Through **beq**, we will jump to **swapmin**. Here we will swap the minimum value with the last value of the array.

Later jumping to **printl.** Looping similarly, we will later print the updated array into the console using **syscall** with respective **$v0** and **$a0** values then jump to done which will jump to **$ra** (return address)

Assembly Code:

        .data

*#  storing required data into memory*

firstarray:  .word 65 12 56 145 41 311 425 601 48 895264

secondarray:  .word 4 0 0 0 3 0 0 0 0 5

        .text

        .globl main

main:

        li      $t1, 0              *# i (index) = 0*

        li      $t7, 0              *# j (index) = 0 for print*

        la      $s0, firstarray     *# fetch base address*

*# initializing minimum =  storing a[0] value in minimum*

        la      $t0, secondarray

*# label for looping throughtout the array and copying to second array*

loop:   slti    $t3, $t1, 10         *# if i == 10 goto done*

*# after completition, jump to printl to print 2nd array*

        beq     $t3, $zero, printl

        sll     $t6, $t1, 2         *# offset = index \* 4*

        add     $t5, $s0, $t6       *# address\_1 = base\_address + offset;*

        lw      $t4, 0($t5)         *# t4= memory[address\_1]  /// t4 = arr\_1[i]*

        add     $t2, $t0, $t6       *# address\_2 = base\_address + offset;*

        sw      $t4, 0($t2)        *# memory[address\_2] = t4 /// arr\_2[i]= t4*

        addi    $t1, $t1, 1         *# i++*

        j       loop

*# for printing the second array*

printl: slti    $t3, $t7, 10         *# if i == 10 goto done*

        beq     $t3, $zero, done

        sll     $t6, $t7, 2         *# offset = index \* 4*

        add     $t2, $t0, $t6       *# address\_2 = base\_address + offset;*

        lw      $t4, 0($t2)        *# t4 = arr\_2[i]*

        li      $v0, 1              *# for printing*

        move    $a0, $t4            *# print arr\_2[i]*

        syscall

        li      $v0, 11             *# print space character*

        li      $a0, 32

        syscall

        addi    $t7, $t7, 1         *# i++*

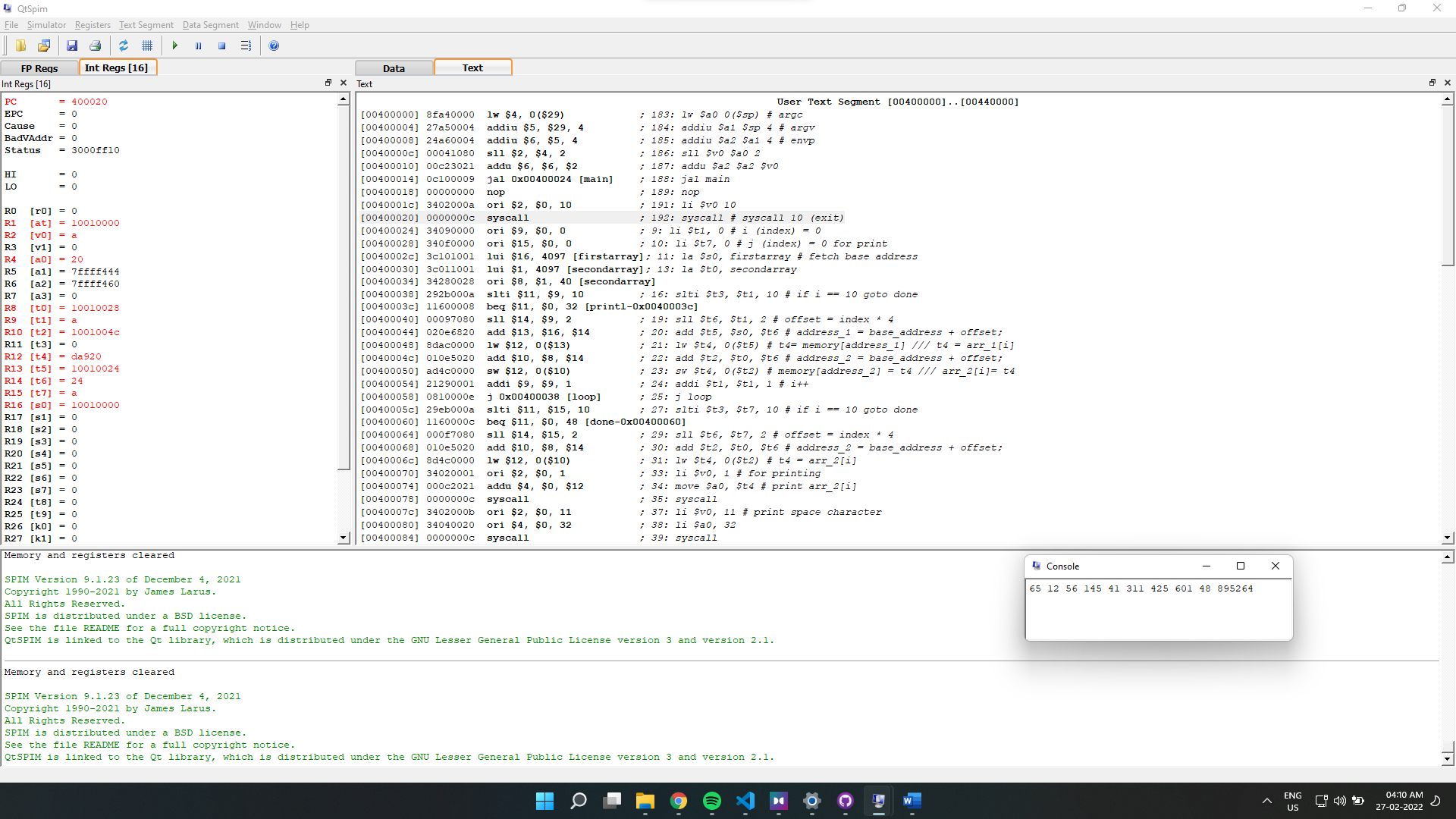
*# jump back to printl to print all values of 2nd array*

        j       printl

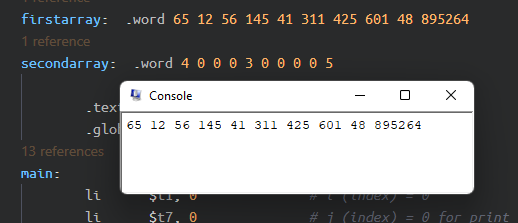
done:

        jr      $ra               *# return from main*

Output:



Console output and input array value:



**c) Write an assembly program to convert red-green-blue (RGB) values for a set of pixels into a single gray value per pixel. You are given an array called pixels, each element of which is a 32-bit word representing a color value. The lowest 8 bits of each color value denote an unsigned integer representing the BLUE value, the next 8 bits are the GREEN value, the next 8 bits are the RED value, and the most significant 8 bits are all zeroes. gray value = (red + green + blue) / 3 (integer divide and truncate). Use a separate procedure rgb2gray and print each RGB value and the corresponding gray value on the console.**

-> Similar to the codes above, we will store the needed values into memory, here for debugging under .data section using data directives like .asciiz (for strings) and .word.

We will use arguments **$a2** and **$a3** for indexing. Store base address for RGB input array in **$s0** and for output gray array in **$s1** .

Since we will be making a procedure call, we will need to save the return address of the main. For that we will allocate 32 bit space by decrementing stack pointer(**$sp**) by 4(in bytes). And store the return address in it using **lw**.

We will jump to the procedure **rgb2gray** using function jal which will update the **$ra** (return address). Inside the **rgb2gray**. We will loop until index i( stored in **$a3**) reaches 10. We will increment I at every end of the loop.

In the loop, At first we will calculate offset by multiplying index by 4 (left shifting by 2 bits) and storing it in a temporary variable **$t2**. Then for getting the i’th element of the rgb array. we will calculate its respective address by adding base(**$s0**) and offset(**$t2**) and store it in **$t4**. Loading the rgb value in **$t5**. We will use **andi** with immediate **#0xff**, to get the lowest significant byte of it which is the value of **blue** and store it in a temporary variable **$t6**. Later right shifting **$t5** by a byte. We get **Green** value at Lowest significant byte. This can again be separated using **andi**, and then will be added to **$t6**. Repeating the right shift, we will have red value at lowest significant byte. Which will be separated similarly using **andi**. Then added to **$t6**.

We will have all the R, G and B values summed up in **$t6**. We will divide it by 3 using **div** and store the quotient in **$t6** itself. Later to store the value from the temporary register **$t6** into the respective I’th element of gray code’s array. by calculating address for the I’th element similar to what we did for RGB array elements. But here we will store the value (using sw) and not load.

After Gray code values of all the 10 RGB code values have been calculated, we will jump to **printl**. Where with the help of **syscall** and similar looping, We will print all the RGB code and its gray code respectively. After the looping it will jump to done which will return to the next address of where the rgb2gray was called. Here, the main’s return address is restored in **$ra** return address

From stack pointer and then the space allocated for it is deallocated by incrementing the stack pointer. After it, the main function will return and the program ends.

Assembly Code:

        .data

*#  storing required data into memory*

rgb:  .word 0xbf49ac 0x68dba6 0xFF0A5D 0x06c012 0x1428e5 0x1ec2b2 0x5e7386 0xae770f 0x286dd0 0x8b10b6

gray:  .word 4 0 0 0 3 0 0 0 0 5

*#  storing strings required to print at the end*

trgb:   .asciiz "RGB: "

tgray:  .asciiz " GRAY: "

        .text

        .globl main

main:

        li      $a2, 0              *# i (index) = 0*

        la      $s0, rgb     *# fetch base address*

*# initializing minimum =  storing a[0] value in minimum*

        la      $s1, gray

        li      $s3, 3       *# to store 3 to divide later*

        li      $a3, 0       *# j index =0 for printing*

        addi    $sp, $sp, -4 *# allocating space on stack*

        sw      $ra, 0($sp)  *# storing main's return address*

        jal     rgb2gray     *# jumping (with link) to rgb2gray function*

        lw      $ra, 0($sp)  *# loading back main's return address*

        addi    $sp, $sp, 4  *# deallocating space on stack*

        jr      $ra          *# returning from main. END OF PROGRAM*

*# procedure(Function) rgb2gray*

rgb2gray:   slti    $t3, $a2, 10         *# if i == 10 goto done*

        beq     $t3, $zero, printl

        sll     $t2, $a2, 2         *# offset = index (i) \* 4*

*# address\_1 = base\_address + offset;*

        add     $t4, $s0, $t2

*# t5= memory[address\_1]  /// t5 = rgb[i]*

        lw      $t5, 0($t4)

*# t6 = rgb[i][7:0] = blue value // last 7 bits by using AND*

        andi    $t6, $t5, 0xFF

*# right shift t5 by 8 so GREEN comes at lowest significant byte.*

        srl     $t5, $t5, 8

*# t7 = rgb[i][7:0] = green value  // last 7 bits by using AND*

        andi    $t7, $t5, 0xFF

*# t6 = t7 + t6 // t6 = blue + green*

        add     $t6, $t7, $t6

*# right shift t5 by 8 so RED comes at lowest significant byte.*

        srl     $t5, $t5, 8

*# t7 = t5[i][7:0] = green value  // last 7 bits by using AND*

        andi    $t7, $t5, 0xFF

*# t6 = t7 + t6 // t6 = blue + green + red*

        add     $t6, $t7, $t6

*# t6 = t6 / t8 = (blue + green + red) /3*

        divu    $t6, $t6, $s3

*# address\_2 = base\_address + offset;*

        add     $t4, $s1, $t2

*#  memory[address\_1]= t6  /// gray[i] = t6 = (r+g+b)/3*

        sw      $t6, 0($t4)

        addi    $a2, $a2, 1         *# i++*

        j       rgb2gray

*# for printing RGB values and its respective Gray values*

printl: slti    $t3, $a3, 10         *# if i == 10 goto done*

        beq     $t3, $zero, done

        sll     $t6, $a3, 2         *# offset = index \* 4*

*# rgb\_address = base\_rgb\_address + offset;*

        add     $t2, $s0, $t6

        lw      $t4, 0($t2)        *# t4 = rgb[i]*

        li      $v0, 4

        la      $a0, trgb            *# printing string "RGB: "*

        syscall

        li      $v0, 1              *# for printing*

        move    $a0, $t4            *# print rgb[i]*

        syscall

        li      $v0, 4

        la      $a0, tgray           *# printing string " GRAY: "*

        syscall

*# gray\_address = base\_gray\_address + offset;*

        add     $t2, $s1, $t6

        lw      $t4, 0($t2)        *# t4 = gray[i]*

        li      $v0, 1              *# for printing*

        move    $a0, $t4            *# print gray[i]*

        syscall

        li      $v0, 11             *# print newline character*

        li      $a0, 0x0a

        syscall

        addi    $a3, $a3, 1         *# i++*

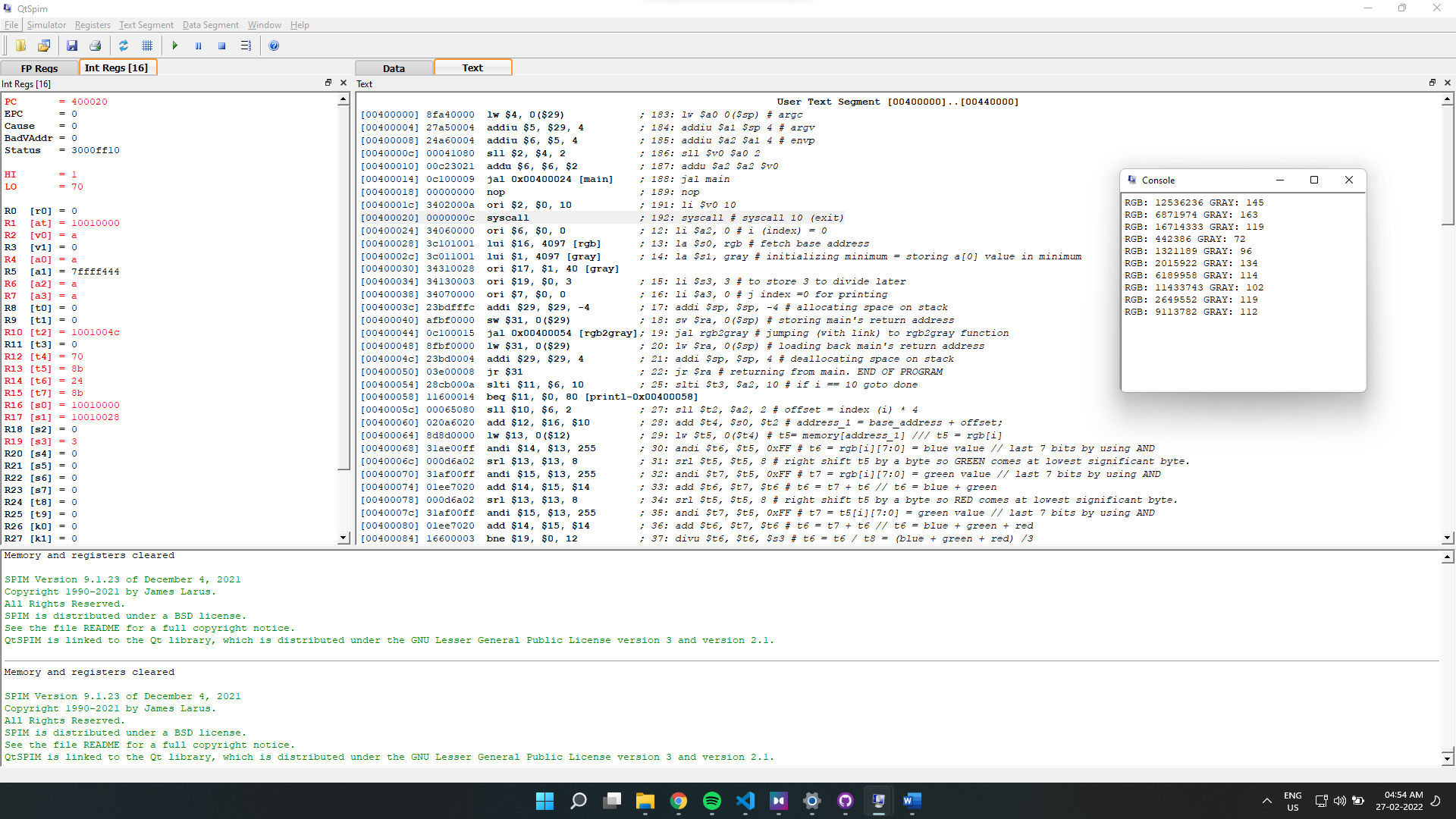
*# jump back to printl to print all values of 2nd array*

        j       printl

done:

        jr      $ra   *# return from rgb2gray to main*

Output:



**Console output and input array value:**

Note: The console displays values in decimal values and not in hex.

