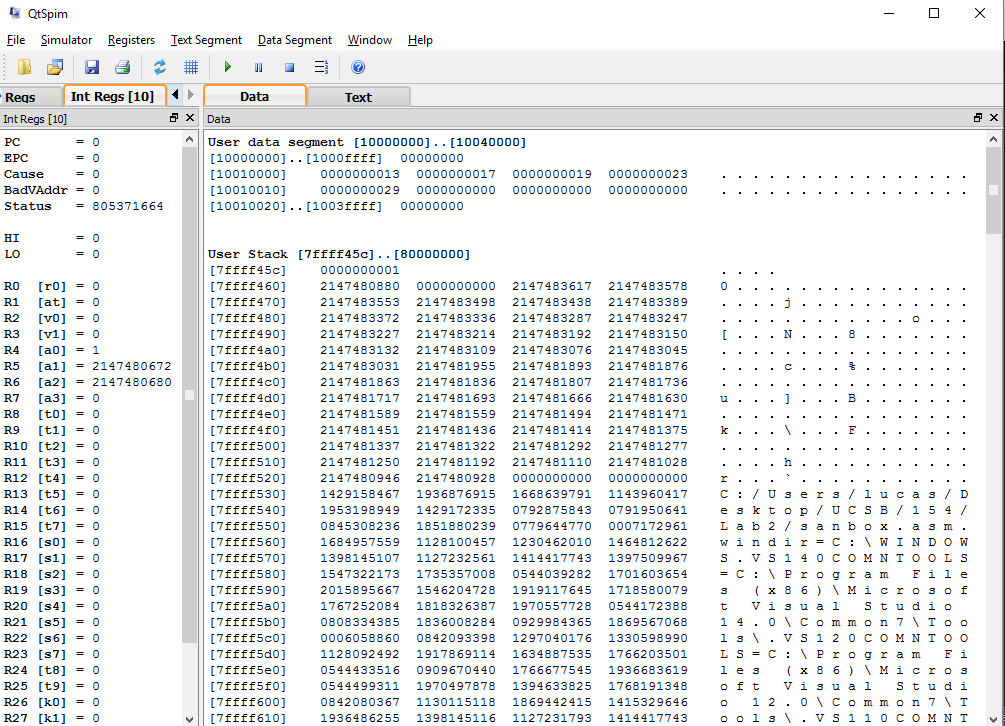
Lucas Lopilato

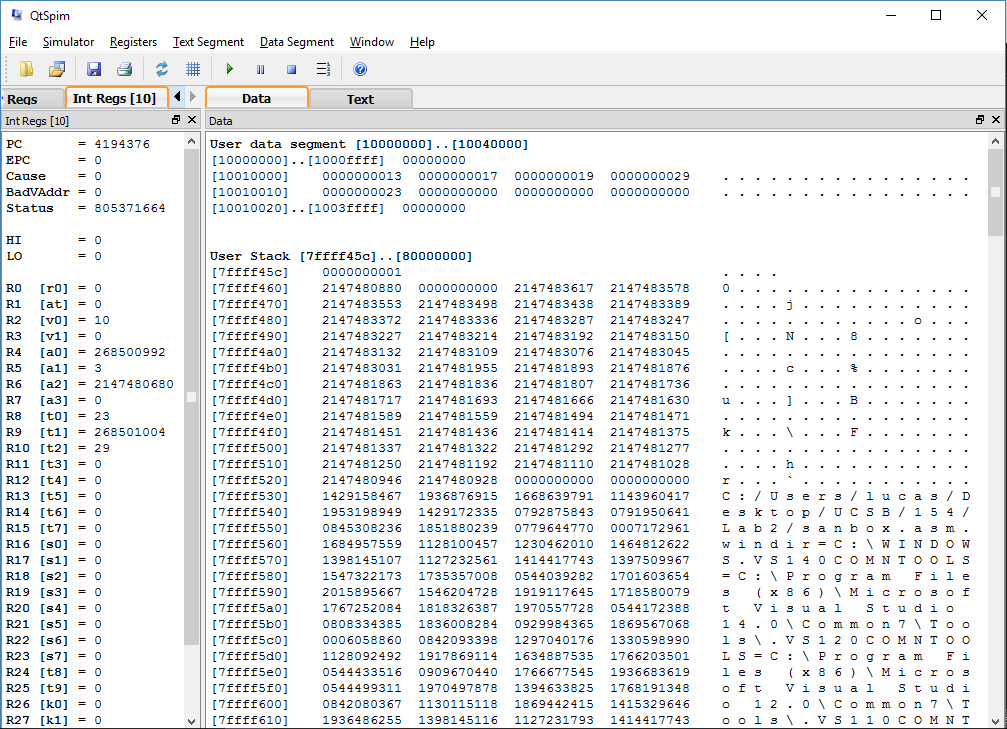
328886-7

CS154 Lab 2

1. Code Attached on Next Page



The above image is prior to running the attached code. The program has the data section which contains a visual representation of data and addresses, a register window (base 10 in this case) to show values stored in registers, and a text section which displays the code loaded. As you can see in the User Data Segment, beginning at address 10010000, the array has each integer sequentially allocated. The address of this array is loaded into a register as commented by the code. Note the original data array is 13,17,19,23,29.



As you can see after the code is ran, the new array is 13,17,19,29,23 (29 and 23 have been swapped). All of the registers either possess memory values, values, or offsets that were used in the calculation as described in the commented code.

**Source Code**

# Load 3 into $a1

li $a1, 3

# Stores the byte offset of an int array index k

# $t1 now holds 4 \* k

sll $t1, $a1, 2

# Adds the byte offset to v and stores the result in $t1

add $t1, $a0, $t1

# Loads the value of array[k] into $t0

lw $t0, 0($t1)

# Loads the value of array[k+1] into $t2

lw $t2, 4($t1)

# Stores the value of array[k+1] into array[k]

sw $t2, 0($t1)

# Stores the old value of array[k] into array[k+1]

sw $t0, 4($t1)

li $v0, 10 # Sets $v0 to "10" to select exit syscall

syscall # Exit



**Source Code**

# Lucas Lopilato

# CS154 Lab 2

# Question 2

# Defines global function main

.globl main

.data

# Define the array of ints

array: .word 4, 1, 3, 9, 2, 5, 8, 10, 6, 7, 14, 13, 12, 11

.text

main:

la $a0, array

addi $a1, $zero, 14

sort:

# Assumes v is originally in a0 and n is in a1

# Assign s1 as v

move $s1, $a0

# Assign s2 as n

move $s2, $a1

# Initialize i as $s3

move $s3, $zero

outerforcondit:

# Check for condition

bge $s3, $s2, afterouterfor

# Initialize j as $s4

addi $s4, $s3, -1

innerforcondit:

# Check for first condition

blt $s4, $zero, afterinnerfor

# Get address for v[j]

sll $t2, $s4, 2

add $t2, $t2, $s1

# Load v[j] and v[j+1] into $t3 and $t4 resp.

lw $t3, 0($t2)

lw $t4, 4($t2)

#check second condition

ble $t3, $t4, afterinnerfor

# Prepare to call swap

move $a1, $s4

jal swap

# Update j

addi $s4, $s4, -1

# Jump Back to the Condition Check

j innerforcondit

afterinnerfor:

# Increment I

addi $s3, $s3, 1

# Jump Back to check condition

j outerforcondit

afterouterfor:

li $v0, 10 # Sets $v0 to "10" to select exit syscall

syscall # Exit

swap:

# Stores the byte offset of an int array index k

# $t1 now holds 4 \* k

sll $t1, $a1, 2

# Adds the byte offset to v and stores the result in $t1

add $t1, $a0, $t1

# Loads the value of array[k] into $t0

lw $t0, 0($t1)

# Loads the value of array[k+1] into $t2

lw $t2, 4($t1)

# Stores the value of array[k+1] into array[k]

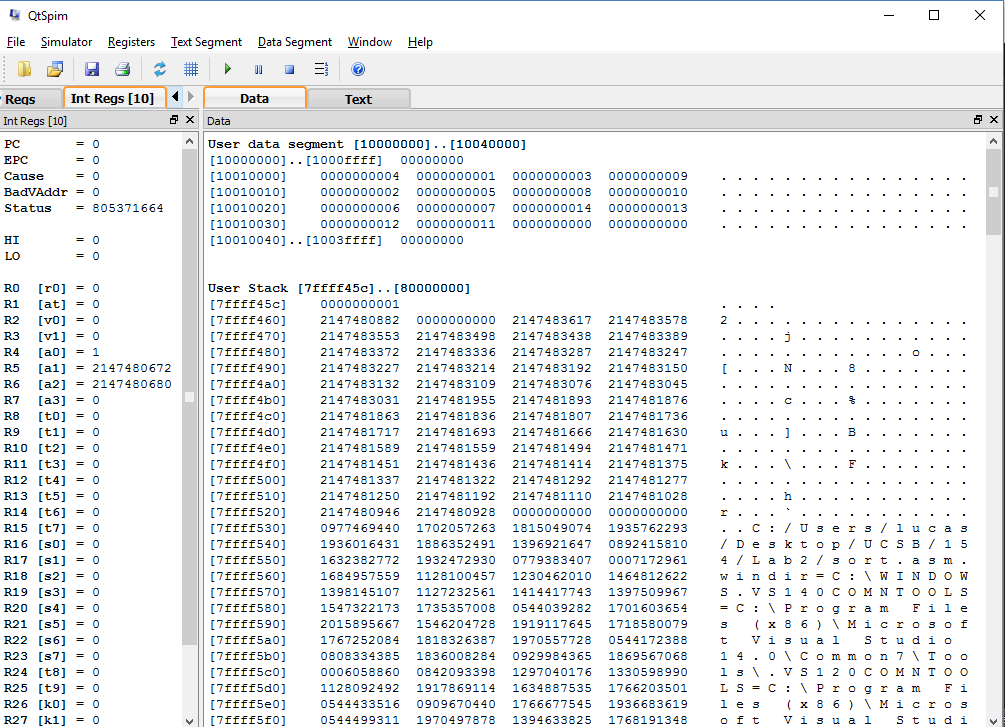
sw $t2, 0($t1)

# Stores the old value of array[k] into array[k+1]

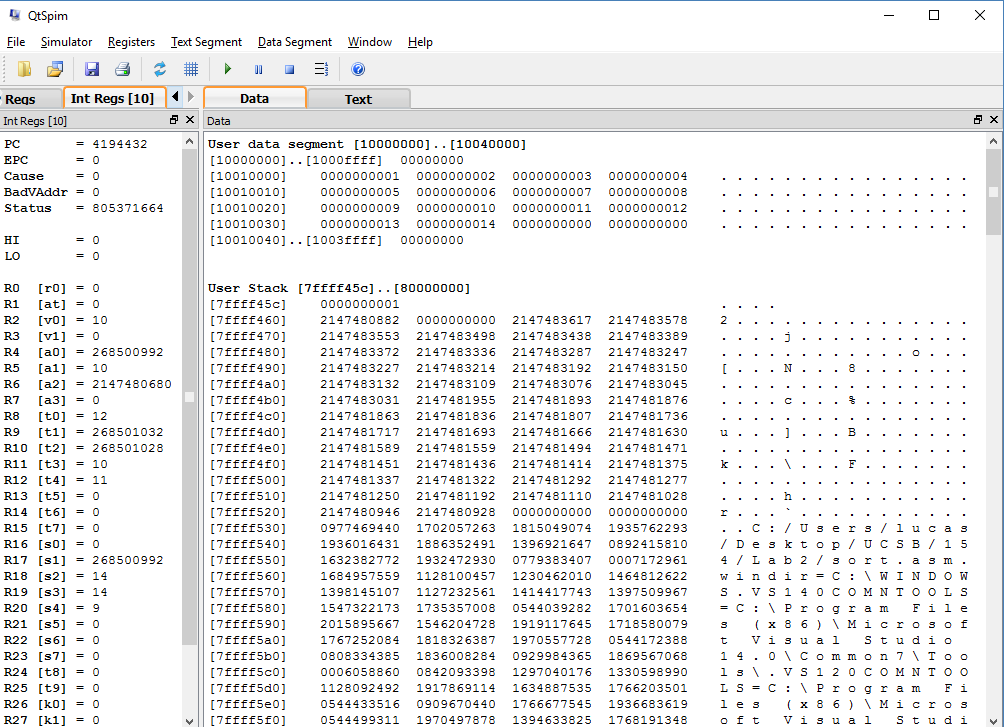
sw $t0, 4($t1)

jr $ra # Sets $v0 to "10" to select exit syscall

**SPIM**



The above SPIM screenshot shows the status of the data before calling the method sort. In the User data segment section, you can see an unsorted array of integers in decimal notation.



The above screenshot shows the result of the array afterwards, where the array is now sorted.