Course Name: Computer Architecture Lab

Course Number and Section: **14:332:333:01**

Experiment: Lab 3

Lab Instructor: Ali Essam Hameed Haddad

Date Performed:10/15/18

Date Submitted: 10/29/18

Submitted by: Yu Liu 173001088

Course Name: ________________

Course Number and Section: **14:332:xxx:xx**

! 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:

1. Static Variables = B (Static)

2. Local Variables = D (Stack)

3. Global Variables = B and C (Static and Heap)

4. Constants = B (Static)

5. Machine Instructions = A (Code)

6. malloc() = C (Heap)

7. String Literals = B and D (Static and Stack)

Exercise 2:

1. int *A = (int*) malloc(sizeof (int) * k);

2. char *str = (char*)malloc(p);

3. int **mat = (int**)malloc(sizeof (int*)*n);

```
for(i = 0; i<n; i++){
*(mat+i) = (int*)malloc (sizeof (int)*m);
}
```

Exercise 3.

1. The instruction loads a 32 bit value from memory of the array s0 with an offset of 12 into the register t0. So it sets t0 equal to array arr[3] since 12 divide by 4 is 3.

2. The instructions start out with but shifting t0 2 bits to the left. In other words, multiply the t0 by 2^2 or 4 and t1 becomes the new address of t0. Next t2 is equal to s0 added with t1 and then load that into register t3 with an offset of 0. For addi, the new t3 is equal to t3 plus an increment of 1.Finally store that value in t3 with an offset of 0.

3. s0 is first loaded into t0 with an offset of 0. The XOR is taken place between t0 and 0xFFF (4095 or 1111 1111 1111 in binary). After the operation the "new" t0 becomes 1111 1111 1110 and the next instruction requires t0 = t0 +1. So 1111 1111 1110 + increment of 1 will be 1111 1111 1111 or 4095 again.

Exercise 4

s0 < s1	s0 <= s1	s0 > s1
slt t0, s0, s1 Bne t0, 0, label	slt t0, s1, s0 Beq t0,0, label	1. sltiu t0, s0, 2 Beq t0, 0, label

Exercise 5

1. t0 is the register representing the variable k. After each loop, t0 will be incremented by 1 to find the offset of t1 that points to the original source array.

2. t1 points to the source array and t2 points to the destination array as stated by:
    a. la t1, source
       la t2, dest

3. The code is between 'loop:' and 'exit:'. It is equivalent to the for loop in C.
    a. slli t3, t0, 2
       add t4, t1, t3
       lw t5, 0(t4)
       beq t5, x0, exit
       add t6, t2, t3

```
sw t5, 0(t6)
addi t0, t0, 1
jal x0, loop
```

4. Pointers are manipulated in assembly language by accessing is values in memory's from the register's memory address. The assembly language will point to a specific array value by creating offsets of that array, then load and store them into a new register.

Exercise 6

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
	addi s3, x0, 10
	addi s3, s0, 0
	addi s3, s1, 0
	addi s3, s2, 0

// 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,
	Else:
	xor s0, x0, x0
	jal x0, exit

// s1 = 2^30
s1 = 1;
for(s0=0;s0<30;s++) {
	addi s0, x0, 0
	addi s1, x0, 1
	 addi t0, x0, 30
	loop:
	      beq s0, t0, exit

s1 *= 2;
}
	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, s1, 0
	loop:
	beq s0, x0, exit
	add s1, s1, s0
	add s0, s0, -1
	jal x0, loop
	exit:

Exercise 7

```
 factorial:
 addi a1, a0, 0 // loads argument from a0 with an increment of 0 and then it is stored in a1
 beq a1, x0, else // if input argument is 1, go to else
 subi a2, a1, 1 // decrements a1 by 1, stores the value in a2
 addi a0, a2, 0 // loads a2 ina0
 jal ra, factorial // recurse
 mul a0, a1, a2 // multiplies a1 by a2 and stores it in a0
 return
else:
 addi a0, x0, 1 // a0=1
 return
```