Laboratory Exercise 7

Procedure calls, stack and parameters

Goals

After this laboratory exercise, you should understand how to invoke a procedure and the mechanism of stack. In addition, you will be able to write some procedures that use stack to pass parameters and return results also.

Literature

Behrooz Parhami (CAMS): Section 6.1

Preparation

Before you start the exercise, you should review the textbook, section 6.1 and read this laboratory carefully.

Procedure calls

A procedure is a subprogram that when called (initiated, invoked) performs a specific task, perhaps leading to one or more results, based on the input parameters (arguments) with which it is provided and returns to the point of call, having perturbed nothing else. In assembly language, a procedure is associated with a symbolic name that denotes its starting address. The jal instruction in MIPS is inteded specifically for procedure calls: it performs the control transfer (unconditional jump) to the starting address of the procedure, while also saving the return address in register $ra.

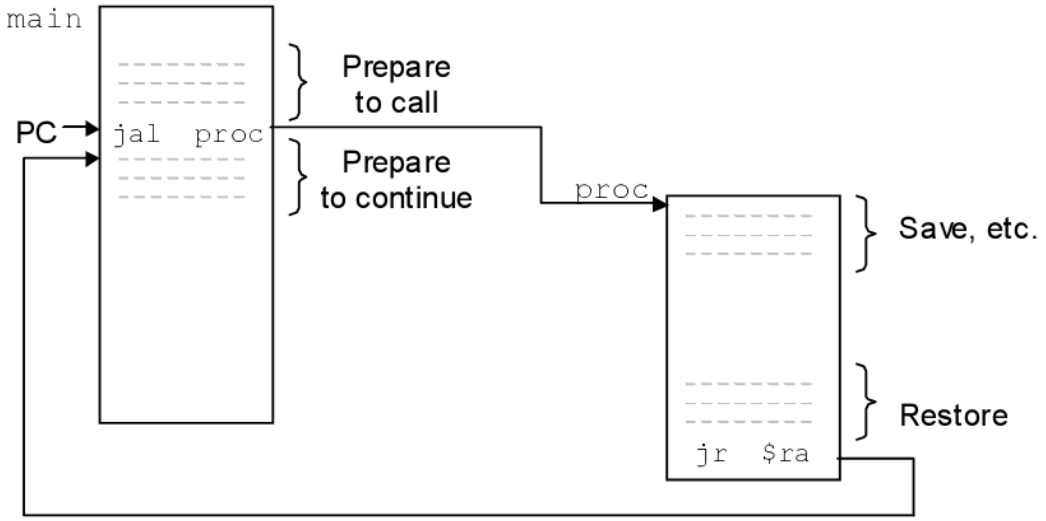


Figure 1. Relationship between the main program and a procedure

Assignments at Home and at Lab

Home Assignment 1

The following program uses **abs** procedure. This procedure is used to find the absolute value in an integer. This program uses two registers, $a0 for input argument and $v0 for result. You should read this example carefully, pay attention to how to write and invoke a procedure.

```
#Laboratory Exercise 7 Home Assignment 1
.text
main:   li      $a0, -45        #load input parameter
        jal     abs             #jump and link to abs procedure
        nop
        add     $s0, $zero, $v0
        li      $v0, 10         #terminate
        syscall
endmain:
#---------------------------------------------------------------
# function abs
# param[in]    $a0     the interger need to be gained the absolute
value
# return       $v0     absolute value
#---------------------------------------------------------------
abs:
      sub   $v0,$zero,$a0     #put -(a0) in v0; in case (a0)<0

      bltz  $a0,done          #if (a0)<0 then done
      nop
      add   $v0,$a0,$zero     #else put (a0) in v0
done:
      jr    $ra
```

Home Assignment 2

In this example, procedure **max** has the function to find the largest elements in three of integers. These intergers are passed to max procedure through $a0, $a1 and $a2 registers. Read this example carefully and try to explain each line of code.

```
#Laboratory Exercise 7, Home Assignment 2
.text
main:  li    $a0,2            #load test input
       li    $a1,6
       li    $a2,9
       jal   max              #call max procedure
       nop
endmain:
#---------------------------------------------------------------
-
#Procedure max: find the largest of three integers
#param[in]  $a0  integers
#param[in]  $a1  integers
#param[in]  $a2  integers
#return     $v0   the largest value
#---------------------------------------------------------------
-
max:  add     $v0,$a0,$zero   #copy (a0) in v0; largest so far
      sub     $t0,$a1,$v0     #compute (a1)-(v0)
      bltz    $t0,okay        #if (a1)-(v0)<0 then no change
      nop
```

```
      add     $v0,$a1,$zero   #else (a1) is largest thus far
okay: sub     $t0,$a2,$v0     #compute (a2)-(v0)
      bltz    $t0,done        #if (a2)-(v0)<0 then no change
      nop
      add     $v0,$a2,$zero   #else (a2) is largest overall
done: jr      $ra             #return to calling program
```

Home Assignment 3

The following program demonstrates the push and pop operations with stack. The value of two register $s0 and $s1 will be swap using stack. Read this example carefully, pay attention to adjust operation and the order of push and pop (sw and lw instructions).

```
#Laboratory Exercise 7, Home Assignment 3
.text
push: addi   $sp,$sp,-8      #adjust the stack pointer
      sw     $s0,4($sp)      #push $s0 to stack
      sw     $s1,0($sp)      #push $s1 to stack
work: nop
      nop
      nop
pop:  lw     $s0,0($sp)      #pop from stack to $s0
      lw     $s1,4($sp)      #pop from stack to $s1
      addi   $sp,$sp,8       #adjust the stack pointer
```

Parameters and results

In this section, we answer the following unresolved questions relating to procedures:

1. How to pass more than four input parameters to a procedure or receive more than two results from it?
2. Where does a procedure save its own parameters and immediate results when calling another procedure (nested calls)?

The stack is used in both cases. Before a procedure call, the calling program pushed the contents of any register that need to be saved onto the top of the stack and follow these with any additional arguments for the procedure. The procedure can access these arguments in the stack. After the procedure terminates, the calling procedure expects to find the stack pointer undisturbed, thus allowing it to restore the save registers to their original states and proceed with its own computations. Thus, a procedure that uses the stack by modifying the stack pointer must save the content of the stack pointer ($sp) at the outset and, at the end, restore $sp to its original state. This is done by copying $sp into the frame pointer register ($fp). Before doing this, however, the old contents of $fp must be saved. Hence, while a procedure is executing, $fp may hold the content of $sp right before it was called; $fp and $sp together “frame” the area of the stack that is in use by the current procedure.

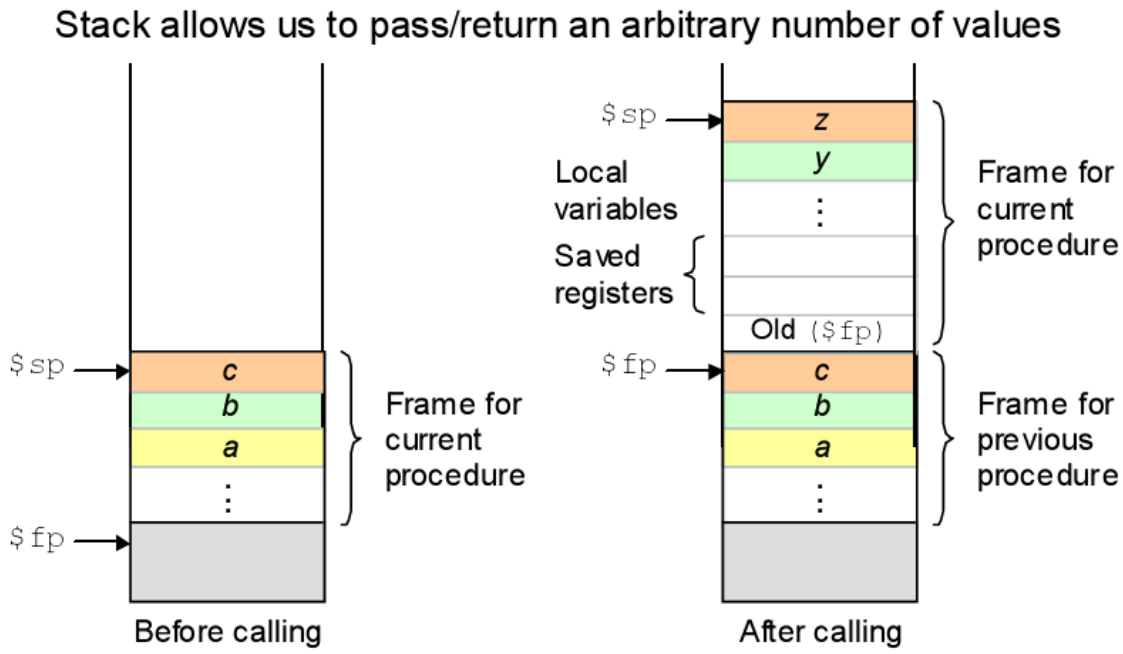


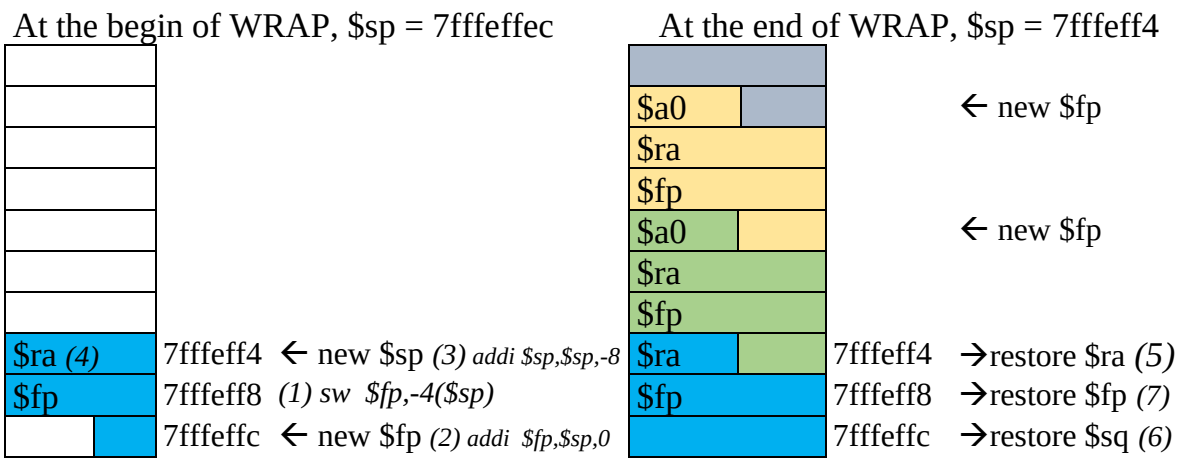
Figure 2. Use of the stack by a procedure

Home Assignment 4

The following program is a recursive procedure for computing n!. Read this program carefully and pay attention to register $sp and $fp at the start and the end of each recursive step.

```
#Laboratory Exercise 7, Home Assignment 4
.data
Message: .asciiz  "Ket qua tinh giai thua la: "

.text
main:  jal     WARP

print: add     $a1, $v0, $zero # $a0 = result from N!
       li      $v0, 56
       la      $a0, Message
       syscall
quit:  li      $v0, 10         #terminate
       syscall
endmain:

#---------------------------------------------------------------------
-
#Procedure WARP: assign value and call FACT
#---------------------------------------------------------------------
-
WARP:    sw      $fp,-4($sp)    #save frame pointer (1)
         addi    $fp,$sp,0      #new frame pointer point to the top (2)
         addi    $sp,$sp,-8     #adjust stack pointer (3)
         sw      $ra,0($sp)     #save return address (4)

         li      $a0,6          #load test input N
         jal     FACT           #call fact procedure
         nop

         lw      $ra,0($sp)     #restore return address (5)
         addi    $sp,$fp,0      #return stack pointer  (6)
```

```
        lw      $fp,-4($sp)    #return frame pointer  (7)
        jr      $ra
wrap_end:

#-----------------------------------------------------------------------
-
#Procedure FACT: compute N!
#param[in]  $a0  integer N
#return     $v0   the largest value
#-----------------------------------------------------------------------
-
FACT:   sw      $fp,-4($sp)         #save frame pointer
        addi    $fp,$sp,0           #new frame pointer point to stack's
top
        addi    $sp,$sp,-12         #allocate space for $fp,$ra,$a0 in
stack
        sw      $ra,4($sp)          #save return address
        sw      $a0,0($sp)          #save $a0 register

        slti    $t0,$a0,2           #if input argument N < 2
        beq     $t0,$zero,recursive#if it is false ((a0 = N) >=2)
        nop
        li      $v0,1               #return the result N!=1
        j       done
        nop
recursive:
        addi    $a0,$a0,-1          #adjust input argument
        jal     FACT                #recursive call
        nop
        lw      $v1,0($sp)          #load a0
        mult    $v1,$v0             #compute the result
        mflo    $v0
done:   lw      $ra,4($sp)          #restore return address
        lw      $a0,0($sp)          #restore a0
        addi    $sp,$fp,0           #restore stack pointer
        lw      $fp,-4($sp)         #restore frame pointer
        jr      $ra                 #jump to calling
fact_end:
```



Assignment 1

Create a new project to implement the program in Home Assignment 1. Compile and upload to simulator. Change input parameters and observe the memory when run the program step by step. Pay attention to register $pc, $ra to clarify invoking procedure process (Refer to figure 7).

Assignment 2

Create a new project to implement the program in Home Assignment 2. Compile and upload to simulator. Change input parameters (register $a0, $a1, $a2) and observe the memory when run the program step by step. Pay attention to register $pc, $ra to clarify invoking procedure process (Refer to figure 7).

Assignment 3

Create a new project to implement the program in Home Assignment 3. Compile and upload to simulator. Pass test value to registers $s0 and $s1, observe run process, pay attention to stack pointer. Goto memory space that pointed by $sp register to view push and pop operations in detail.

Assignment 4

Create a new project to implement the program in Home Assignment 4. Compile and upload to simulator. Pass test input through register $a0, run this program and test result in register $v0. Run this program in step-by-step mode, observe the changing of register $pc, $ra, $sp and $fp. Draw the stack through this recursive program in case of n=3 (compute 3!).

Assignment 5

Write a procedure to find the largest, the smallest and these positions in a list of 8 elements that are stored in regsiters $s0 through s7. For example:
Largest: 9,3 => The largest element is stored in $s3, largest value is 9
Smallest: -3,6 => The smallest element is stored in $s6, smallest value is -3
Tips: using stack to pass arguments and return results.

Conclusions

Before you pass the laboratory exercise, think about the questions below:

- What registers that the Caller need to save by convention?
- What registers that the Callee need to save by convention?
- In push label of Home Assignment 3, could we change the order of adjust stack and store word operations? If yes, what should we have to modify?
- What is stack pointer?
- What is frame pointer?