**Lab 3 Report**

**ECE 154A**

**Vishal Seenivasan**

1. I spent approximately 4 hours on this lab.
2. I did not modify the C code and attempted to recreate it in Assembly as close as possible.
3. sort.s:

##############################################################################

# File: sort.s

# Skeleton for ECE 154A

##############################################################################

.data

student:

.asciz "Vishal S:\n" # Place your name in the quotations in place of Student

.globl student

nl: .asciz "\n"

.globl nl

sort\_print:

.asciz "[Info] Sorted values\n"

.globl sort\_print

initial\_print:

.asciz "[Info] Initial values\n"

.globl initial\_print

read\_msg:

.asciz "[Info] Reading input data\n"

.globl read\_msg

code\_start\_msg:

.asciz "[Info] Entering your section of code\n"

.globl code\_start\_msg

key: .word 268632064 # Provide the base address of array where input key is stored(Assuming 0x10030000 as base address)

output: .word 268632144 # Provide the base address of array where sorted output will be stored (Assuming 0x10030050 as base address)

numkeys: .word 6 # Provide the number of inputs

maxnumber: .word 10 # Provide the maximum key value

## Specify your input data-set in any order you like. I'll change the data set to verify

data1: .word 1

data2: .word 2

data3: .word 3

data4: .word 5

data5: .word 6

data6: .word 8

.text

.globl main

main: # main has to be a global label

addi sp, sp, -4 # Move the stack pointer

sw ra, 0(sp) # save the return address

li a7, 4 # print\_str (system call 4)

la a0, student # takes the address of string as an argument

ecall

jal process\_arguments

jal read\_data # Read the input data

j ready

process\_arguments:

la t0, key

lw a0, 0(t0)

la t0, output

lw a1, 0(t0)

la t0, numkeys

lw a2, 0(t0)

la t0, maxnumber

lw a3, 0(t0)

jr ra

### This instructions will make sure you read the data correctly

read\_data:

mv t1, a0

li a7, 4

la a0, read\_msg

ecall

mv a0, t1

la t0, data1

lw t4, 0(t0)

sw t4, 0(a0)

la t0, data2

lw t4, 0(t0)

sw t4, 4(a0)

la t0, data3

lw t4, 0(t0)

sw t4, 8(a0)

la t0, data4

lw t4, 0(t0)

sw t4, 12(a0)

la t0, data5

lw t4, 0(t0)

sw t4, 16(a0)

la t0, data6

lw t4, 0(t0)

sw t4, 20(a0)

jr ra

counting\_sort:

#########################

## your code goes here ##

#########################

#Equivalent C code in parantheses in comments

#Create count array on stack

add t0, zero, sp #t0 holds address of count[maxnumber+1] (int count[maxnumber+1)

addi t1, a3, 1 #Maxnumber + 1

slli t1, t1, 2 #Multiply by 4

neg t1, t1 #Make negative

add sp, sp, t1 #Move stack pointer maxnumber+1 back

#Initialize n

add t2, zero, zero #t2 is n (int n)

#Loop 1

for\_1: bgt t2, a3, end\_1 #(for(n = 0; n++; n <= maxnumber))

add t4, t2, zero #Hold n in t4

slli t4, t4, 2 #Multiply by 4

add t3, t0, t4 #t3 holds address of count[n]

sw zero, 0(t3) #(count[n] = 0)

addi t2, t2, 1 #n+1

j for\_1

end\_1:

add t2, zero, zero #Reset n

for\_2: bge t2, a2, end\_2 #(for(n = 0; n++; n < numkeys))

add t4, t2, zero #Hold n in t4

slli t4, t4, 2 #Multiply by 4

add t3, a0, t4 #t3 holds the address of keys[n]

lw t5, 0(t3) #Load keys[n] into t5

slli t5, t5, 2 #Multiply keys[n] by 4 for index value

add t6, t0, t5 #t6 holds the value of count[keys[n]]

lw t4, 0(t6) #Load count[keys[n]] into t4

addi t4, t4, 1 #(count[keys[n]]++)

sw t4, 0(t6) #Store incremented value back in count[keys[n]]

addi t2, t2, 1 #n++

j for\_2

end\_2:

addi t2, zero, 1 #Reset n to 1

for\_3: bgt t2, a3, end\_3 #(for(n = 1; n++; n <= maxnumber))

add t4, t2, zero #Hold n in t4

slli t4, t4, 2 #Multiply by 4 for index value n

add t3, t0, t4 #t3 holds address of count[n]

addi t4, t4, -4 #Decrement n to n-1

add t5, t0, t4 #t5 holds address of count[n-1]

lw t5, 0(t5) #t5 holds count[n-1]

lw t6, 0(t3) #t6 holds count[n]

add t6, t6, t5 #t6 holds count[n]+count[n-1]

sw t6, 0(t3) #(count[n] = count[n]+count[n-1])

addi t2, t2, 1 #n++

j for\_3

end\_3:

add t2, zero, zero #Reset n to 0

for\_4: bge t2, a2, end\_4 #(for(n = 0; n++; n < numkeys))

add t4, t2, zero #Hold n in t4

slli t4, t4, 2 #Multiply by 4 for index value n

add t3, a0, t4 #t3 holds address of keys[n]

lw t3, 0(t3) #t3 holds keys[n]

slli t5, t3, 2 #Multiply t3 by 4 and store in t5 for index value keys[n]

add t6, t0, t5 #t6 holds address of count[keys[n]]

lw t6, 0(t6) #t6 holds count[keys[n]]

addi t6, t6, -1 #t6 holds count[keys[n]]-1

slli t6, t6, 2 #Multiply t6 by 4 for index value count[keys[n]]-1

add t6, a1, t6 #t6 holds address of output[count[keys[n]]-1]

sw t3, 0(t6) #(output[count[keys[n]]-1] = keys[n])

add t5, t0, t5 #t5 hold address of count[keys[n]]

lw t6, 0(t5) #t6 holds count[keys[n]]

addi t6, t6, -1 #(count[keys[n]]--)

sw t6, 0(t5) #Store decremented count[keys[n]]

addi t2, t2, 1 #n++

j for\_4

end\_4:

#Deallocate stack

neg t1, t1 #Flip t1 positive again

add sp, sp, t1 #Move the stack pointer maxnumber+1 forward again

#########################

jr ra

#########################

##################################

#Dont modify code below this line

##################################

ready:

jal initial\_values # print operands to the console

mv t2, a0

li a7, 4

la a0, code\_start\_msg

ecall

mv a0, t2

jal counting\_sort # call counting sort algorithm

jal sorted\_list\_print

# Usual stuff at the end of the main

lw ra, 0(sp) # restore the return address

addi sp, sp, 4

jr ra # return to the main program

print\_results:

add t0, zero, a2 # No of elements in the list

add t1, zero, a0 # Base address of the array

mv t2, a0 # Save a0, which contains base address of the array

loop:

beq t0, zero, end\_print

addi, t0, t0, -1

lw t3, 0(t1)

li a7, 1

mv a0, t3

ecall

li a7, 4

la a0, nl

ecall

addi t1, t1, 4

j loop

end\_print:

mv a0, t2

jr ra

initial\_values:

mv t2, a0

addi sp, sp, -4 # Move the stack pointer

sw ra, 0(sp) # save the return address

li a7, 4

la a0, initial\_print

ecall

mv a0, t2

jal print\_results

lw ra, 0(sp) # restore the return address

addi sp, sp, 4

jr ra

sorted\_list\_print:

mv t2, a0

addi sp, sp, -4 # Move the stack pointer

sw ra, 0(sp) # save the return address

li a7,4

la a0,sort\_print

ecall

mv a0, t2

#swap a0,a1

mv t2, a0

mv a0, a1

mv a1, t2

jal print\_results

#swap back a1,a0

mv t2, a0

mv a0, a1

mv a1, t2

lw ra, 0(sp) # restore the return address

addi sp, sp, 4

jr ra

1. Some more examples of initializing arrays would be useful – there is only one example in the slides of arrays and it does not go over initialization.