Course Name: Computer Architecture Lab

Course Number and Section: **14:332:333:03**

Experiment: [Experiment # 3 – C Memory Management and Introduction to RISC-V]

Lab Instructor: Christos Mitropoulos

Date Performed: October 10 2018

Date Submitted: October 24 2018

Submitted by: Vancha Verma 173004061

Course Name: Computer Architecture Lab

Course Number and Section: **14:332:333:03**

! Important: Please include this page in your report if the submission is a paper submission. For electronic submission (email or Sakai) please omit this page.

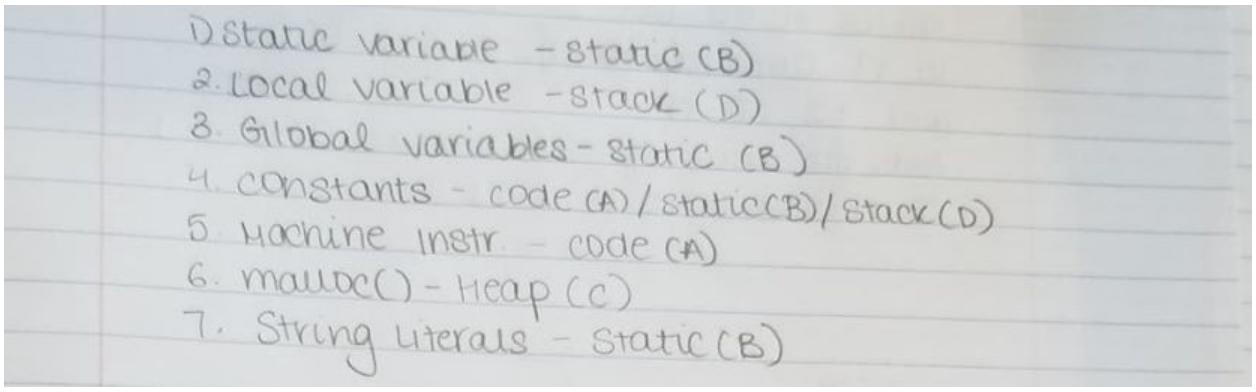
----------For Lab Instructor Use ONLY----------

GRADE: _____

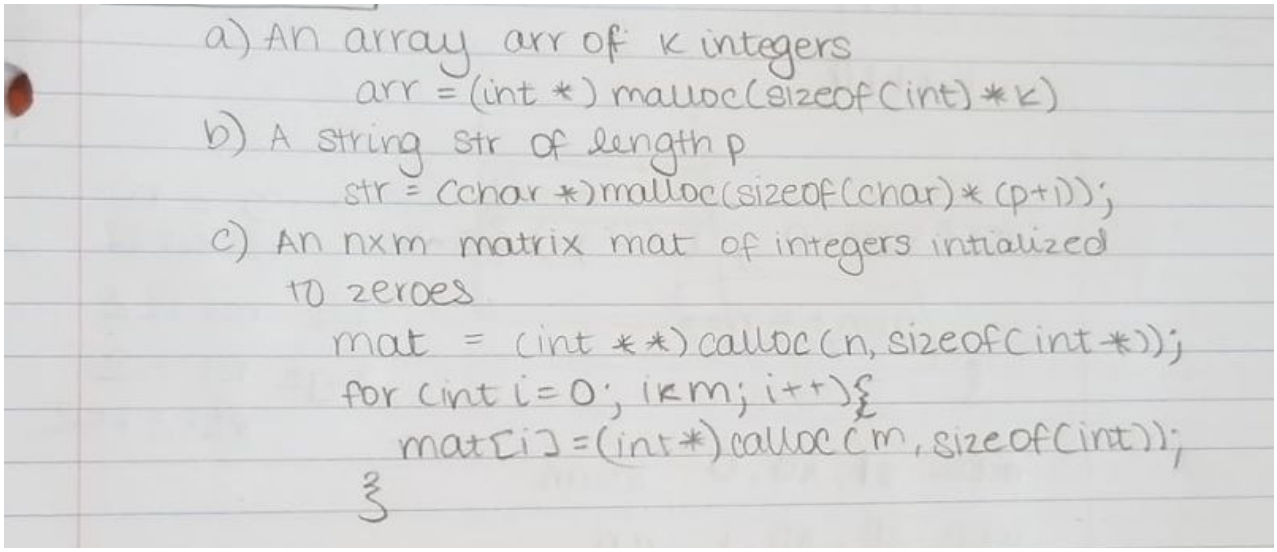
COMMENTS:

Electrical and Computer Engineering Department
School of Engineering
Rutgers University, Piscataway, NJ 08854
ECE Lab Report Structure

Exercise 1: Match the items on the left with the memory segment in which they are stored. Answers may be used more than once, and more than one answer may be required.



Exercise 2: Write the code necessary to properly allocate memory (on the heap) in the following scenarios



Exercise 3: **Assume we have an array in memory that contains int* arr = {1,2,3,4,5,6,0}. Let the value of arr be a multiple of 4 and stored in register s0. What do the snippets of RISC-V code do?**

a) lw t0, 12(s0) #loads the value from register s0, offset 12 into t0. Since the value of arr are multiples of 4 and 12/4 = 3, we would load arr[3] into t0, which is 4.

b) slli t1, t0, 2 #shifts the number from register t0 twice and stores in the register t1
add t2, s0, t1 #adds the values from s0 and t1 and stores them in t2
lw t3, 0(t2) #loads word from regist t2, offset 0 and stores it in t3
addi t3, t3, 1 #adds the number from t3 with the decimal 1 and stores it in t3
sw t3, 0(t2) # stores word from register t2, offset 0 in t3

c) lw t0, 0(s0) #loads word from register s0, offset 0 (arr[0] = 1)into register t0. Thus, t0 = 1
xori t0, t0, 0xFFF #xor the hexadecimal number 0xFFF (1111 1111 1111) with t0 bit by bit in binary and store each bit in t0.
addi t0, t0, 1 #add the decimal value 1 with value of t0 from the previous instruction and save the new value in register t0

Exercise 4: What are the instructions to branch to label on each of the following conditions? The only branch instructions you may use are beq and bne.

s0 < s1	s0 <= s1	s0 > 1
slt v0, s0, s1 bne s0, 0, L1	slt v0, s1, s0 beq s0, 0, L1	sltiu v0, s0, 2 beq v0, 0, L1

Exercise 5: **Open the files lab3_ex5_c.c and lab3_ex5_assembly.s. The assembly code provided (.s file) is a translation of the given C program into RISC-V. Your task is to find/explain the following components of this assembly file.**

a. The register representing the variable k.
The register t0 is represented by ‘k’. To get the offset for each iteration, it is multiplied by 4 and put in a different register at the start of the loop. In the end, t0’s value is increased by 1 to get the new index.

b. The registers acting as pointers to the source and dest arrays.
t1 is a pointer used to point at the source and t2 points to destination. t4 holds the value of t1 with the offset of t3.

c. The assembly code for the loop found in the C code.
The loop is labeled ‘loop’. It starts by checking is the value of t5 is the same as 0. If they are equal, the program jumps to ‘exit’. The loop ends by saying ‘jal x0 loop’, which jumps by to the top of the loop and starts the next iteration.

d. How the pointers are manipulated in the assembly code.
Pointers can be manipulated by changing the value of the offset. For example, t3 is the offset of

the pointer t1. t3 is multiplied by four every loop, which changes where the pointer is; thus, changing the value stored in t4.

Exercise 6: Translate between the C and RISC-V code. You may want to use the RISC-V Reference Card for more information on the instruction set and syntax. In all of the C examples, we show you how the different variables map to registers – you don't have to worry about the stack or any memory-related issues. You may assume all registers are initialized to zero

C	**RISC V**

// s0 -> a, s1 -> b // s2 -> c, s3 -> z
int a = 4, b = 5, c = 6, z;
z = a + b + c + 10;
	addi s0, x0, 4
	addi s1, x0, 5
	addi s2, x0, 6
	add s3, s0, s1
	add s3, s3, s2
	addi s3, s3, 10

// s0 -> int * p = intArr;
// s1 -> a;
*p = 0;
int a = 2;
p[1] = p[a] = a;
	sw x0, 0(s0)
	addi s1, x0, 2
	sw s1, 4(s0)
	slli t0, s1, 2
	add t0, t0, s0
	sw s1, 0(t0)

// s0 -> a, s1 -> b
int a = 5, b = 10;
if(a + a == b) {
a = 0;
} else {
b = a - 1;
}
	addi s0, x0, 5
	addi s1, x0, 10
	add t0, s0, s0
	bne t0, s1, logic
	addi s1, s0, -1
	jal x0, exit
	logic:
	sub s0, s0, s0
	exit:

// a -> s0, b-> s1, c -> t0
int a = 0;
int b = 0;
int c = 30;
 	
while(a != c){
b = b + b;
a = a + 1;
}
	addi s0, x0, 0
	addi s1, x0, 1
	addi t0, x0, 30
	loop:
	beq s0, t0, exit
	add s1, s1, s1
	addi s0, s0, 1
	jal x0, loop
	exit:

// s0 -> n, s1 -> sum //
assume n > 0 to start
int sum;
for(sum=0;n>0;sum+=n--);
	addi s1, x0, 0
	addi s0, x0, 1
	loop:
	bge 0, s0, exit
	add s1, s1, s0
	addi s0, s0, =1
	jal x0, loop
	
	exit:

Exercise 7: Implement a function factorial in RISC-V that has a single integer parameter n and returns n!. A stub of this function can be found in the file factorial.s. You will only need to add instructions under the factorial label, and the argument that is passed into the function is configured to be located at the label n. You may solve this problem using either recursion or iteration.

```
factorial:
      addi x2,x2,-16 #move the pointer
      sw   x0,8(x2)
      sw   x10,0(x2)
      addi x6,x10,-1 #n-1
      bge  x6,x0,Label1 #if n is greater than 1
      addi x10,x0,1
      addi x2,x2,16 #pop
      jalr x0,0(x1) #return val

    Label1:
      addi x10,x10,-1 #n-1
      jal  x1,factorial #factorial of next number
      add  x5,x10,x0 #put factorial in x5
```

```
lw   x10,0(x2) #restore pointer
lw   x1,8(x2)
addi x2,x2,16
mul  x10,x10,x5 #multiply n-1 factorial by n
jalr x0,0(x1) #return
```