Course Name: Comp Arch Lab 1

Course Number and Section: 14:332:333:02

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

Lab Instructor: Jalal Abdulbaqi

Date Performed: 10/10/2018

Date Submitted: 10/23/2018

Submitted by: [Jahidul Islam - 171001155]

Course Name: Comp Arch Lab 1

Course Number and Section: 14:332:333:02

! 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

1. Purpose / Introduction / Overview – describe the problem and provide background information
2. Approach / Method – the approach took, how problems were solved
3. Results – present your data and analysis, experimental results, etc.
4. Conclusion / Summary – what was done and how it was done

.

Introduction/Approach:

In this experiment, we were properly introduced with RISC-V's machine language. Comparisons between c language and the machine language were practiced with exercises displaying the truly lower level of language that machine language resides in. Code from the programmer is broken down into much simpler terms for the machine to process. Thus, each individual action taken for granted in c, must be explicitly written in machine language.

1) Given choices: A-Code B-Static C-Heap D-Stack

a) Static Variables **B**
b) Local Variables **D**
c) Global Variables **B**
d) Constants **A,B,D**
e) Machine Instructions **A**
f) malloc() **C**
g) String Literals **B**

2)

a) arr = (int *) malloc(sizeof(int) * k);
b) str = (char *) malloc(sizeof(char) * (p + 1));
c) mat = (int **) calloc(n, sizeof(int *));
```
for (int i = 0; i < m; i++) {
mat[i] = (int *) calloc(m, sizeof(int));
}
```

3)

a) The first snippet sets register t0 equal to arr[3]
b) The next part increments the array element specified by t2, written as arr[t2], by 1
c) The last snipper sets the register t0 to the two's complement negation version of arr[0]

4)

s0<s1	s0<=s1	s0>1
slt t0, s0, s1 bne t0, 0, label	slt t0, s1, s0 beq t0, 0, label	sltiu t0, s0, 2 beq t0, 0, label

5)

a. The register representing the variable k.

Variable k: t0

b. The registers acting as pointers to the source and dest arrays.
Source Pointer: t1
Destination: t2

c. The assembly code for the loop found in the C code.

```
loop:
          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
exit:
```

d. How the pointers are manipulated in the assembly code.

In this assembly code, pointers are used to get the value of array index. We can see that t1 is the pointer for array source. Thus in order to access source[k], what we need to do is k = index * 4. This is because the integer size is 4 thus the necessity to multiply by 4. After that, we now have t1 + k which will give us the value of a[k]. From the given assembly code, we can observe that pointers store the base address of array. If we were to then add an index to the base address we would attain the value of a[k].

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
	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	addi s0, x0, 5
int a = 5, b = 10;	addi s1, x0, 10
if(a + a == b) {	add t0, s0, s0
a = 0;	bne t0, s1, else
} else {	xor s0, x0, x0
b = a - 1;	jal x0, exit
}	else:
	addi s1, s0, -1
	exit:
// This code computes s1 = 2^30	addi s0, x0, 0
s1 = 1;	addi s1, x0, 1
for(s0=0;s0<30;s++) {	addi t0, x0, 30
s1 *= 2;	loop:
}	beq s0, t0, exit
	add s1, s1, s1
	addi s0, s0, 1
	jal x0, loop
	exit:
// s0 -> n, s1 -> sum	addi s1, s1, 0
// assume n > 0 to start	loop:
int sum;	beq s0, x0, exit
for(sum=0;n>0;sum+=n--);	add s1, s1, s0
	add s0, s0, -1
	jal x0, loop
	exit:

7)

```
.data
n: .word 8

.text
main:
  la t0, n
  lw a0, 0(t0)
  jal ra, factorial

  addi a1, a0, 0
  addi a0, x0, 1
  ecall # Print Result

  addi a0, x0, 10
```

```
    ecall # Exit

factorial:
            # YOUR CODE HERE
    li a0, 15
    jalr x0,0(x1)
```

Conclusion:

Concepts of memory and machine language were introduced and practiced throughout each exercise. Overall, machine language is very logical with its approach of requiring each step of the program to be written explicitly. In following the procedure for practices given at the the beginning of the lab, success was attained in properly answering each exercise question.