

San Jose State University
Department of Computer Engineering

CMPE 140 Lab Report

Lab 4 Report

Title MIPS Instruction Set Architecture & Programming (3)

Semester Fall

Date 03/03/2020

by



Name Thien N Hoang

SID 012555673

Name Phat Le



SID 012067666

Lab Checkup Record

Week	Performed By (signature)	Checked By (signature)	Tasks Successfully Completed*	Tasks Partially Completed*	Tasks Failed or Not Performed*
1	TH 		100%		

* Detailed descriptions must be given in the report.

Authors

 A portrait of a young man with short, dark hair, wearing a dark blue t-shirt. He is looking directly at the camera with a neutral expression.	<p style="text-align: center;">Thien Hoang</p> <p>I was born and raised in Ho Chi Minh City, Viet Name. I came to the United States when I was 16 years old. Before transferring to SJSU, I lived in Houston Texas with my family. I am currently a 5th-year Computer Engineering major at San Jose State University. I have an interest in building the circuit and aspire to be a circuit designer in the future. I enjoy riding a motorcycle on the weekend. It helps me release my stress</p>
 A portrait of a young man with dark hair, wearing a light blue button-down shirt. He is smiling and looking slightly to the right of the camera.	<p style="text-align: center;">Phat Le</p> <p>I am an international student from Vietnam, also a transfer student from Seattle, and currently a senior Computer Engineering student at San Jose State University. My primary goal is to invent a device that will have huge improvements in humanity. By understanding the way computer software and hardware works, I believe my dream can become true one day. I also enjoy playing RPG console video games and singing alone with my guitar. Sounds very introvert right?!</p>

Introduction

The purpose of the lab is to get familiar with the MIPS implementation of stacks, arrays, function procedures, and recursive function procedures. Specifically, given an array of 50 elements, the primary task is to access the value of elements in the array, perform arithmetic calculation, and use the result as the input parameter for the given MIPS assembly factorial function.

Design methodology

The first task is to find the value from the index 25 and 30 of the array of 50 elements. The assembly code for the array is already provided from the lab instruction in which the base address is 0x100. Furthermore, every index from the array has already been given a value that is equal to the index times three ($i*3$). The primary goal is to continue the MIPS assembly program that combines values at index 25 and 30 and then divided by 30. In order to find the exact address of the array index from memory, the assembly needs to multiply the index number by 4 and add it with the base address of the array ($i*4+0x100$). As a result, the value at the index can be loaded to register using the address found from the formula above.

```
void main()
{
    int n, f;
    int my_array[50]; // Create the array
    for(i=0; i<50; i=i+1){
        my_array[i] = i*3;
    }
    /*You will write MIPS code for the following parts*/
    // Arithmetic calculation
    n = (my_array[25]+ my_array[30])/30;
    // Factorial
    f = Factorial(n);
    return;
}

// Recursive factorial procedure
int Factorial(int n)
{
    if (n <= 1) return 1;
    else return
        (n*Factorial(n-1));
}
```

Figure 1: C++ pseudo code of the array initialization and the Factorial functional implementation given from the lab's instruction.

The second task is to observe the execution by single-step through all instructions and then record the register changes to the test log as shown in *Table 3* below. In addition, another requirement is to sketch the stack diagram which would include the stack pointer position, the addresses, and the data value stored at the addresses of the stack.

Simulation Result

The simulation was successful since the value loaded to registers was as expected. Specifically, values loaded to register \$t4 and \$t5 from index 25 and index 30 were respectively equal to 75 and 90 which matches the equation $\text{index} \times 3$. The result of the arithmetic calculation $((a[25] + a[30]) / 30)$ was equal to 5 which also matched the value at register \$a1. Therefore, the input argument for the factorial function is also 5.

According to *Table 1*, the stack pointer(\$sp) always begins at the top of the stack (2ffc). As every time the recursive function is called, the stack would allocate more space by decreasing the value of the top's stack address to store the return address as well as the value of the parameter. After the recursive function reached its base condition ($n \leq 1$), the stack pointer began to increase and restore the value of the return address and the parameter from the stack memory until the function returned to the next instruction of main function.

Table 1: The stack changes when calling the recursive factorial function.

Storing to the stack

Address	Data	\$sp position
2ffc		<-\$sp
Address	Data	\$sp position
2ffc		
2ff8	\$a1(5)	
2ff4	\$ra(306c)	
		<-\$sp
Address	Data	\$sp position
2ffc		
2ff8	\$a1(5)	
2ff4	\$ra(306c)	
2ff0	\$a1(4)	
2fec	\$ra(30a0)	<-\$sp
Address	Data	\$sp position
2ffc		
2ff8	\$a1(5)	
2ff4	\$ra(306c)	
2ff0	\$a1(4)	
2fec	\$ra(30a0)	
2fe8	\$a1(3)	
2fe4	\$ra(30a0)	
		<-\$sp
Address	Data	\$sp position
2ffc		
2ff8	\$a1(5)	
2ff4	\$ra(306c)	
2ff0	\$a1(4)	
2fec	\$ra(30a0)	
2fe8	\$a1(3)	
2fe4	\$ra(30a0)	
2fe0	\$a1(2)	
2fdc	\$ra(30a0)	
		<-\$sp
Address	Data	\$sp position
2ffc		
2ff8	\$a1(5)	
2ff4	\$ra(306c)	
2ff0	\$a1(4)	
2fec	\$ra(30a0)	
2fe8	\$a1(3)	
2fe4	\$ra(30a0)	
2fe0	\$a1(2)	
2fdc	\$ra(30a0)	
2fd8	\$a1(1)	
2fd4	\$ra(30a0)	<-\$sp

Table 2: The stack changes when the recursive factorial function reaches its base condition.

Restoring value from stack		
Address	Data	\$sp position
2ffc		<-\$sp
2ff8	\$a1(5)	
2ff4	\$ra(306c)	
2ff0	\$a1(4)	
2fec	\$ra(30a0)	
2fe8	\$a1(3)	
2fe4	\$ra(30a0)	
2fe0	\$a1(2)	
2fdc	\$ra(30a0)	
2fd8	\$a1(1)	
2fd4	\$ra(30a0)	

Address	Data	\$sp position
2ffc		<-\$sp
2ff8	\$a1(5)	
2ff4	\$ra(306c)	
2ff0	\$a1(4)	
2fec	\$ra(30a0)	
2fe8	\$a1(3)	
2fe4	\$ra(30a0)	
2fe0	\$a1(2)	
2fdc	\$ra(30a0)	
2fd8	\$a1(1)	
2fd4	\$ra(30a0)	

Address	Data	\$sp position
2ffc		<-\$sp
2ff8	\$a1(5)	
2ff4	\$ra(306c)	
2ff0	\$a1(4)	
2fec	\$ra(30a0)	
2fe8	\$a1(3)	
2fe4	\$ra(30a0)	
2fe0	\$a1(2)	
2fdc	\$ra(30a0)	
2fd8	\$a1(1)	
2fd4	\$ra(30a0)	

Address	Data	\$sp position
2ffc		<-\$sp
2ff8	\$a1(5)	
2ff4	\$ra(306c)	
2ff0	\$a1(4)	
2fec	\$ra(30a0)	
2fe8	\$a1(3)	
2fe4	\$ra(30a0)	
2fe0	\$a1(2)	
2fdc	\$ra(30a0)	
2fd8	\$a1(1)	
2fd4	\$ra(30a0)	

Address	Data	\$sp position
2ffc		<-\$sp
2ff8	\$a1(5)	
2ff4	\$ra(306c)	
2ff0	\$a1(4)	
2fec	\$ra(30a0)	
2fe8	\$a1(3)	
2fe4	\$ra(30a0)	
2fe0	\$a1(2)	
2fdc	\$ra(30a0)	
2fd8	\$a1(1)	
2fd4	\$ra(30a0)	

Conclusion

To sum up, all the tasks for this lab have been successfully performed. Furthermore, the lab provides important concepts such as array, stacks, recursive procedures, and procedures using the MIPS instruction. Sketching the stack diagram also gives us better understanding about the state of the stack memory when performing function calling. Overall, we have not faced any challenging problems in this lab.

Successful Tasks

1. We were able to record the data on the test log table.
2. We were able to sketch of the stack status diagram
3. We were able to implement the recursive factorial with the MIPS instruction
4. We were able to get the correct stack after every recursive call

Appendix

Source code

Lab4.asm

```

Main:    addi $a0, $0, 0x100    # array base address = 0x100
        addi $a1, $0, 0        # i = 0
        addi $t0, $0, 3
        addi $t1, $0, 50       # $t1 = 50
CreateArray_Loop:
        slt $t2, $a1, $t1      # i < 50?
        beq $t2, $0, Exit_Loop # if not then exit loop
        sll $t2, $a1, 2        # $t2 = i * 4 (byte offset)
        add $t2, $t2, $a0       # address of array[i]
        mult $a1, $t0          # i * 3
        mflo $t3               # $t3 = i * 3
        sw $t3, 0($t2)         # save array[i]
        addi $a1, $a1, 1       # i = i + 1
        j CreateArray_Loop
Exit_Loop:
        # your code goes in here...
        # arithmetic calculation
        # ...
        addi $a2, $0, 25       # index 25
        sll $t2, $a2, 2        # 25*4
        add $t2, $t2, $a0       # address of array[25]
        lw $t4, 0($t2)         # load array[25] to $t4          #all of this can be written as lw $t4, 100($a0)

        addi $a2, $0, 30       # index 30
        sll $t2, $a2, 2        # 30*4offset
        add $t2, $t2, $a0       # address of array[30]
        lw $t5, 0($t2)         # load array[30] to $t4          #all of this can be written as lw $t4, 120($a0)

        add $a1, $t4, $t5       # n = array[25] + array[30]
        addi $a3, $0, 30
        divu $a1, $a3          # n/30
        mflo $a1               # n = n/30
        sw $a1, 0($0)          # store n to 0x00

        # factorial computation
        jal factorial          # call procedure
        add $s0, $v0, $0       # return value f = factorial(n)
        j done
factorial:
        addi $sp, $sp, -8      # make room on the stack
        sw $a1, 4($sp)         # store input n($a1)
        sw $ra, 0($sp)         # store $ra

        # your code goes in here
        addi $t0, $0, 2        # $to = 2
        slt $t0, $a1, $t0      # n <=1?
        beq $t0, $0, else      # false: go to else
        addi $v0, $0, 1        # yes: return 1
        addi $sp, $sp, 8       # restore $sp
        jr $ra                 # return
else:    addi $a1, $a1, -1      # n = n -1
        jal factorial          # recursive call
        lw $ra, 0($sp)         # restore return address
        lw $a1, 4($sp)         # restore input
        addi $sp, $sp, 8       # restore $sp
        mul $v0, $a1, $v0      # n* factorial(n-1)
        jr $ra                 # return
done:    sw $s0, 16($0)        # store f = n! in 0x10

```

Table 3: Test log from observing the register changes throughout every MIPS instruction

Add r	MIPS Instruction	Machine Code	Registers				Memory Content	
			\$a1	\$sp	\$ra	\$v0	[0x00]	[0x10]
3034	Exit_Loop:addi \$a2, \$0, 25	20060019	0x0000032	0x00002ffc	0x00000000	0x00000000	0	0
3038	sll \$t2, \$a2, 2	00065080	0x0000032	0x00002ffc	0x00000000	0x00000000	0x00000090	0
303c	add \$t2, \$t2, \$a0	01445020	0x0000032	0x00002ffc	0x00000000	0x00000000	0x00000090	0
3040	lw \$t4, 0(\$t2)	8d4c0000	0x0000032	0x00002ffc	0x00000000	0x00000000	0x00000090	0
3044	addi \$a2, \$0, 30	2006001e	0x0000032	0x00002ffc	0x00000000	0x00000000	0x00000090	0
3048	sll \$t2, \$a2, 2	00065080	0x0000032	0x00002ffc	0x00000000	0x00000000	0x00000090	0
304c	add \$t2, \$t2, \$a0	01445020	0x0000032	0x00002ffc	0x00000000	0x00000000	0x00000090	0
3050	lw \$t5, 0(\$t2)	8d4d0000	0x0000032	0x00002ffc	0x00000000	0x00000000	0x00000090	0
3054	add \$a1, \$t4, \$t5	018d2820	0x0000032	0x00002ffc	0x00000000	0x00000000	0x00000090	0
3058	addi \$a3, \$0, 30	2007001e	0x00000a5	0x00002ffc	0x00000000	0x00000000	0x00000090	0
305c	divu \$a1, \$a3	00a7001b	0x00000a5	0x00002ffc	0x00000000	0x00000000	0x00000090	0
3060	mflo \$a1	00002812	0x00000a5	0x00002ffc	0x00000000	0x00000000	0x00000090	0
3064	sw \$a1, 0(\$0)	ac050000	0x0000005	0x00002ffc	0x00000000	0x00000000	0x00000090	0
3068	jal factorial	0c000c1d	0x0000005	0x00002ffc	0x00000000	0x00000000	0x00000005	0
306c	add \$s0, \$v0, \$0	00408020	0x0000005	0x00002ffc	0x0000306c	0x00000078	0x00000005	0x00000000
3070	j done	08000c2d	0x0000005	0x00002ffc	0x0000306c	0x00000078	0x00000005	0x00000000
3074	factorial:addi \$sp, \$sp, -8	23bdfff8	0x0000001	0x00002fdc	0x000030a0	0x00000000	0x00000000	0x00000000
3078	sw \$a1, 4(\$sp)	afa50004	0x0000001	0x00002fd4	0x000030a0	0x00000000	0x00000000	0x00000000
307c	sw \$ra, 0(\$sp)	afb00000	0x0000001	0x00002fd4	0x000030a0	0x00000000	0x00000000	0x00000000

3080	addi \$t0, \$0, 2	20080002	0x00000001	0x00002fd4	0x000030a0	0x00000000	0x00000005	0x00000000
3084	slt \$t0, \$a1, \$t0	00a84020	0x00000001	0x00002fd4	0x000030a0	0x00000000	0x00000005	0x00000000
3088	beq \$t0, \$0, els	11000003	0x00000001	0x00002fd4	0x000030a0	0x00000000	0x00000005	0x00000000
308c	addi \$v0, \$0, 1	20020001	0x00000001	0x00002fd4	0x000030a0	0x00000000	0x00000005	0x00000000
3090	addi \$sp, \$sp, 8	23bd0008	0x00000001	0x00002fd4	0x000030a0	0x00000001	0x00000005	0x00000000
3094	jr \$ra	03e00008	0x00000001	0x00002fdc	0x000030a0	0x00000001	0x00000005	0x00000000
3098	else:addi \$a1, \$a1, -1	20a5ffff	0x00000002	0x00002fdc	0x000030a0	0x00000000	0x00000002	0x00000004
309c	jal factorial	0c000c1d	0x00000001	0x00002fdc	0x000030a0	0x00000000	0x00000002	0x00000004
30a0	lw \$ra, 0(\$sp)	8fbf0000	0x00000004	0x00002ff4	0x000030a0	0x00000018	0x00000005	0x00000000
30a4	lw \$a1, 4(\$sp)	8fa50004	0x00000004	0x00002ff4	0x0000306c	0x00000018	0x00000005	0x00000000
30a8	addi \$sp, \$sp, 8	23bd0008	0x00000005	0x00002ff4	0x0000306c	0x00000018	0x00000005	0x00000000
30ac	mul \$v0, \$a1, \$v0	70a21002	0x00000005	0x00002ffc	0x0000306c	0x00000018	0x00000005	0x00000000
30b0	jr \$ra	03e00008	0x00000005	0x00002ffc	0x0000306c	0x00000078	0x00000005	0x00000000
30b4	done: sw \$s0, 16(\$0)	ac100010	0x00000005	0x00002ffc	0x0000306c	0x00000078	0x00000005	0x00000078