Course Name: Computer Architecture and Assembly Laboratory

Course Number and Section: 14:332:331:01

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

Lab Instructor: Mengmei Ye

Date Performed: 10/1218

Date Submitted: 10/12/18

Submitted by: [Aryeh Ness, RUID: 173000855]

Course Name: Computer Architecture and Assembly Laboratory

Course Number and Section: 14:332:331:04

! 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:

Exercise 1:

1. Static variables=static, since these variables have fixed size.
2. Local variables=stack, since stack is defined as storing “Local variables and arguments inside functions.”
3. Global variables=static, since these variables have fixed size but can also be modified with the execution of the program .
4. Constants=code, static and stack. With DEFINE, you can replace something in the code. With constant declaration, it’s stored in the static or stack depending on being declared in a function or not.
5. Machine Instructions=code, since these are set instructions, written in code, to execute the program-they load at the start and do not change.
6. malloc()=heap, since heap is the space for dynamic data, and malloc is a function that changes the availability of space in the program. Malloc enters the heap to free up and allocate data.
7. String Literals=static, since these have a fixed, unchanging size.

Exercise 2:

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

Exercise 3:

a) Sets register t0 equal to arr[3].
b) Increments the array element specified by t2 (i.e. arr[t2]) by 1
c) Sets the register t0 to the two’s complement negation of arr[0]

Exercise 4:

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

Exercise 5:

a) The register containing the variable k is t0.
b) The registers acting as pointers to the source and dest arrays are t1 and t2, respectively.
c) The assembly code for the loop found in the C code is the following:

```
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
```

d) The pointers are manipulated in the assembly code in the following ways: the source pointer, t1, is used to create the value in the register t4 that is used to check the conditional of the loop. It is added to register t3, the register that is the value of variable k, with two added zeroes at the end. When the loop runs, this will be used to equate source and dest. The destination pointer, t2, is used in practically the same way: its value is added into register t6 with register t3, which points back all the way to variable k as previously discussed.

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
	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, else
	xor s0, x0, x0
	jal x0, exit
	else:
	addi s1, s0, -1
	exit:

// computes s1 = 2^30
s1 = 1;
for(s0=0;s0<30;s++) {
  s1 *= 2;
}
	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, s1, 0
	loop:
	beq s0, x0, exit
	add s1, s1, s0
	add s0, s0, -1
	jal x0, loop
	exit:

Exercise 7: The factorial value is stored in a3 and a5. A0 is the original factorial:

```
        lw a2, 0(t0)
  addi a6, x0, 1
  add a3, a2, x0
  loop:
              sub a2, a2, a6
    add a5, a3, x0
    beq a2, x0, exit
    add a3, x0, x0
    add a4, a2, x0
    jal x0, multiply
      exit:
  jal x0, done
      multiply:
      beq a4, x0, here
      add a3, a3, a5
      sub a4, a4, a6
      jal x0, multiply
    here:
    jal, x0, loop
    done:
```