San Jose State University
Department of Computer Engineering

CMPE 200 Lab Report

Assignment 4 Report

Title MIPS Instruction Set Architecture & Programming (3)

Semester Fall 2022 **Date** 10/02/2022

by

Name HARISH MAREPALLI *(typed)* **SID** 016707314 *(typed)*

ABOUT THE AUTHOR

Harish Marepalli had done bachelor's in Electronics and Communication Engineering at Gokaraju Rangaraju Institute of Engineering and Technology, Hyderabad, Telangana, India. He has 3 years of experience as a Systems Engineer in Tata Consultancy Services and his role was software developer. Currently, he is pursuing his master's in Computer Engineering at San Jose State University. His native place is Hyderabad and mother tongue is Telugu. Additionally, he does speak English and Hindi and would love to connect with the people of any region and build network.

LIST OF CONTENTS

1. INTRODUCTION

This activity is used to write a MIPS assembly code to build a 50-entry array with the base address 0x100. The array needs to be accessed to perform some arithmetic calculations, and the result of the calculation will be used as the input argument to the MIPS assembly program for the factorial function.

2. MY GROUP

Name: Student_Team 6
Harish Marepalli – 016707314
Tirumala Saiteja Goruganthu – 016707210

3. GIVEN TASK

a. The task is to take the given assembly code and fill the missing code.
b. The function of the program is to build a 50-entry array with the base address 0x100.
c. To perform the required arithmetic calculations, an array needs to be accessed.
d. Compute the factorial of a number using a recursive procedure.
e. The result of the calculation will be used as the input argument to the MIPS assembly program for the factorial function.

4. STEPS TAKEN TO COMPLETE THE TASK

a. Understood the part of the given that is already provided.
b. Imported the given lab4.asm file into the MARS software.
c. Completed the actual missing part of the code under the lines "#your code goes in here."
d. Assigned the values to the registers as specified.
    a. $a1 <- n
    b. $a0 <- array base address
    c. $s0 <- n!
e. Implemented the factorial function as a recursive procedure.
f. Written the final value of 'n' obtained from the arithmetic calculation to the memory location at address 0x00.
g. The factorial of n is written to the memory location at address 0x10.
h. Assembled the code and checked for any errors.
i. Executed the code step by step.
j. Observed, Understood, and Recorded the values in the given log table.

5. TEST LOG

CMPE 200 Assignment 4 Test Log

Programmer's Names: HARISH MAREPALLI **Date: 10/01/2022**

Record the observed contents of registers and data memory after each instruction is executed.

Addr	MIPS Instruction	Machine Code	Registers				Memory Content	
			$a1	$sp	$ra	$v0	[0x00]	[0x10]
3034	lw $t0, 100($a0)	0x8c880064	0x00000032	0x00002ffc	0x00000000	0x00000000	0x00000000	0x00000000
3038	lw $t1, 120($a0)	0x8c890078	0x00000032	0x00002ffc	0x00000000	0x00000000	0x00000000	0x00000000
303c	add $t1, $t0, $t1	0x01094820	0x00000032	0x00002ffc	0x00000000	0x00000000	0x00000000	0x00000000
3040	addi $t0, $0, 30	0x2008001e	0x00000032	0x00002ffc	0x00000000	0x00000000	0x00000000	0x00000000
3044	divu $t1, $t0	0x0128001b	0x00000032	0x00002ffc	0x00000000	0x00000000	0x00000000	0x00000000
3048	mflo $a1	0x00002812	0x00000005	0x00002ffc	0x00000000	0x00000000	0x00000000	0x00000000
304c	sw $a1, 0($0)	0xac050000	0x00000005	0x00002ffc	0x00000000	0x00000000	0x00000005	0x00000000
3050	jal factorial	0x0c000c19	0x00000005	0x00002ffc	0x00003054	0x00000000	0x00000005	0x00000000
3054	add $s0, $v0, $0	0x00408020	0x00000005	0x00002ffc	0x00003054	0x00000078	0x00000005	0x00000000
3058	sw $s0, 16($0)	0xac100010	0x00000005	0x00002ffc	0x00003054	0x00000078	0x00000005	0x00000078
305c	li $v0, 10	0x2402000a	0x00000005	0x00002ffc	0x00003054	0x0000000a	0x00000005	0x00000000
3060	syscall	0x0000000c	0x00000005	0x00002ffc	0x00003054	0x0000000a	0x00000005	0x00000000
3064	addi $sp, $sp, -8	0x23bdfff8	0x00000001	0x00002fd4	0x00003090	0x00000000	0x00000005	0x00000000
3068	sw $a1, 4($sp)	0xafa50004	0x00000001	0x00002fd4	0x00003090	0x00000000	0x00000005	0x00000000
306c	sw $ra, 0($sp)	0xafbf0000	0x00000001	0x00002fd4	0x00003090	0x00000000	0x00000005	0x00000000
3070	addi $t0, $0, 2	0x20080002	0x00000001	0x00002fd4	0x00003090	0x00000000	0x00000005	0x00000000
3074	slt $t0, $a1, $t0	0x00a8402a	0x00000001	0x00002fd4	0x00003090	0x00000000	0x00000005	0x00000000
3078	beq $t0, $0, else	0x11000003	0x00000001	0x00002fd4	0x00003090	0x00000000	0x00000005	0x00000000
307c	addi $v0, $0, 1	0x20020001	0x00000001	0x00002fd4	0x00003090	0x00000001	0x00000005	0x00000000
3080	addi $sp, $sp, 8	0x23bd0008	0x00000001	0x00002fdc	0x00003090	0x00000001	0x00000005	0x00000000
3084	jr $ra	0x03e00008	0x00000001	0x00002fdc	0x00003090	0x00000001	0x00000005	0x00000000
3088	addi $a1, $a1, -1	0x20a5ffff	0x00000001	0x00002fdc	0x00003090	0x00000000	0x00000005	0x00000000
308c	jal factorial	0x0c000c19	0x00000001	0x00002fdc	0x00003090	0x00000000	0x00000005	0x00000000
3090	lw $ra, 0($sp)	0x8fbf0000	0x00000004	0x00002ff4	0x00003054	0x00000018	0x00000005	0x00000000
3094	lw $a1, 4($sp)	0x8fa50004	0x00000005	0x00002ff4	0x00003054	0x00000018	0x00000005	0x00000000
3098	addi $sp, $sp, 8	0x23bd0008	0x00000005	0x00002ffc	0x00003054	0x00000018	0x00000005	0x00000000
309c	mul $v0, $a1, $v0	0x70a21002	0x00000005	0x00002ffc	0x00003054	0x00000078	0x00000005	0x00000000
30a0	jr $ra	0x03e00008	0x00000005	0x00002ffc	0x00003054	0x00000078	0x00000005	0x00000000
30a4								

6. SCREEN CAPTURES

a. Figures 1 and 2 are screenshots of the assembly code in the MARS editor.

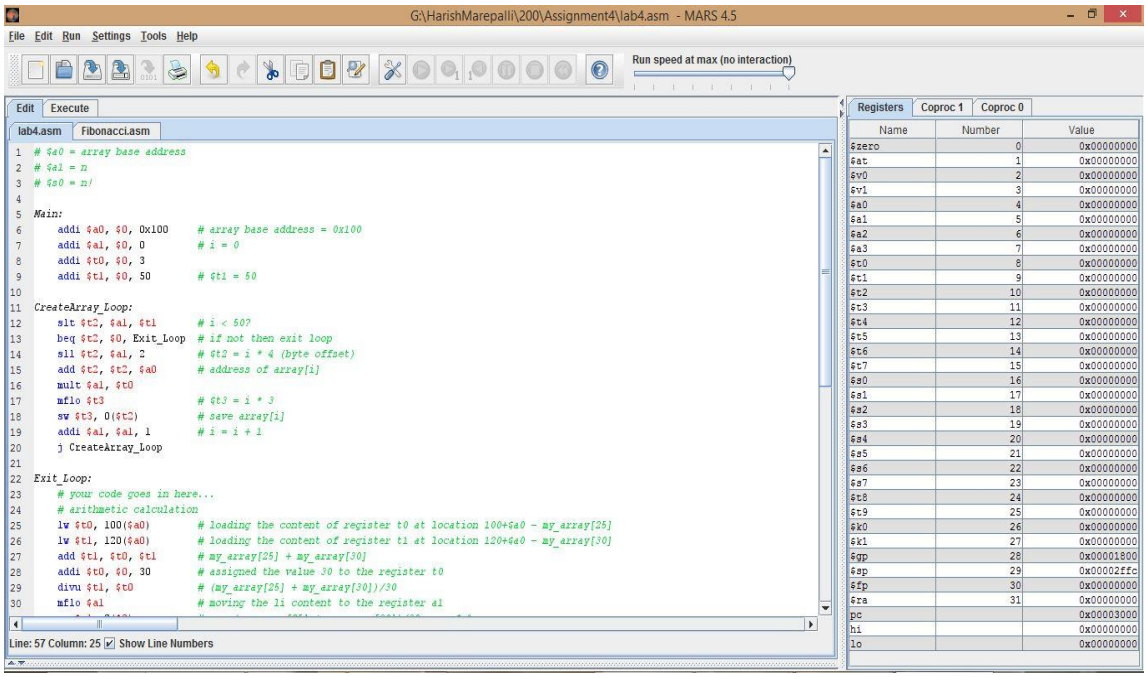


Fig. 1

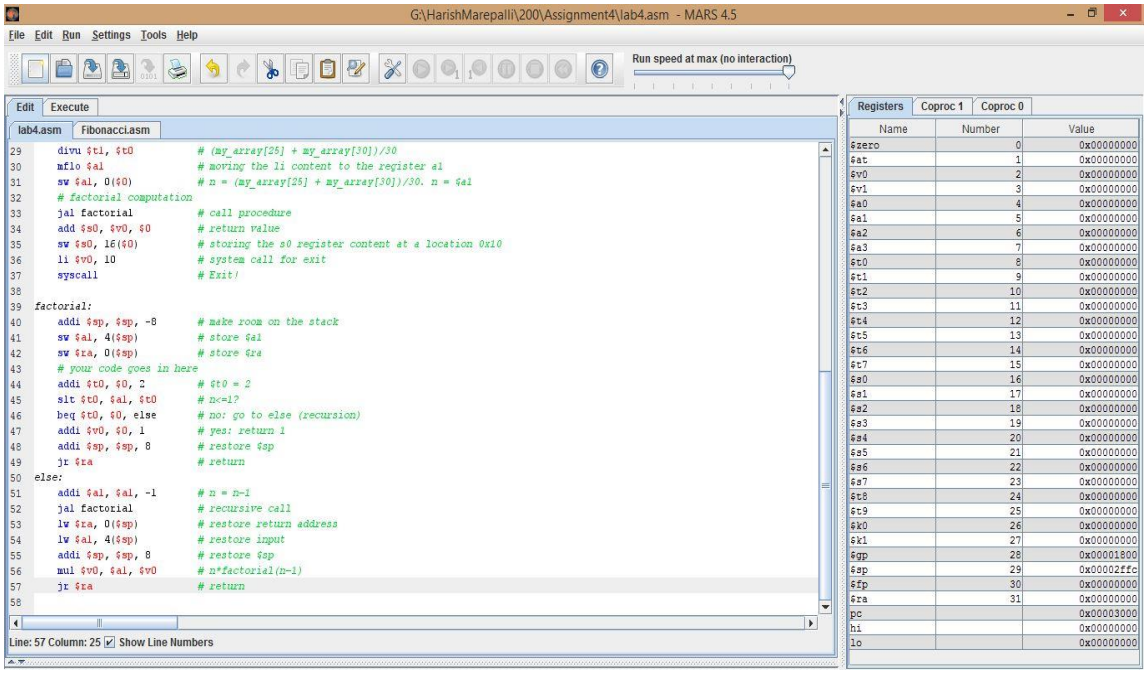


Fig. 2

b. Figure 3 is the screenshot of the execution window after assembling and before executing the code in MARS.

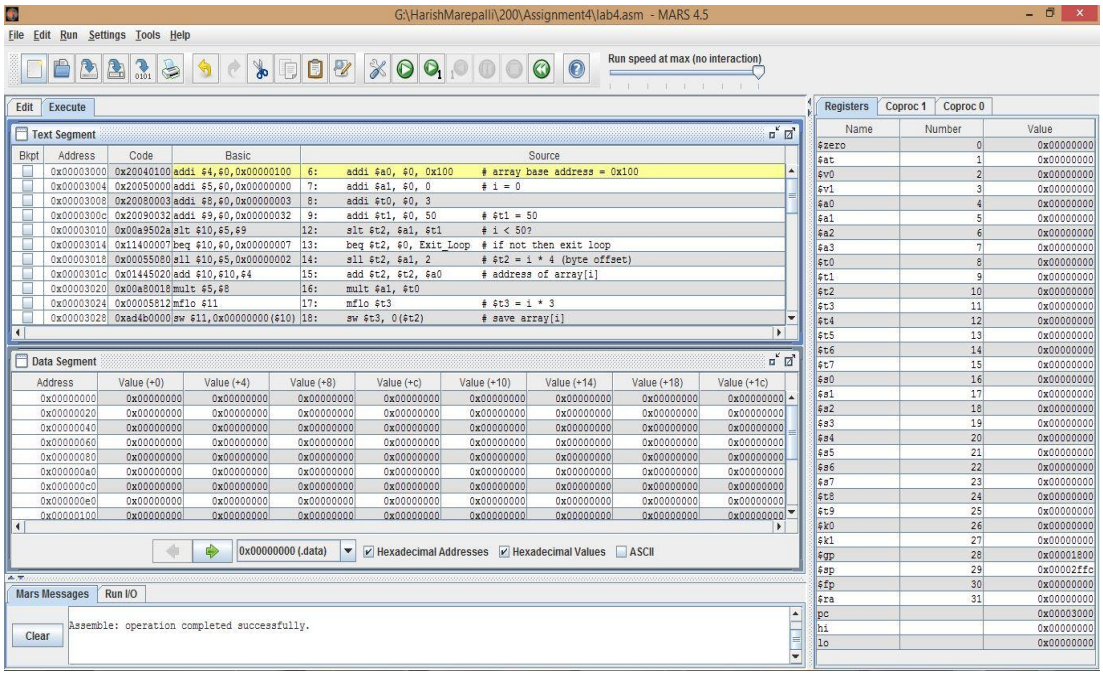


Fig. 3

c. Figure 4 is the screenshot of the execution window after assembling and after executing the code in MARS.

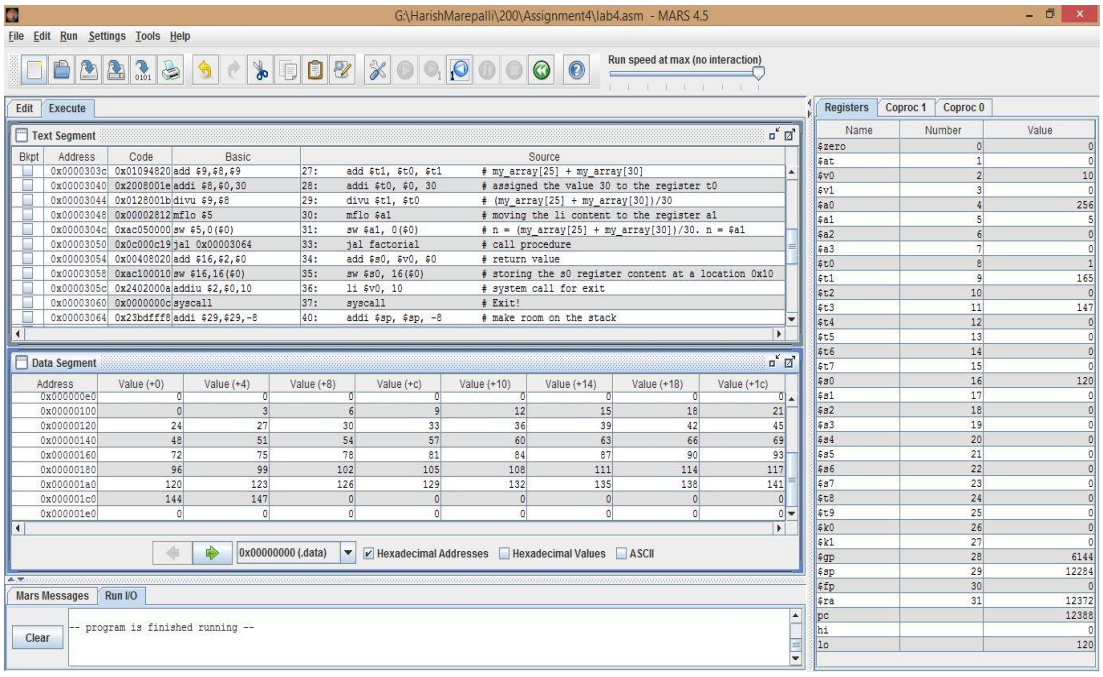


Fig. 4

7. STACK STATUS DIAGRAM

Figures 5 and 6 in the appendix are the stack status diagrams that show the addresses, stack pointer position, and values of $a1 and $ra after each iteration.

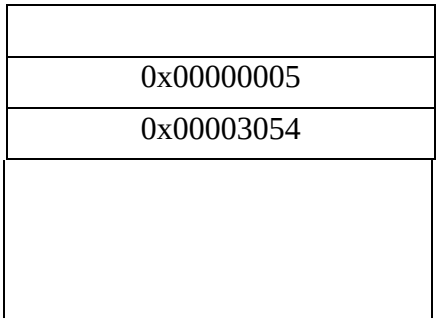
a. When n=5, $a1 = 0x00000005 and $ra = 0x00003054
b. When n=4, $a1 = 0x00000004 and $ra = 0x00003090
c. When n=3, $a1 = 0x00000003 and $ra = 0x00003090
d. When n=2, $a1 = 0x00000002 and $ra = 0x00003090
e. When n=1, $a1 = 0x00000001 and $ra = 0x00003090
f. When n=0: In this case also 2 locations will be allocated. Since the condition fails at this iteration, these 2 locations are deallocated again.

8. DISCUSSION SECTION

Following are the observations related to stack status:

a. For every iteration, the value in the Stack Pointer register changes.
b. The old values remain in the Stack Pointer register and the values get stacked up.
c. The value of a1 register changes accordingly with the value for which the factorial has to be calculated.
d. The value of the return address also can be noted from the stack.
e. At last, when the value of n reduces to zero, two more locations will be allocated and deallocated automatically.
f. This process is a recursive procedure since there is a recursive call in the program.
g. The values in the array get stored in the data segment addresses.

A sample stack diagram is shown below. The actual stack status diagram is shown in the appendix.



The top cell's code is 0x00002ffc and it is also called $SP_{old.}$

The cell which has the value 0x00000005 is a1 and it's code is 0x00002ff8.

The cell which has the value 0x00003054 is return address and it is also the SP_{new}. It's address is 0x00002ff4.

9. COLLABORATION SECTION

1. Understood the array operations by having a discussion and collaboration.
2. Understood how the arithmetic calculation is done in the MIPS program.
3. Written MIPS assembly program for the factorial program in MARS by collaborating with each other.
4. Understood how to perform a recursive factorial operation.
5. Discussed about the stack status and understood how the stack pointer changes for every iteration and how the values get stacked up by collaborating with each other.
6. Assembled and executed the codes and observed all the operations and values.
7. By collaborating with each other, debugged each line to know the contents of the relevant registers and recorded the memory values at certain addresses.
8. Understanding the stack operation was something was challenging, but by collaborating with each other, it was understandable.

10. CONCLUSION

In conclusion, by writing, assembling, and executing, gained familiarity with the MIPS implementation of arrays, stacks, procedures, and recursive procedures. I also understood the MIPS ISA, assembly programming, as well as testing. Finally, from this assignment, I gained familiarity about how the stack works.

APPENDIX

SOURCE CODE:

```
# $a0 = array base address
# $a1 = n
# $s0 = n!

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
   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
   lw $t0, 100($a0)           # loading the content of register t0 at location 100+$a0 -
my_array[25]
   lw $t1, 120($a0)           # loading the content of register t1 at location 120+$a0 -
my_array[30]
   add $t1, $t0, $t1          # my_array[25] + my_array[30]
   addi $t0, $0, 30           # assigned the value 30 to the register t0
   divu $t1, $t0              # (my_array[25] + my_array[30])/30
   mflo $a1                   # moving the li content to the register a1
   sw $a1, 0($0)              # n = (my_array[25] + my_array[30])/30. n = $a1
   # factorial computation
   jal factorial              # call procedure
   add $s0, $v0, $0           # return value
   sw $s0, 16($0)             # storing the s0 register content at a location 0x10
   li $v0, 10                 # system call for exit
   syscall                    # Exit!

factorial:
   addi $sp, $sp, -8          # make room on the stack
   sw $a1, 4($sp)             # store $a1
   sw $ra, 0($sp)             # store $ra
```

```
    # your code goes in here
    addi $t0, $0, 2          # $t0 = 2
    slt $t0, $a1, $t0        # n<=1?
    beq $t0, $0, else        # no: go to else (recursion)
    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
```

This is the source code for running the factorial. It involves several MIPS instructions.

STACK STATUS DIAGRAM:

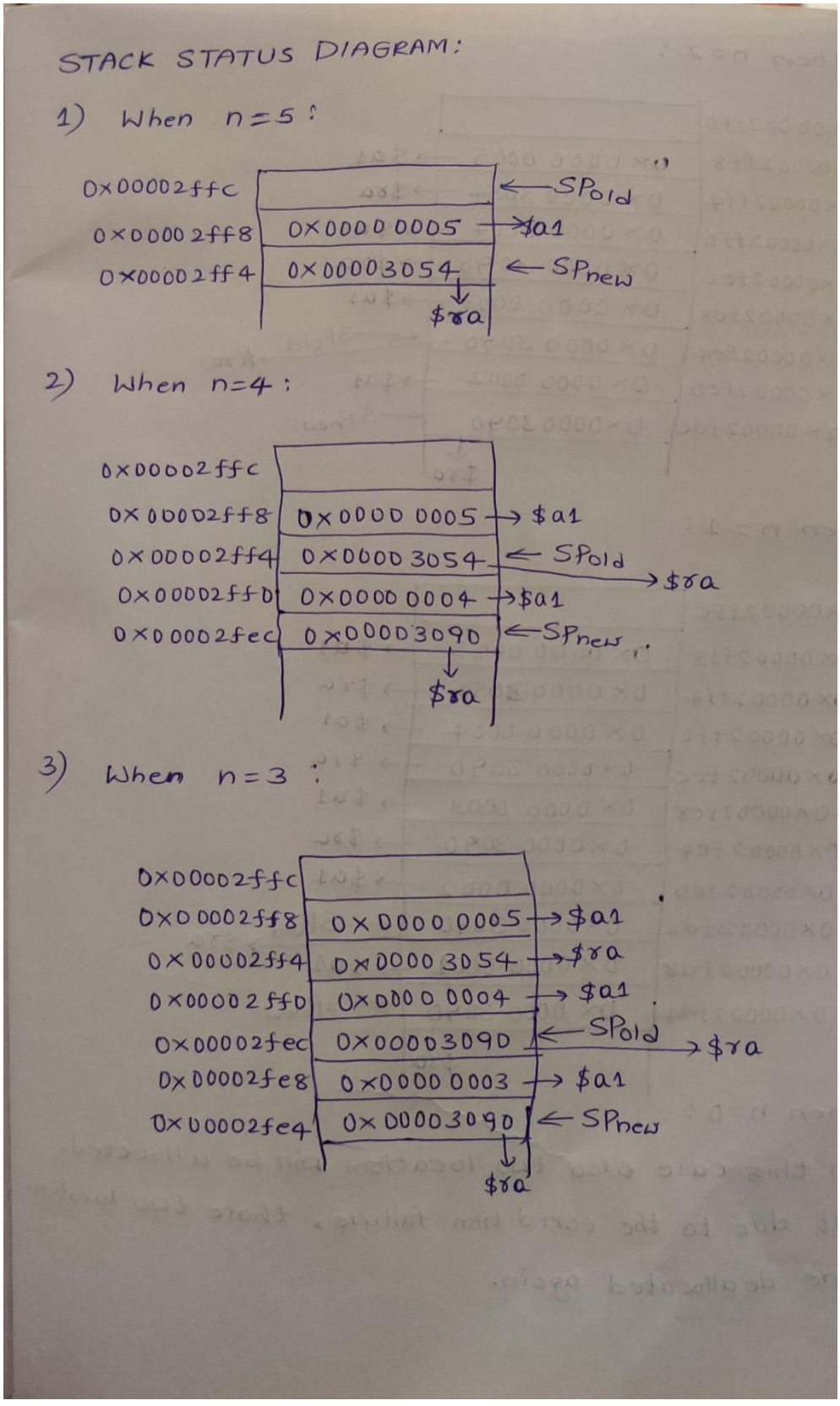
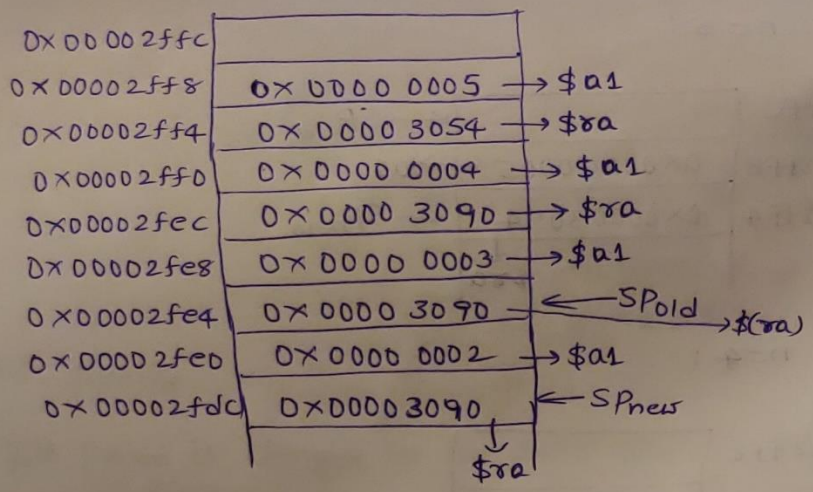
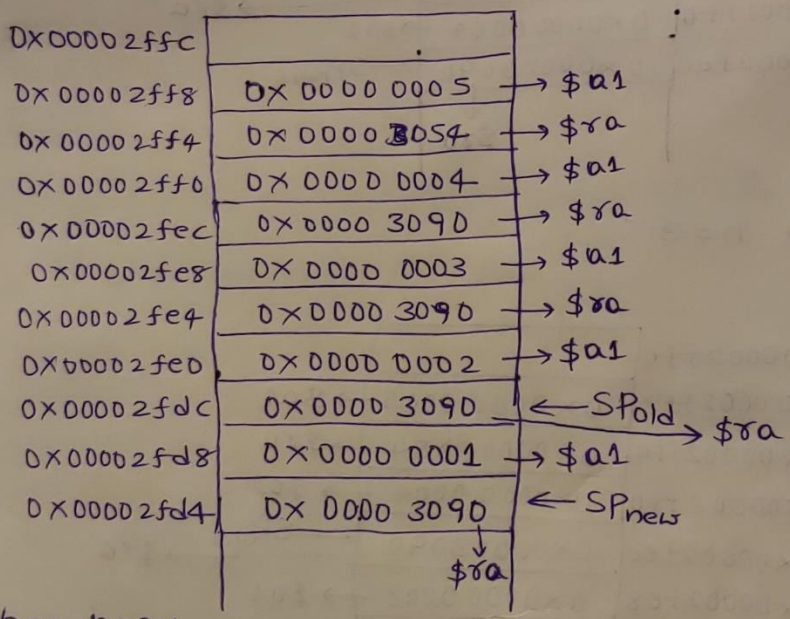


Fig. 5

4) When n=2:



5) When n=1:



5) When n=0:

In this case also two locations will be allocated. But due to the condition failure, those two locations are deallocated again.

Fig. 6