

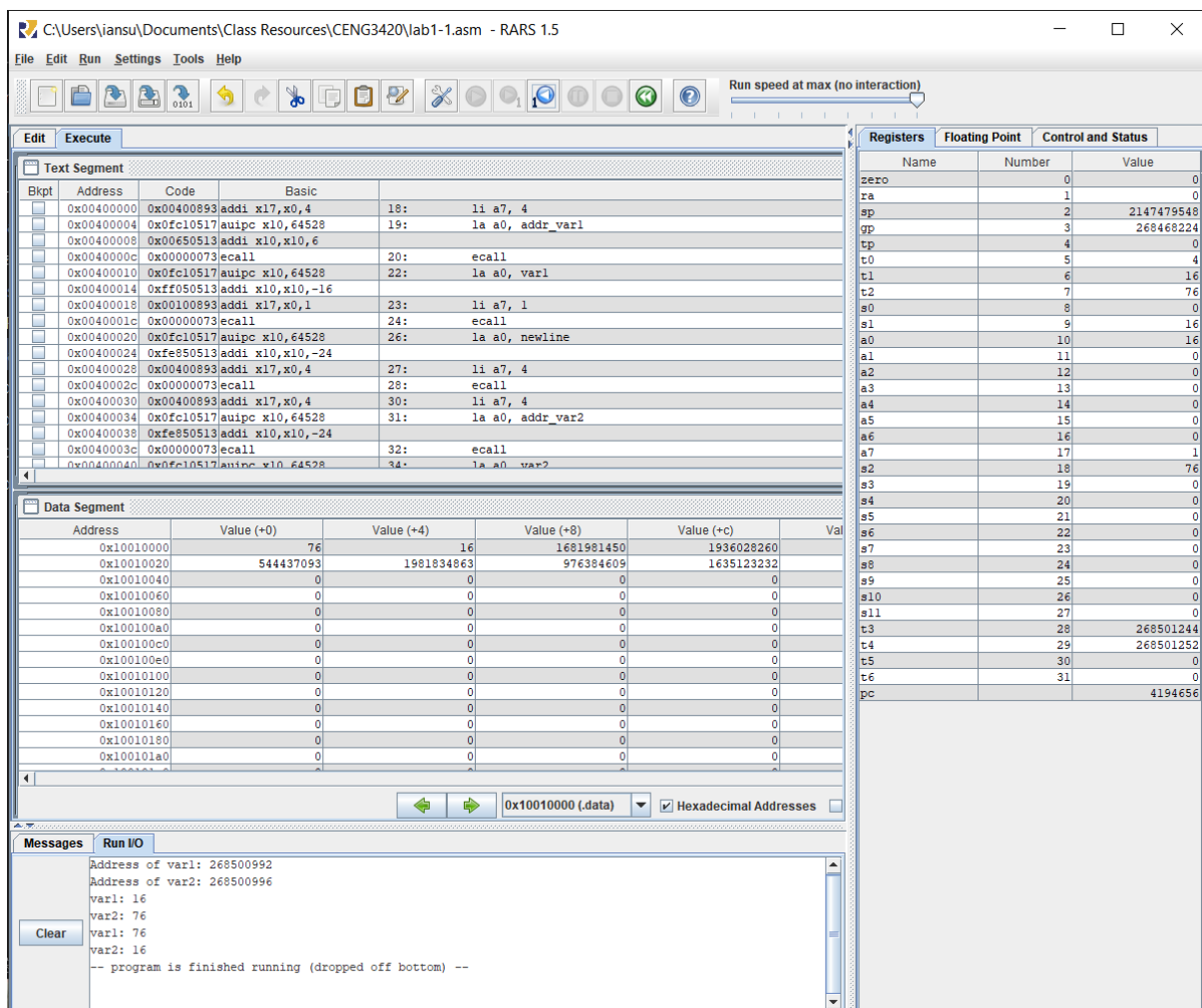
# CENG3420 Lab1 Report

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 `var1` 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 3<sup>rd</sup> 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:

**Text Segment**

Bkpt	Address	Code	Basic
	0x00400000	0x0d40006f jal x0,212	15: j _start #ensure prog start from _start
	0x00400004	0xffc10113 addi x2,x2,-4	18: addi sp, sp, -4
	0x00400008	0x00112023 sw x1,0(x2)	19: sw ra, 0(sp)
	0x0040000c	0x0fc10917 auipc x18,64528	20: la s2, arr
	0x00400010	0xff490913 addi x18,x18,-12	
	0x00400014	0x00000993 addi x19,x0,0	21: li s3, 0
	0x00400018	0x00100893 addi x17,x0,1	23: li a7, 1
	0x0040001c	0x000092503 lw x10,0(x18)	24: lw a0, (s2)
	0x00400020	0x00000073 ecall	25: ecall
	0x00400024	0x00400893 addi x17,x0,4	26: li a7, 4
	0x00400028	0x0fc10517 auipc x10,64528	27: la a0, space
	0x0040002c	0x00250513 addi x10,x10,2	
	0x00400030	0x00000073 ecall	28: ecall
	0x00400034	0x00490913 addi x18,x18,4	29: addi s2, s2, 4
	0x00400038	0x00198993 addi x19,x19,1	30: addi s3, s3, 1
	0x0040003c	0x00a9a313 slti x6,x19,10	31: slti t1, s3, 10
	0x00400040	0x00030463 beq v6,v0,2	32: beq t1, zero, end loop arr

**Data Segment**

Address	Value (+0)	Value (+4)	Value (+8)	Value (+c)	Val
0x10010000	-1	5	4	2	
0x10010020	22	78	2097162	1634890305	
0x10010040	1852403311	670311	1634890305	1717641337	
0x10010060	2618	0	0	0	
0x10010080	0	0	0	0	
0x100100a0	0	0	0	0	
0x100100c0	0	0	0	0	
0x100100e0	0	0	0	0	
0x10010100	0	0	0	0	
0x10010120	0	0	0	0	
0x10010140	0	0	0	0	
0x10010160	0	0	0	0	
0x10010180	0	0	0	0	
0x100101a0	0	0	0	0	

**Registers**

Name	Number	Value
zero	0	0
ra	1	4194612
sp	2	2147479548
gp	3	268468224
tp	4	0
t0	5	0
t1	6	0
t2	7	4
a0	8	0
a1	9	0
a0	10	268501034
a1	11	268500992
a2	12	0
a3	13	9
a4	14	5
a5	15	0
a6	16	0
a7	17	10
s2	18	268501032
s3	19	10
s4	20	268501024
s5	21	268501008
s6	22	0
s7	23	268501028
s8	24	0
s9	25	268501012
s10	26	268501028
s11	27	0
t3	28	9
t4	29	1
t5	30	78
t6	31	8
pc		4194620

**Messages**

```

Array before partitioning:
-1 22 8 35 5 4 11 2 1 78
Array after partitioning:
-1 5 4 2 1 8 11 35 22 78
-- program is finished running (0) --

```

### 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:

C:\Users\iansu\Documents\Class Resources\CENG3420\lab1-3.asm - RARS 1.5

File Edit Run Settings Tools Help

Run speed at max (no interaction)

**Edit Execute**

**Text Segment**

Bkpt	Address	Code	Basic
	0x00400000	0x1480006f jal x0,328	15: j _start #jump to the main code
	0x00400004	0xffc10113 addi x2,x2,-4	18: addi sp, sp, -4
	0x00400008	0x00112023 sw x1,0(x2)	19: sw ra, 0(sp)
	0x0040000c	0x0fc10917 auipc x18,64528	20: la s2, arr
	0x00400010	0xff490913 addi x18,x18,-12	
	0x00400014	0x00000993 addi x19,x0,0	21: li s3, 0
	0x00400018	0x00100893 addi x17,x0,1	23: li a7, 1
	0x0040001c	0x00092503 lw x10,0(x18)	24: lw a0, (s2)
	0x00400020	0x00000073 ecall	25: ecall
	0x00400024	0x00400893 addi x17,x0,4	26: li a7, 4
	0x00400028	0x0fc10517 auipc x10,64528	27: la a0, space
	0x0040002c	0x00250513 addi x10,x10,2	
	0x00400030	0x00000073 ecall	28: ecall
	0x00400034	0x00490913 addi x18,x18,4	29: addi s2, s2, 4
	0x00400038	0x00198993 addi x19,x19,1	30: addi s3, s3, 1
	0x0040003c	0x00168e13 addi x25,x13,1	31: addi t3, a3, 1
	0x00400040	0x000a9a13 slli v6,v19,10	32: slli r1, s3, 10

**Data Segment**

Address	Value (+0)	Value (+4)	Value (+8)	Value (+c)	Value (+10)
0x10010000	-1	1	2	4	
0x10010020	35	78	2097162	1634890305	
0x10010040	171603058	1920090368	1629518177	1919251558	
0x10010060	0	0	0	0	
0x10010080	0	0	0	0	
0x100100a0	0	0	0	0	
0x100100c0	0	0	0	0	
0x100100e0	0	0	0	0	
0x10010100	0	0	0	0	
0x10010120	0	0	0	0	
0x10010140	0	0	0	0	
0x10010160	0	0	0	0	
0x10010180	0	0	0	0	
0x100101a0	0	0	0	0	

0x10010000 (.data) Hexadecimal Addresses

**Registers**

Name	Number	Value
zero	0	0
ra	1	4194704
sp	2	2147479548
gp	3	268468224
tp	4	0
t0	5	0
t1	6	0
t2	7	2
s0	8	0
s1	9	0
a0	10	268501034
a1	11	268500992
a2	12	8
a3	13	7
a4	14	3
a5	15	0
a6	16	0
a7	17	10
s2	18	268501032
s3	19	10
s4	20	268501000
s5	21	268501000
s6	22	0
s7	23	268501004
s8	24	0
s9	25	0
s10	26	9
s11	27	9
t3	28	8
t4	29	2
t5	30	4
t6	31	4
pc		4194712

**Messages** Run I/O

Array before quicksort:  
-1 22 8 35 5 4 11 2 1 78  
Array after quicksort:  
-1 1 2 4 5 8 11 22 35 78  
-- program is finished running (0) --

Clear