RUTGERS
UNIVERSITY

Course Name: Computer Architecture and Assembly Lab
Course Number and Section: 14:332:333:02A

Experiment: Experiment 4: C Memory Management and Introduction to RISC-V

Lab Instructor: Jalal Abdulbaqi

Date Performed: 10/24/18

Date Submitted: 11/7/18

Submitted by: Vedanta Dhobley – 171002654

1. RISC-V functions

1.1. Write a function triple in RISC-V that, when given integer x, returns 3x

```
main:
addi x11, x0, 3 #x stored in register x11 and assigned value of 3
jal x1, triple
addi x10, x0, 1 #x stored in register x10 and assigned value 1
ecall
addi x10, x0, 10 #x stored in register x10 and assigned value 10
ecall
triple:
add x5, x11, x11 #register x5 added to register x11 and assigned to register x11
add x11, x11, x5 #register x11 added to register x11 and assigned to register x5
jalr x0, 0(x1) #return value of 3x
```

otherwise:

```
double: add a0, a0, a0
jr ra
```

1.2. Write a function power in RISC-V that takes in two numbers x and n and returns x^n. You may assume that n>=0 and that multiplication will always result in a 32-bit.

```
main:
  addi x11, x0, 5 #x stored in x11 and assigned a value of 5
  addi x10, x0, 3 #n stored in x10 and assigned a value of 3
  jal x1, Power
  addi x10, x0, 1
  ecall
  addi x10, x0, 10
  ecall
Power:
  addi x5, x0, 0
  addi x6, x0, 1
loop:
  beq x5, x10, exit
  mul x6, x6, x11
  addi x5, x5, 1
  jal x0, loop
exit:
  addi x11, x6, 0
  jalr x0, 0(x1)
```

2. RISC-V Arrays and List: Comment each snippet with what the snippet does. Assume that there is an array, $int\ arr[6] = \{3, 1, 4, 1, 5, 9\}$, which is starts at memory address 0xBFFFFF00, and a linked list struct (as defined below), $struct\ ll *\ lst;$, whose first element is located at address 0xABCD0000. s0 then contains arr's address, 0xBFFFFF00, and s1 contains lst's address, 0xABCD0000. You may assume integers and pointers are 4 bytes and that structs are tightly packed.

```
struct ll {
   int val;
struct ll * next;
}
```

2.1.

```
lw t0, 0(s0) // loads the value of arr[0] into t0
lw t1, 8(s0) // loads the value of arr[2] into t1
add t2, t0, t1 // adds the values of arr[0] + arr[2] and stores into t1
sw t2, 4(s0) // stores the sum of the addition in arr[1]
```

2.2.

```
add t0, x0, x0 //stores 0 in the register t0
loop:
   slti t1, t0, 6 //checks if register xt0 less than 6; if so xt1 = 1; else xt1 = 0
   beq t1, x0, end //checks if register xt1 equal to 0; if so Label = "end"; else skips
   slli t2, t0, 2 //shit register xt0 left 2 bits; stores value in register xt2
   add t3, s0, t2 //adds register xs0 and register xt2; stores in register xt3
   lw t4, 0(t3) //loads values at 0th offset of address register xt3 in register xt4
   sub t4, x0, t4 //multiplies register xt4 by −1
   sw t4, 0(t3) //stores register xt4 into 0th offset of address in register xt3
   addi t0, t0, 1 //increments register xt0 by 1 per iteration
   jal x0, loop //returns to beginning of loop
end:
```

2.3.

```
loop:
   beq s1, x0, end //checks node address = 0; if so label prints "end"; else skips
   lw t0, 0(s1) //loads contents at node address into register xt0
   addi t0, t0, 1 //increments node by 1 per iteration
   sw t0, 0(s1) //stores address into register xs1
   lw s1, 4(s1) //loads next node address into register xs1
   jal x0, loop //returns back to "loop"
end:
```

3. RISC-V Calling Conventions

3.1. How do we pass arguments into functions?

We pass arguments into functions in RISC-V by using the 8 function argument registers (a0-a7) and call their definitions. When passing an argument by reference, one passes a pointer to the value in memory

3.2. How are values returned by functions?

A value is returned after it is labeled a specific type and then called during the program. Values can be handles, integers, objects, etc. With value-returning functions, the return value can be used in an expression with value registers a0 and a1.

3.3. What is sp and how should it be used in the context of RISC-V functions?

sp stands for stack pointer. We subtract the stack pointer in order to free up space. The stack is used to restore the value of registers that have been overwritten during execution.

3.4. Which values need to be saved before using jal?

Registers a0 to a7, t0 to t6, and ra must be saved before jal.

3.5. Which values need to be restored before using jr to return from a function?

Registers sp, gp, gp, and s0 to s11.

4. Writing RISC-V Functions

Write a function sumSquare in RISC-V that, when given an integer n, returns the summation below. If n is not positive, then the function returns 0.

$$n^2 + (n-1)^2 + (n-2)^2 + (n-3)^2 + \cdots + 1^2$$

For this problem, you are given a RISC-V function called square that takes in an integer and returns its square. Implement sumSquare using square as a subroutine.

```
sumSquare: addi sp, sp − 12 #space for 3 words on stack
sw ra, 0(sp) #store return address
sw s0, 4(sp) #store register xs0
sw s1, 8(sp) #store register xs1
add s0, a0, x0 #set register xs0 to parameter n
add s1, x0, x0 #set register xs1 to 0
loop: bge x0, s0, end #if register xs0 is not positive
add a0, s0, x0 #set register xa0 to register xs0 prior to square function
jal ra, square #call square function
add s1, s1, a0 #add returned value to register xs1
addi s0, s0, −1 #decrement register xs0 by 1 per iteration.
jal x0, loop #return to loop
end: add a0, s1, x0 #set register xa0 to register xs1
lw ra, 0(sp) #restore ra value
lw s0, 4(sp) #restore register xs0 value
lw s1, 8(sp) #restore register xs1 value
addi sp, sp, 12 #space on stack for 3 words
jr ra #return to caller
```