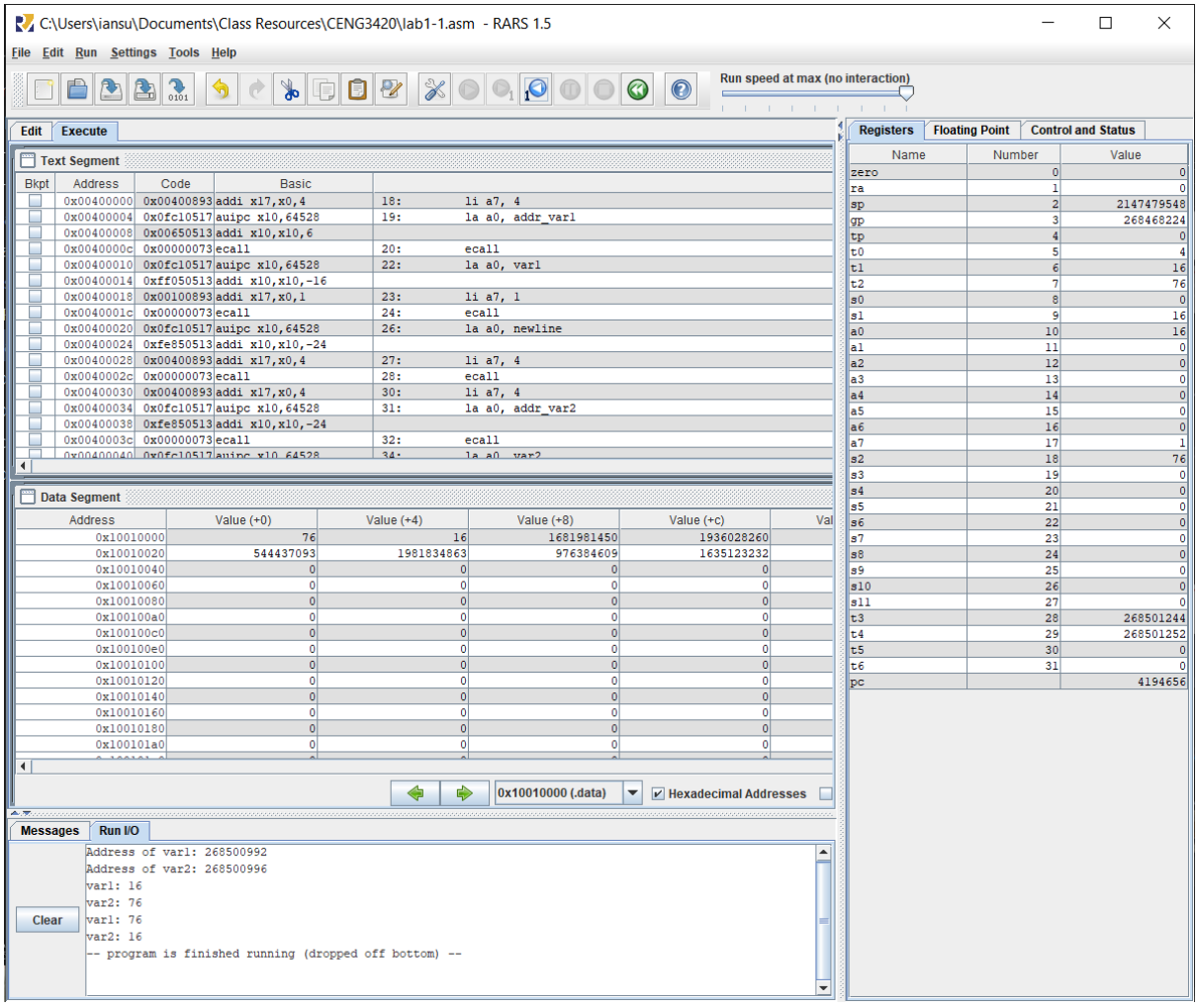
CENG3420 Lab1 Report

Ian Ha Jin Quan (1155138078)

Lab1-1

I use `la` to load the address of var1 and var2 to a0 and print them out using system call. Then I load their value to s1 and s2 respectively. I use `addi` to increment s1 by 1. I use the `addi t0, zero, 4` to load a value 4 into t0, and use `mul s2, s2, t0` to multiply s2 by 4. Then I use `sw s1, var1, t1` and `sw s2, var2, t2` to store the value back to var1 and var2 respectively, and use syscall to print them out. For the swap operation, I first load var1 and var2 to t1 and t2 respectively, then I store t1 to var2 and t2 to var 1 using similar command as above, and print them out.

Console result:



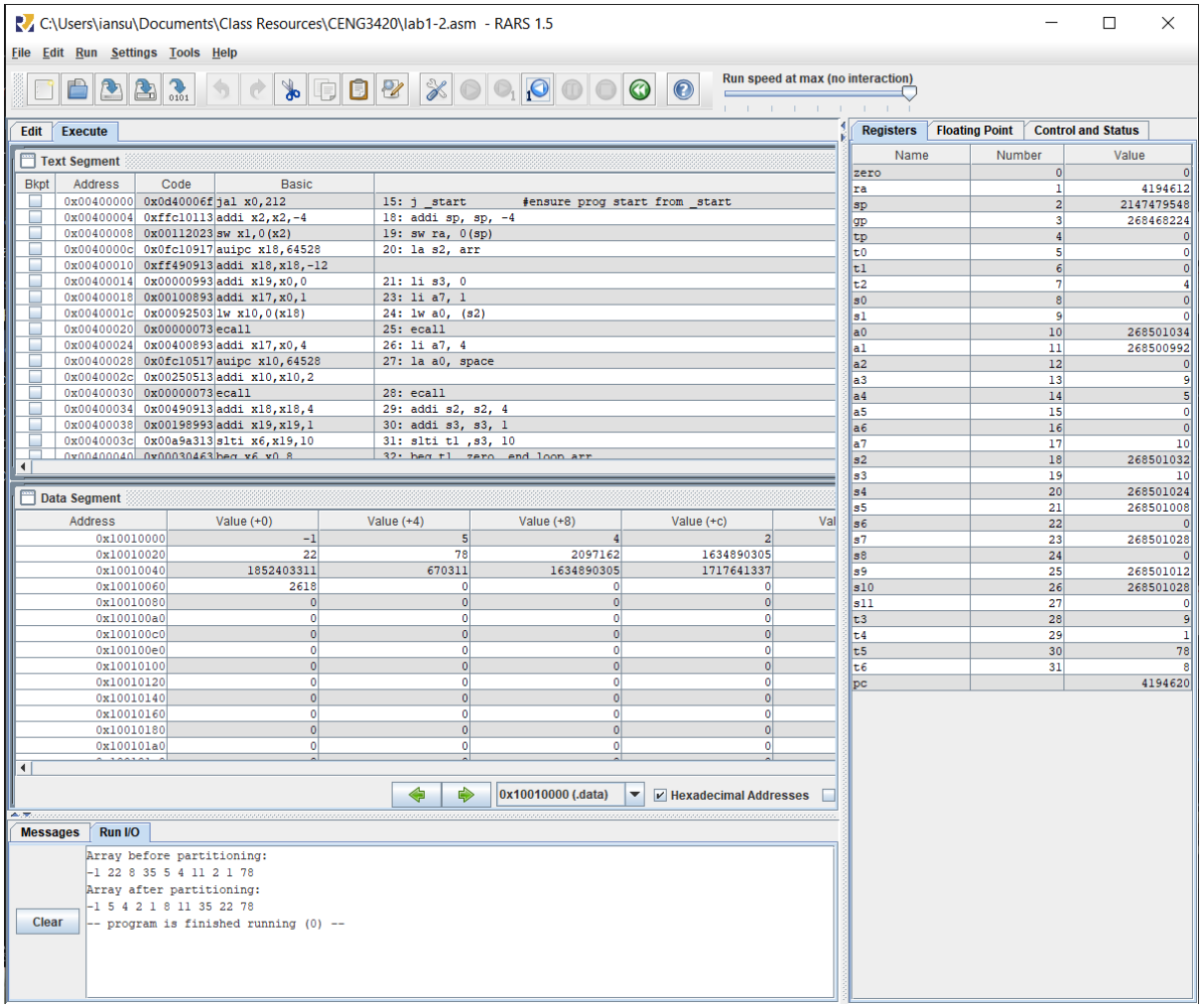
Lab1-2

In this section, I followed the pseudo code given in the slides,

```
1: function PARTITION(A, lo, hi)
2:     pivot ← A[hi]
3:     i ← lo-1;
4:     for j = lo; j ≤ hi-1; j ← j+1 do
5:         if A[j] ≤ pivot then
6:             i ← i+1;
7:             swap A[i] with A[j];
8:         end if
9:     end for
10:    swap A[i+1] with A[hi];
11:    return i+1;
12: end function
```

First, I swap the 3rd element (8) to the last element. I then load the address of array into a1, load 0 to a2 indicating the lo, load 9 into a3 indicating high, and call the `part` function. In the part function, I first load the value of last element (pivot) into t1, then I initialise t2 as lo-1 (i), t3 as lo (j). Then I loop through the array, incrementing i and swap array[i] and array[j] if array[j] is smaller than the pivot. Before returning, I swap the last element with element i+1, completing the partition.

Console result:



Lab1-3

For this part, I reused the partition function from the last part. For the recursive quicksort function, I followed the pseudo code given in the slides:

```
1: function QUICKSORT(A, lo, hi)
2:     if lo < hi then
3:         p ← partition(A, lo, hi);
4:         quicksort(A, lo, p - 1);
5:         quicksort(A, p + 1, hi);
6:     end if
7: end function
```

```
42  sort:
43  bge a2, a3, sort_return #return if lo>=hi
44  addi sp, sp, -16
45  sw ra, 0(sp)
46
47  add s9, a2, zero #load lo to s9
48  add s10, a3, zero #load hi to s10
49
50  jal part
51
52  add s11, a4, zero #load pivot to s11
53
54  sw s9, 4(sp) #s9 = lo
55  sw s10, 8(sp) #s10 = hi
56  sw s11, 12(sp)#s11 = pivot
57
58  #quicksort left
59  add a2, s9, zero#lo = lo
60  addi a3, s11, -1 #hi = pivot-1
61  jal sort
62
63  #quicksort right
64  addi a2, s11, 1 #lo = pivot + 1
65  add a3, s10, zero #hi = hi
66  jal sort
67
68  lw ra, 0(sp)
69  lw s9, 4(sp)
70  lw s10, 8(sp)
71  lw s11, 12(sp)
72  addi sp, sp, 16
73  sort_return:
74  jr ra
75
```

I reserved additional 12 bytes of space in the stack for storing s9, s10 and s11 as the lo, hi ,and pivot position to prevent the data from getting lost in the recursive calls. I also added a return value in the partition function: loading a4 with the value of i+1, indicating the pivot position after the partition.

Console result: