

PC - holds addr of current instr.

\$ra

jal  
jr

mechanisms

---

Policy

param. passing  
return value  
cleanup  
setup

50-57

↳ saved registers

t0-t9

↳ temp

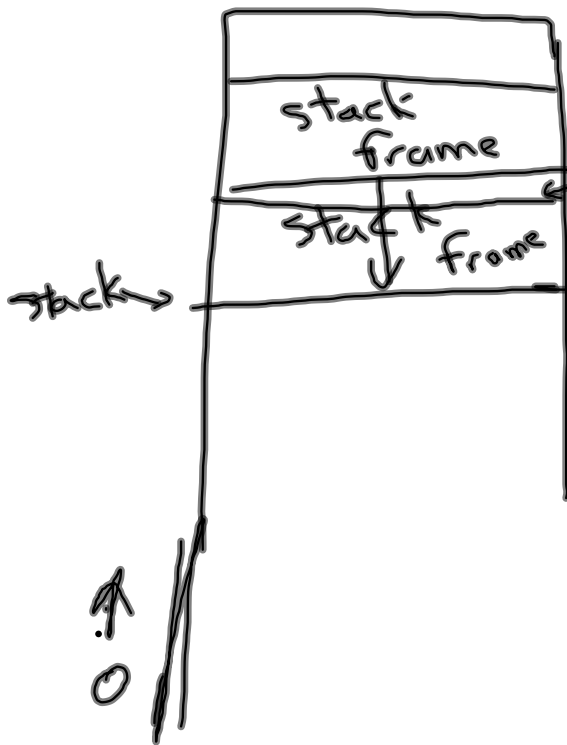
v0-v1

↳ return values

a0-a3

↳ arguments

# Method Calls Stack

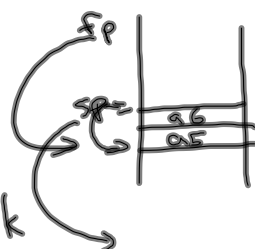


## Stack frame

- parameters
- local variables
- return value
- return addr

## Calling conventions

### Caller:

- save registers that are "temporary"  
 $a0..a3, t0..t9$   
(copy onto stack)
  - pass parameters  
first 4:  $a0..a3$   
next: go on stack
  - execute `jal` (go to functions first instr.)
- 
- The diagram shows a vertical stack of memory cells. The top cell is labeled 'fp' (frame pointer). Below it are four cells labeled 'a0', 'a1', 'a2', and 'a3'. Below those are two more cells labeled 'a5' and 'a6'. An arrow labeled 'sp' (stack pointer) points to the top of the 'a5' cell. Another arrow labeled 'fp' points to the top of the 'fp' cell. A curved arrow points from the 'sp' label to the 'fp' label, indicating the relationship between the two pointers.

### function

- allocate memory on the stack  
`addi $sp, $sp, 8`
- save "callee-saved" registers  
 $s0..s7, fp, ra$   
`sw $fp, 0($sp)`  
`sw $ra, 4($sp)`
- update  $fp$   
 $fp = sp + (\text{frame size} - 4)$

### Return

1. put return value in  $v0$  or  $v1$
2. restore "callee-saved" registers
3. pop the stack frame  
 $sp = sp + \text{frame size}$
4. `jr $ra`

<http://cs.gettysburg.edu/~cpresser/cs221/examples/3-8/>

```
main:
    .text
    .globl main
    #ask for a number
    addi $v0, $zero, 51    #load the syscall id into v0
    la   $a0, msg         #la - load the address of the string
    syscall
    add  $s0, $a0, $zero  #copy the return value into s0

    #ask for a number
    addi $v0, $zero, 51
    la   $a0, msg
    syscall
    add  $s1, $a0, $zero  #copy the return value into s1

    #call the sum functional
    add  $a0, $s0, $zero  #set up the first parameter
    add  $a1, $s1, $zero  #set up the second parameter

    jal  sum              #call the method
    add  $s2, $v0, $zero  #copy the return value into a variable

    #print the result
    addi $v0, $zero, 56
    la   $a0, msg2       #prep the string argument
    add  $a1, $s2, $zero  #prep the int argument
    syscall              #call the MessageDialogInt system call

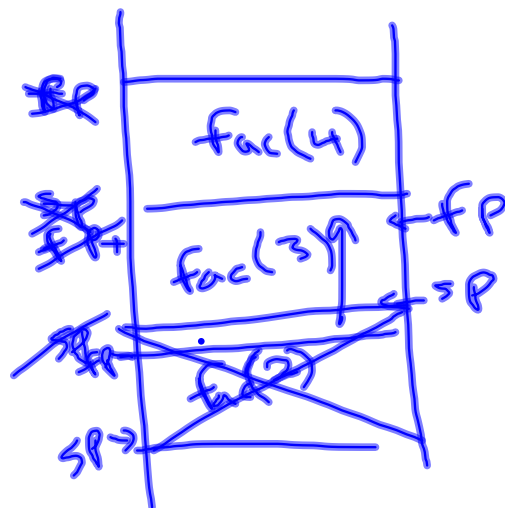
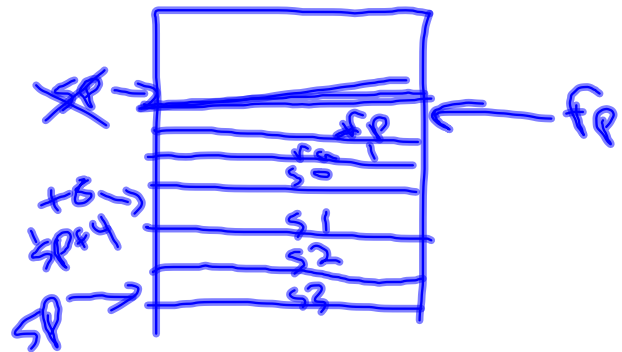
    #exit
    addi $v0, $zero, 10
    syscall

    .data
    .align 0
msg:   .ascii "Enter a number."
msg2:  .ascii "The sum is: "
```

SUM:

```
#allocate memory on the stack for the variable we use.  
#save the saved registers we will use: fp, ra, s0, s1, s2, s3  
addi $sp, $sp, -24  
sw $fp, 20($sp)  
sw $ra, 16($sp)  
sw $s0, 12($sp)  
sw $s1, 8($sp)  
sw $s2, 4($sp)  
sw $s3, 0($sp)  
  
#modify the frame pointer  
addi $fp, $sp, 20
```

} put registers  
on the stack





```

#make s0 the smaller of a0 and a1
# if(a0 < a1){
#     s0 = a0;
#     s1 = a1;
# }
# else {
#     s0 = a1;
#     s1 = a0;
# }
#make s1 the larger
slt    $t0,    $a0,    $a1    #set t0 to 1 if a0 < a1
beq    $t0,    $zero,  else
add    $s0,    $zero,  $a0
add    $s1,    $zero,  $a1
j      endif
else:
add    $s0,    $zero,  $a1
add    $s1,    $zero,  $a0
endif:

#perform the calculation (loop from one to the other)
#for(s2 = s0; s2 <= s1; s2++)
#   s3 = s3 + s2;
add    $s2,    $zero,  $s0
add    $s3,    $zero,  $zero    #s3 = 0
comp:  addi    $t1,    $s1,    1
slt    $t0,    $s2,    $t1    #check if s2 <= s1
beq    $t0,    $zero,  end    #if not fall out of the loop
add    $s3,    $s3,    $s2
addi   $s2,    $s2,    1
j      comp
end:

```

```
#put the return value in v0
add    $v0,    $s3,    $zero

#restore the saved registers
lw     $fp,    20($sp)
lw     $ra,    16($sp)
lw     $s0,    12($sp)
lw     $s1,    8($sp)
lw     $s2,    4($sp)
lw     $s3,    0($sp)
#pop the stack frame
addi   $sp,    $sp, -24
#return to the caller's address
jr     $ra
```