DHA Suffa University
Department of Computer Science
Computer Organization & Assembly Language
Spring 2021
Lab # 09 (Recursive Procedures & Stack)

Objective:

To understand the recursive calls of procedures using Stack.

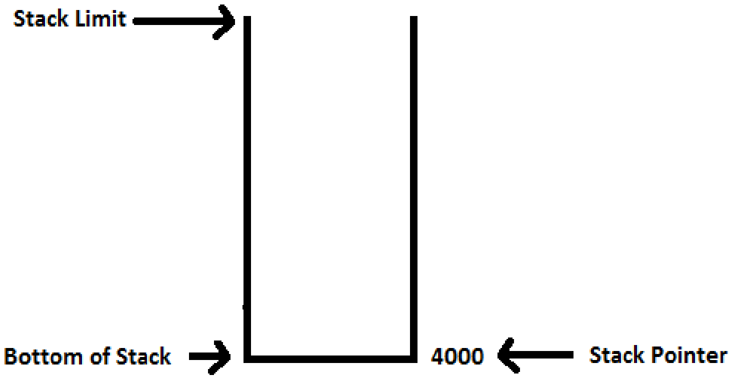
Use of Stack in MIPS:

There is no difference between the temporary and saved variables in how they work. The difference is in how they are used, or rather, how they ought to be used.

The MIPS calling convention specifies how the various registers are to be used --
the `$v` registers are for function returns, the `$a` registers are for function arguments,
the `$t` variables are temporary *caller saved* registers, while the `$s` registers are *callee saved*.

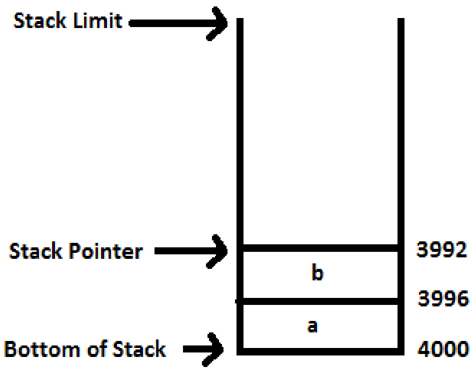
The difference between callee and caller saved is as follows: when calling a function, the convention guarantees that the `$s` registers are the same after return whereas the convention does not guarantee this for the `$t` registers. Of course this means that if you wish to use the `$s` registers in a routine, you must save and restore their values.

While working with procedures in MIPS one should have clear concept of how stack works. Look at the figure below:



Initially when Stack is empty then Stack Pointer points at the Bottom of Stack as shown in the above figure. When something needs to be pushed on the Stack then it is needed to decrement the Stack Pointer. In MIPS $SP register contains the Stack Pointer value.

If a variable “a” is pushed on the Stack then Stack Pointer would be decreased by 4. If any other variable “b” is pushed on the Stack then Stack pointer would be further decreased by 4. After these two consecutive pushes Stack would loo k like below figure:



When it is needed to pop a value from stack then first increase the stack pointer by 4 and then pop that value and similarly do the same procedure to pop other values present in the stack.

Example 1

```
.data
newline: .asciiz "\n"
.text
main:

addi $s0, $zero, 100

jal valueIncrement

la $a0, newline
li $v0, 4
syscall

li $v0, 1
move $a0, $s0
syscall

li $v0,10
syscall


valueIncrement:
#save in memory
addi $sp, $sp, -4
sw $s0, 0($sp)
```

```
addi $s0, $s0, 100

li $v0, 1
move $a0, $s0
syscall

lw $s0, 0($sp)
addi $sp, $sp, 4

jr $ra
```

Example 2

```
.data
.text
.globl  main
.ent main
main:
        li      $a0 , 6
        li      $a1 , 0
        li      $a2 , 1
        jal     fib
        move    $a0 , $s0
        li      $v0 , 1
        syscall
        li      $v0 , 10
        syscall
```

```
.end main
.globl  fib
.ent    fib
fib:
        subu    $sp , $sp , 4
        sw      $ra , ($sp)
        sub     $a0 , $a0 , 1
        blt     $a0 , 1   , fib_base
        jal     fib
        add     $s0 , $a2 , $a1
        move    $a1 , $a2
        move    $a2 , $s0
fib_base:
        lw      $ra , ($sp)
        addu    $sp , $sp , 4
        jr      $ra
.end    fib
```

Example 3

```
.data

msg1: .asciiz "Enter a number:\n"
msg2: .asciiz "Factorial = "
.text

li $t3, 1
la $a0, msg1
li $v0, 4
syscall


li $v0, 5
syscall
```

```
move $a0, $v0
move $a1, $a0

jal fact

la $a0, msg2
li $v0, 4
syscall

li $v0, 1
move $a0, $v1
syscall

li $v0, 10
syscall

fact:
sub $sp, $sp, 4
sw $ra, ($sp)
sub $a0, $a0, 1
beq $a0, 0, basecase

jal fact
mul $a1, $a1, $t3
add $t3, $t3, 1
move $v1, $a1

basecase:
lw $ra, ($sp)
add $sp, $sp, 4
jr $ra
```

LAB ASSIGNMENT 09

(1) Write a Program to calculate m^n using recursive procedure calls.

(2) Write a procedure which takes a number as an argument and tells whether the number is prime or not.