Agenda

- Announcements
- Introduction
- MIPS instructions: procedures and stacks

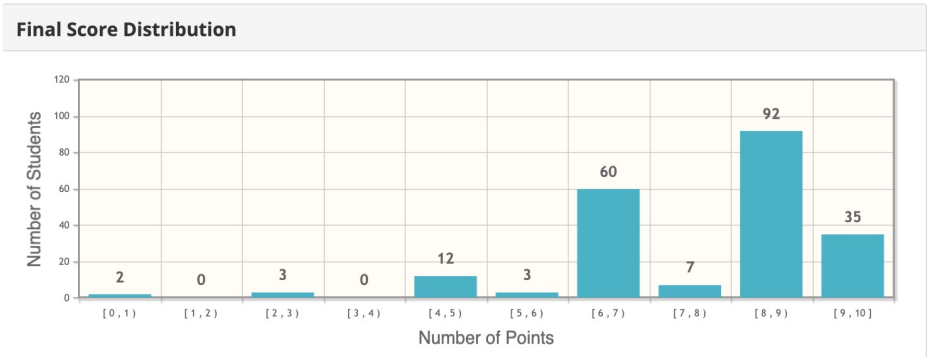
Announcements (3/31)

- Labs:
  - Lab 4 - don't forget due this Thursday
  - Lab 5 - released on this Friday @ noon
- Quiz 5
  - Announcement and PEW questions
- Today
  - Finish stack operations/primitives
  - Stacks and procedures

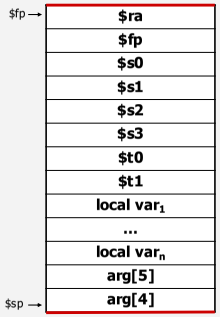
Announcements (4/5)

- Labs:
  - Lab 5 - don't forget due this Thursday
  - Lab 6 - released on this Friday @ noon
- Quiz 5
  - Great, job!
  - i-type instructions, sign-extended (SE) and zero-extended (ZE)
  - Communicate with your cohort leader ☺!
- Quiz 6
  - This Friday
  - Thursday, announcement and PEW questions
- Participation assignment
  - Used for participation grade (5%)
  - Create three questions (per assignment PDF)
  - Communicate with your cohort leader
- Today
  - Finish stacks and procedures

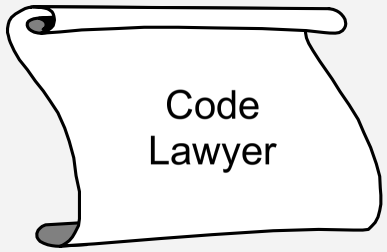
Grade Statistics	
Submissions	214
Total Score Possible	10
Mean	7.34
Median	8
Mode	8
Range	0 - 10
Quartile 1	6
Quartile 3	8
Standard Deviation	1.82



Overview

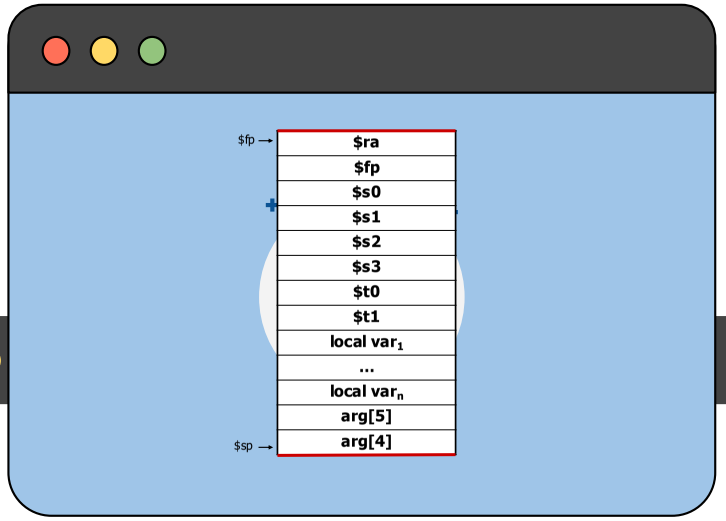


Contracts



Stack Frame Templates

Example



Overview

- Refresher
- Allocated memory
- Convention

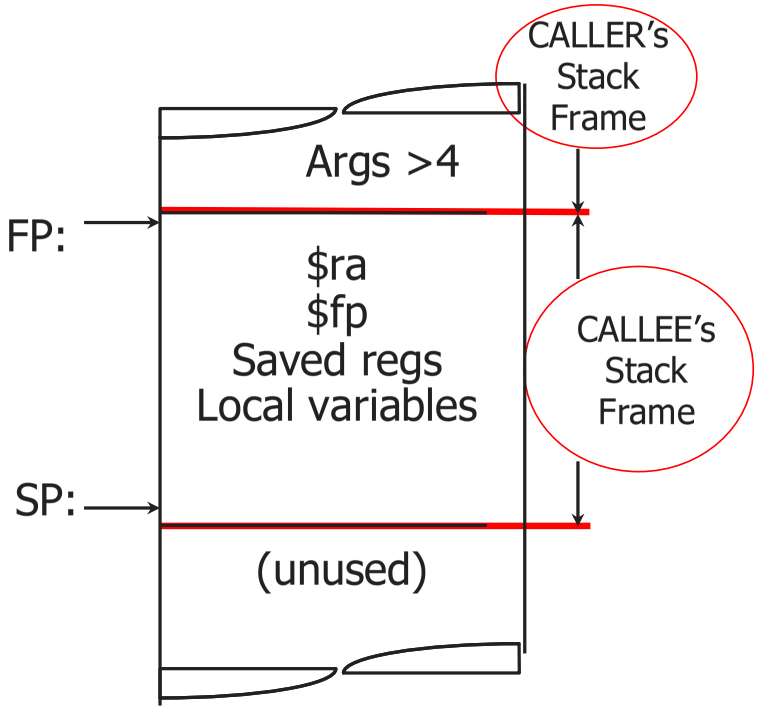
Refresher

- Call a procedure
  - jal instruction
- Return from a procedure
  - ja instruction
  - $31 = $ra (return address back to caller)
- Pass arguments into a procedure
  - $a registers
- Return values from a procedure
  - $v registers
- Store variables
  - Create a Stack Frame
  - $30 = $fp (frame pointer)
    - points to the start of callee's activation record on the stack
    - we also use it to access extra args (>4)
  - $29 = $sp (stack pointer, points to TOP of stack)
    - points to the end of callee's activation record on the stack
  - Together: $29 and $30 are bookends to activation record

Stack Frame

At a minimum, allocated memory (i.e., stack frame) for the following

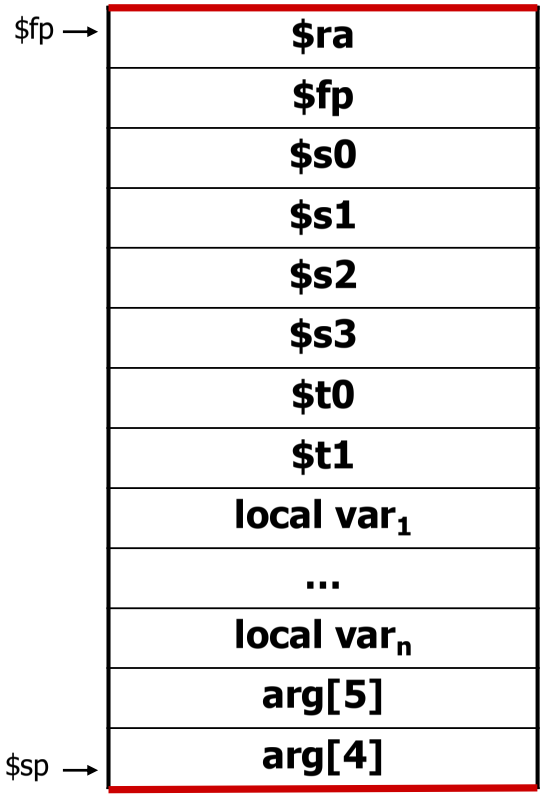
- Return to caller
  - Frame and stack pointer (more on this ☺)
  - return address
- Local variables
  - go out-of-scope when the callee procedure returns
- Temporary and saved variables
  - caller is expecting not to change
- Arguments if greater than 4
  - Callee will "spill" into stack frame
  - In reverse order



Convention: Typical Stack Frame

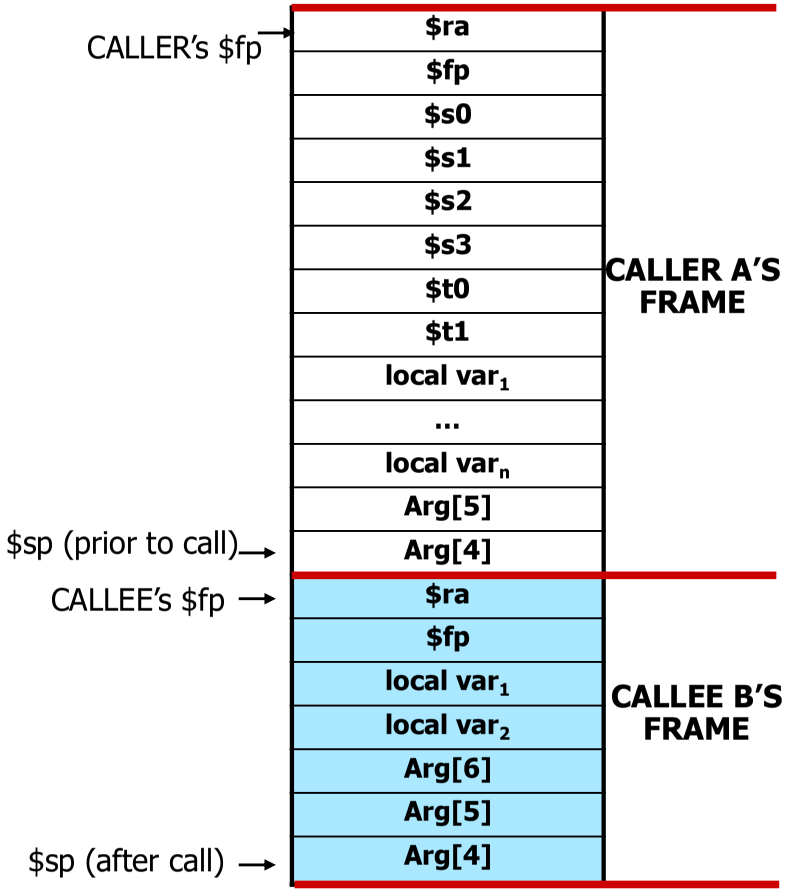
Stored in this order:

- $ra and $fp
- Saved registers (modified by the procedure)
  - e.g.: $s0, $s1, $s2, $s3
- Temporary registers (must survive procedure call)
  - e.g.: $t0, $t1
  - saved immediately before procedure call; restored immediately after
- local variables needed (not in registers)
  - e.g.: locals var_1 ... var_n
- Spilled arguments
  - Reverse order, e.g., arg[4], arg[5]



Convention: Caller and Callee

- Procedure A (CALLER) procedure B (CALLEE)
- Inspecting the callers stack frame, can you tell the max number of arguments passed to procedure B?
  - Yes, 6
- Where in CALLEE's stack frame can the CALLER's $fp be found?
  - At -4($fp)
- Where in CALLEE's stack frame can the CALLER's $sp be found?
  - At 4($fp) … ahh … so tricky ☺
- Where in CALLEE's stack frame can the return address be found?
  - At 0($fp)



MIPS Register Convention

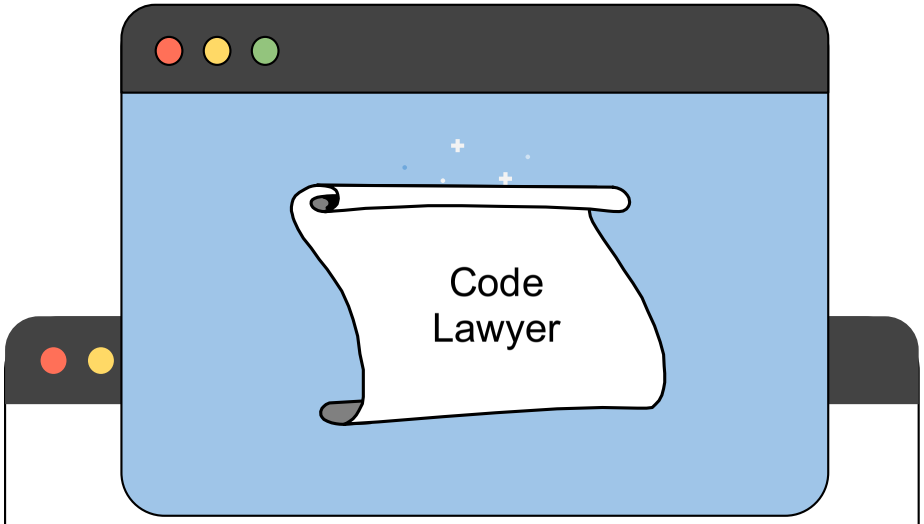
Will use

- temporary ($t) and saved ($s) registers
- stack frame registers ($fp and $sp)
- return address ($ra) register

Will not use

- registers reserved for system ($k0-$k1, $at)
- global variable ($gp) register (e.g,, addi $gp, $zero, 1000)

Name	Register number	Usage
$zero	0	the constant value 0
$at	1	assembler temporary
$v0-$v1	2-3	procedure return values
$a0-$a3	4-7	procedure arguments
$t0-$t7	8-15	temporaries
$s0-$s7	16-23	saved by callee
$t8-$t9	24-25	more temporaries
$k0-$k1	26-27	reserved for operating system
$gp	28	global pointer
$sp	29	stack pointer
$fp	30	frame pointer
$ra	31	return address



Caller and Callee Contracts

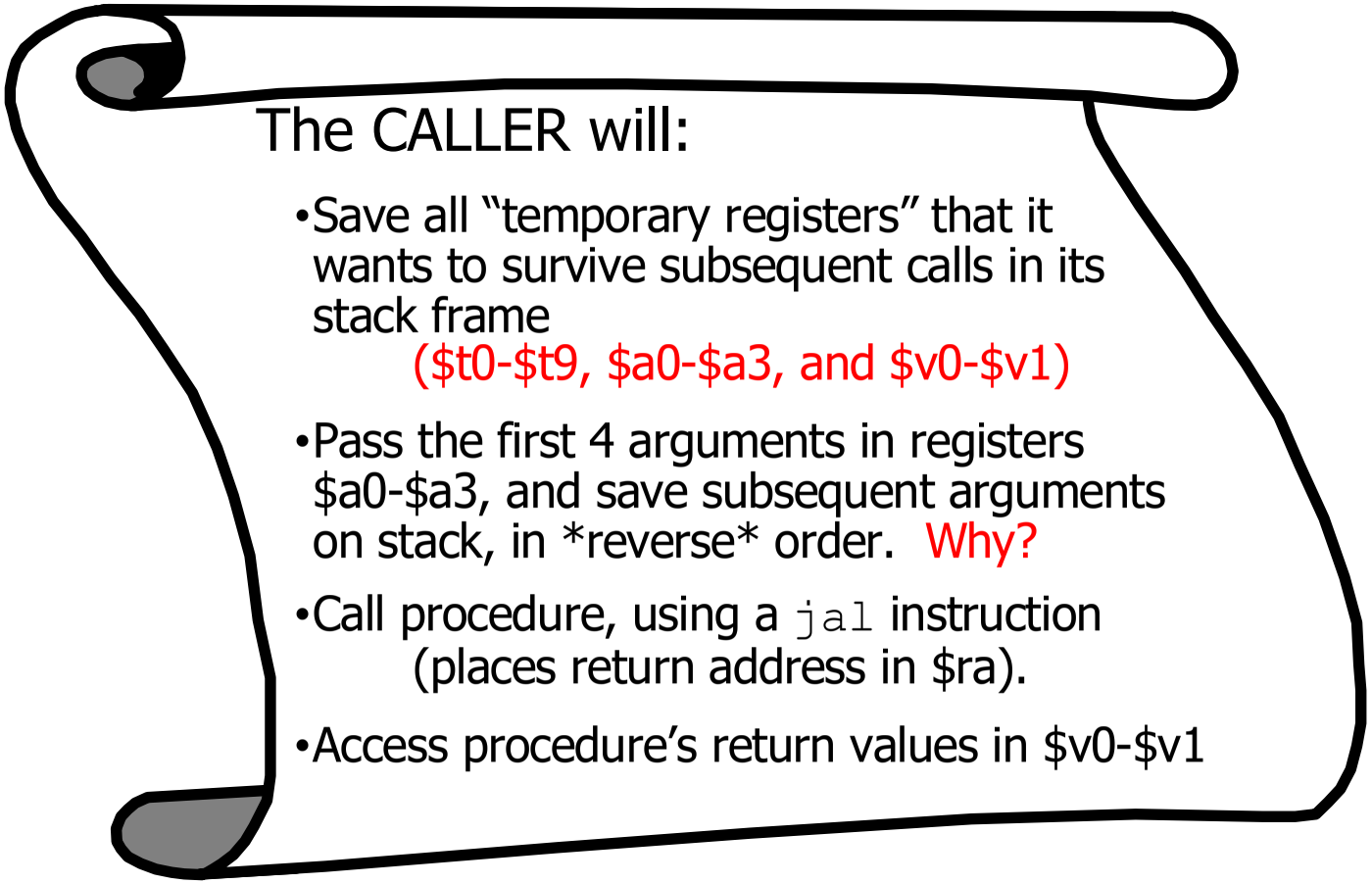
- Motivation
- Caller
- Callee

Motivation: What is Fair?

What registers need to be saved (in memory)?

- CHOICE 1… anything that a Callee touches
  - except the return value registers
- CHOICE 2… Give the Callee access to everything
  - Caller saves those registers it expects to remain unchanged
- CHOICE 3… Something in between
  - Give the Callee some "temporary" registers to play with
  - If the Caller cares about these, it must preserve them ($t registers)
  - Give the Caller some registers that the Callee won't clobber
  - If the Callee touches them, it must restore them ($s registers)

MIPS designers chose #3


The CALLER will:
•Save all "temporary registers" that it wants to survive subsequent calls in its stack frame
($t0-$t9, $a0-$a3, and $v0-$v1)
•Pass the first 4 arguments in registers $a0-$a3, and save subsequent arguments on stack, in *reverse* order. Why?
•Call procedure, using a jal instruction (places return address in $ra).
•Access procedure's return values in $v0-$v1

If needed the CALLEE will:

1) Allocate a stack frame with space for saved registers, local variables, and spilled args

2) Save any **"saved registers"** used:
($ra, ~~$sp~~, $fp, ~~$gp~~, $s0-$s7)
Note: $sp is not explicitly saved, but changes to it are simply undone. Also, we will ignore $gp.

3) If CALLEE has local variables -or- needs access to args on the stack, save CALLER's frame pointer and set $fp to 1st entry of CALLEE's stack

4) EXECUTE procedure
5) Place return values in $v0-$v1
6) Restore saved registers
7) Fix $sp to its original value
8) Return to CALLER with jr $ra

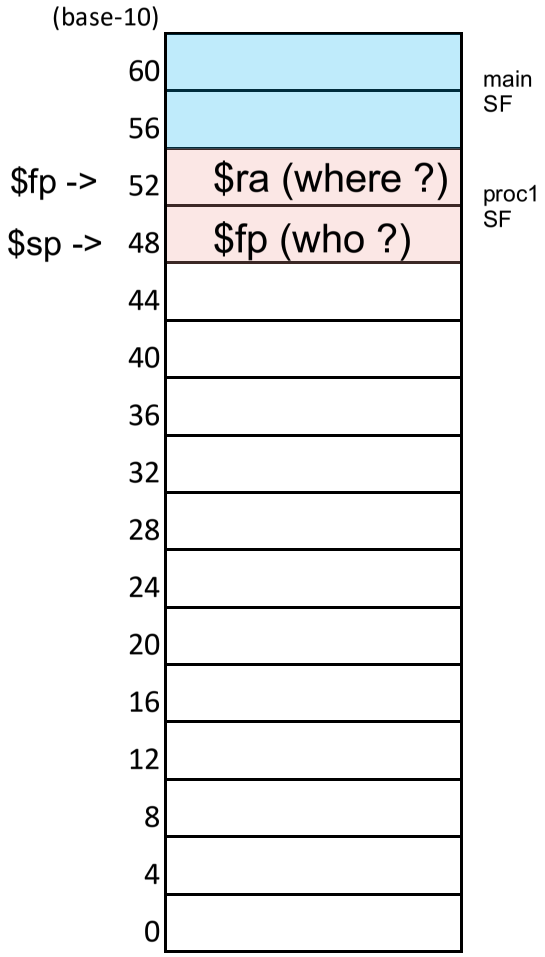
Stack Frame Templates (non-inclusive)

- Template 1
- Template 2
- Template 3
- Template 4
- Template 5
- Template 6

Template 1: leaf procedure with minimal stack frame

```
main:
    jal proc1              # call proc1
    ...                    # other main stuff

proc1:
    addi $sp, $sp, -8
    sw $ra, 4($sp)         # Save $ra
    sw $fp, 0($sp)         # Save $fp
    addi $fp, $sp, 4       # Set $fp

    ...          # do the task

    addi $sp, $fp, 4       # Restore $sp
    lw $ra, 0($fp)         # Restore $ra
    lw $fp, -4($fp)        # Restore $fp
    jr $ra                 # Return
```

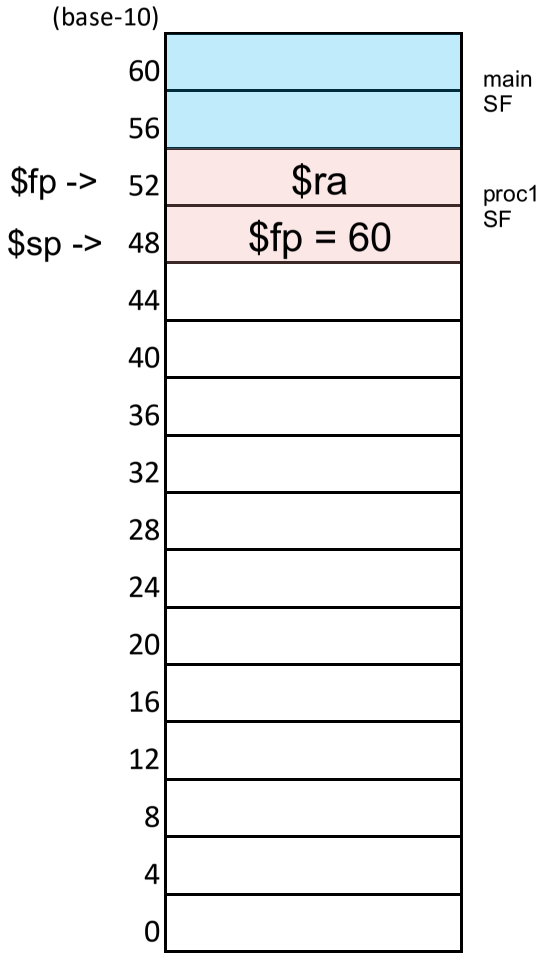
- main:
  - doesn't need to preserve any temporary registers
- proc1:
  - doesn't call another procedure
  - doesn't touch any saved regs
  - creates minimal stack frame



Template 2: non-leaf procedure with minimal stack frame

```
proc1:
    addi $sp, $sp, -8
    sw $ra, 4($sp)      # Save $ra
    sw $fp, 0($sp)      # Save $fp
    addi $fp, $sp, 4    # Set $fp

    ...
    jal proc2           # call another
    ...

    addi $sp, $fp, 4    # Restore $sp
    lw $ra, 0($fp)      # Restore $ra
    lw $fp, -4($fp)     # Restore $fp
    jr $ra              # Return
```

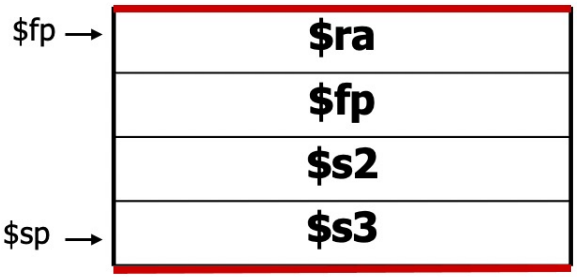
- proc1:
  - calls proc2
  - doesn't need to protect/preserve any registers
  - minimal stack frame

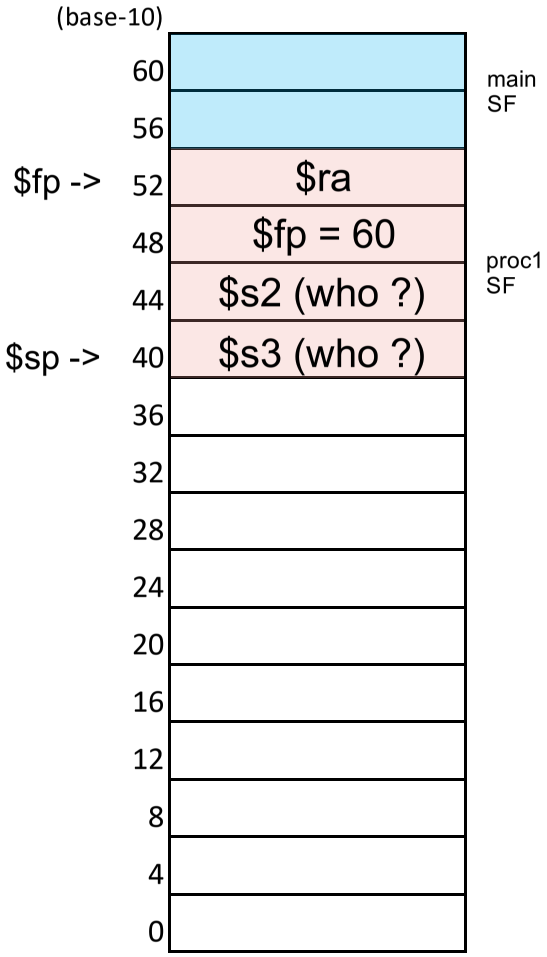


Template 3: leaf procedure that protects saved registers

```
proc1:
    addi $sp, $sp, -8
    sw $ra, 4($sp)      # Save $ra
    sw $fp, 0($sp)      # Save $fp
    addi $fp, $sp, 4    # Set $fp

    addi $sp, $sp, -8   # room for $s2-$s3
    sw $s2, 4($sp)      # Save $s2
    sw $s3, 0($sp)      # Save $s3

    ...                 # do the task

    lw $s2, -8($fp)     # restore $s2
    lw $s3, -12($fp)    # restore $s3

    addi $sp, $fp, 4    # Restore $sp
    lw $ra, 0($fp)      # Restore $ra
    lw $fp, -4($fp)     # Restore $fp
    jr $ra              # Return
```

- proc1:
  - is leaf
  - needs to save/restore $s2-$s3
  - Push on stack frame

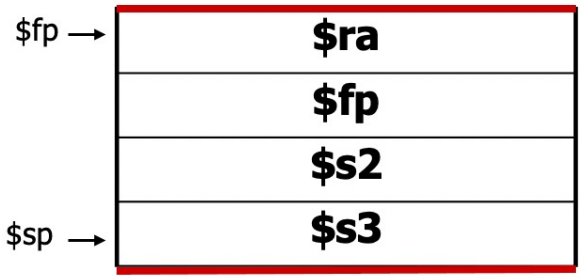


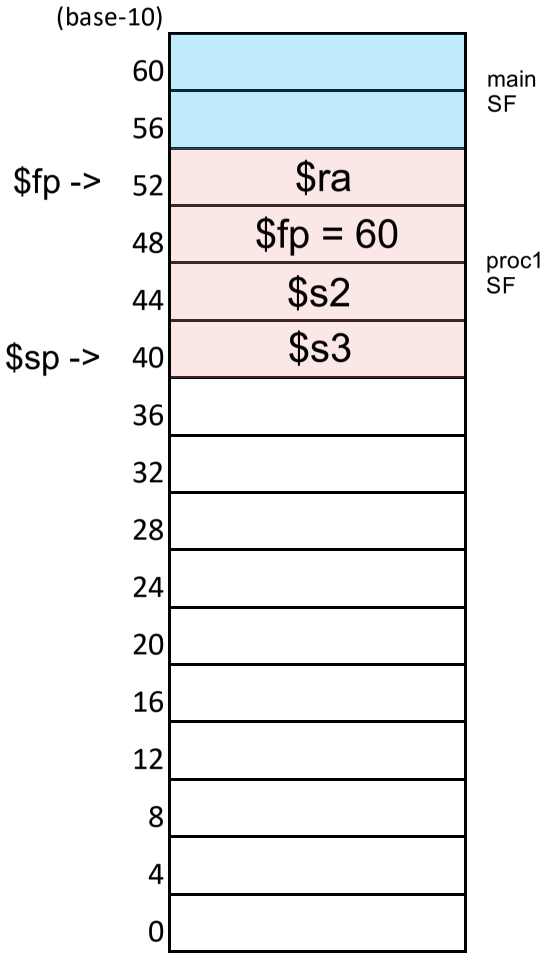


Template 4: non-leaf procedure protects saved registers

```
proc1:
    addi $sp, $sp, -8
    sw $ra, 4($sp)          # Save $ra
    sw $fp, 0($sp)          # Save $fp
    addi $fp, $sp, 4        # Set $fp

    addi $sp, $sp, -8       # room for $s2-$s3
    sw $s2, 4($sp)          # Save $s2
    sw $s3, 0($sp)          # Save $s3

    ...
    jal proc2               # call another
    ...

    lw $s2, -8($fp)         # restore $s2
    lw $s3, -12($fp)        # restore $s3

    addi $sp, $fp, 4        # Restore $sp
    lw $ra, 0($fp)          # Restore $ra
    lw $fp, -4($fp)         # Restore $fp
    jr $ra                  # Return
```

- proc1:
  - calls proc2
  - needs to save/restore $s2-$s3
  - Push on stack frame

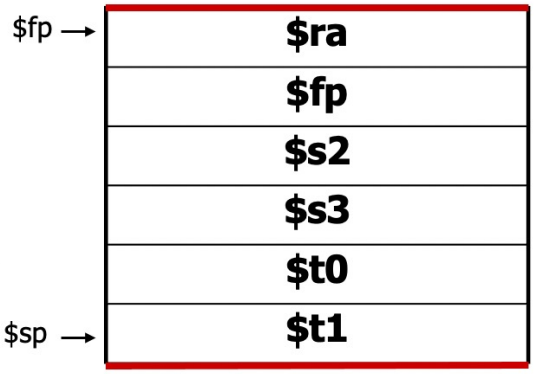


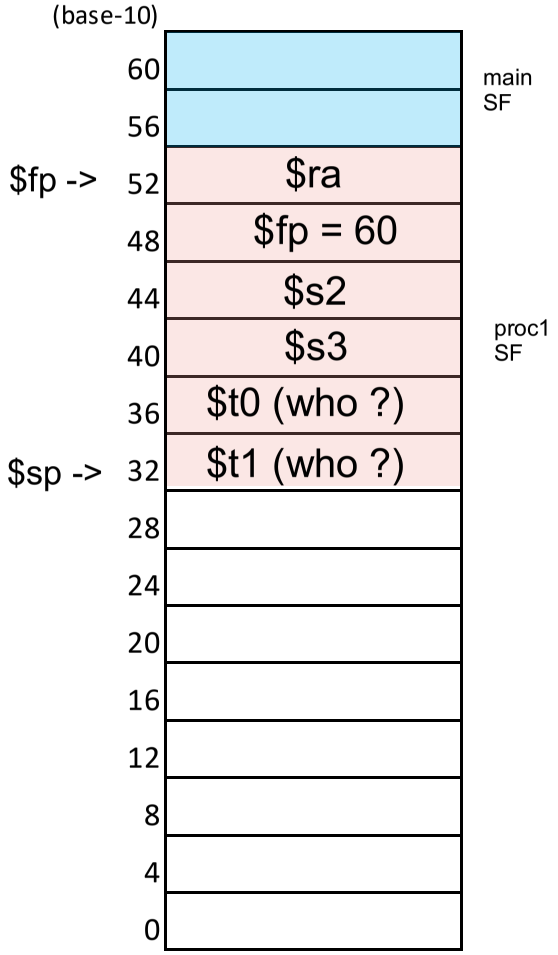


Template 5: non-leaf procedure that also protects temporary registers

```
proc1:
    addi $sp, $sp, -8
    sw $ra, 4($sp)          # Save $ra
    sw $fp, 0($sp)          # Save $fp
    addi $fp, $sp, 4        # Set $fp

    addi $sp, $sp, -8       # allocate memory
    sw $s2, 4($sp)          # save $s2
    sw $s3, 0($sp)          # save $s3
    ...
    addi $sp, $sp, -8       # allocate memory
    sw $t0, 4($sp)          # save $t0
    sw $t1, 0($sp)          # save $t1
    jal proc2               # call procedure
    lw $t0, -16($fp)        # restore $t0
    lw $t1, -20($fp)        # restore $t1
    ...
    lw $s2, -8($fp)         # restore $s2
    lw $s3, -12($fp)        # restore $s3

    addi $sp, $fp, 4        # Restore $sp
    lw $ra, 0($fp)                  # Restore $ra
    lw $fp, -4($fp)         # Restore $fp
    jr $ra                  # Return
```

- proc1:
  - calls proc2
  - needs to protect registers $t0-$t1 (i.e., survive proc2 procedure call).
  - push on stack frame



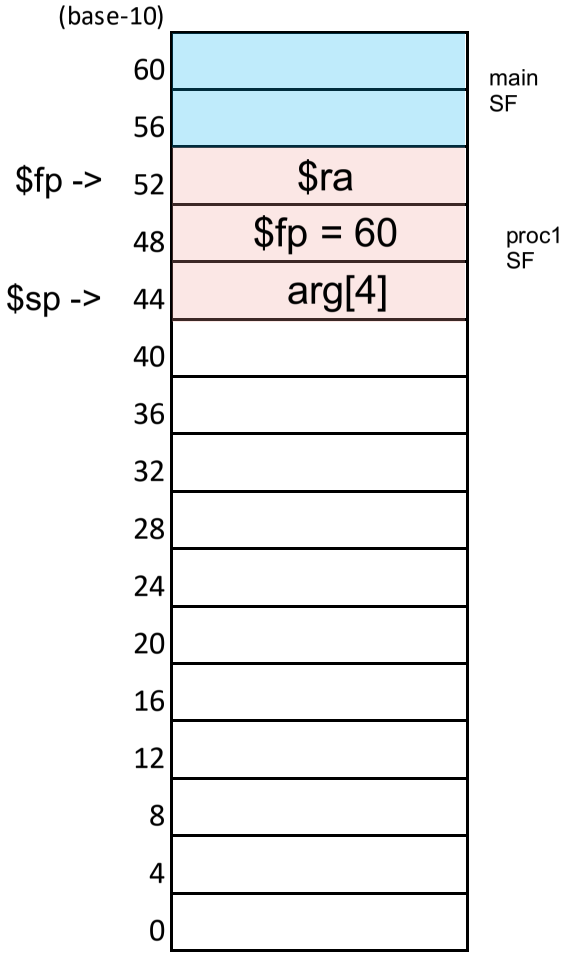


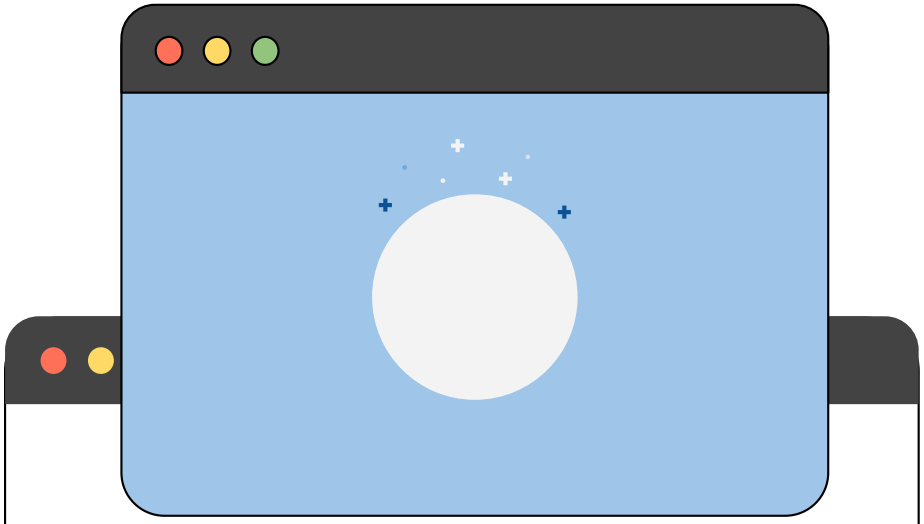
Template 6: non-leaf procedure, callee more than 4 arguments

```
proc1:
    addi $sp, $sp, -8
    sw $ra, 4($sp)          # Save $ra
    sw $fp, 0($sp)          # Save $fp
    addi $fp, $sp, 4        # Set $fp

    ...
    addi $sp, $sp, -4       # allocate memory
    ori $a0, $0, 40         # Put 40 in ...
    sw  $a0, 0($sp)         # save arg[4]

    ori $a0, $0,  0         # Put  0 in $a0
    ori $a1, $0, 10         # Put 10 in $a1
    ori $a2, $0, 20         # Put 20 in $a2
    ori $a3, $0, 30         # Put 30 in $a3

    jal proc2

    addi $sp, $fp, 4        # Restore $sp
    lw $ra, 0($fp)          # Restore $ra
    lw $fp, -4($fp)         # Restore $fp
    jr $ra                  # Return
```

- proc1:
  - calls proc2
  - proc2 has more than 4 arguments

proc1's stack frame





Example

➔ Work through together

Assume proc1 is called by main, and main expects $s0 register to survive procedure call.

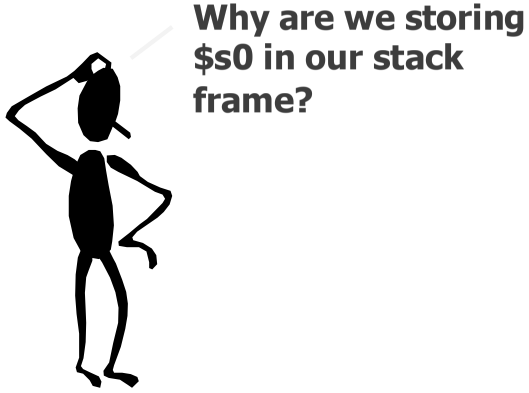
For the provided code, answer the following questions:

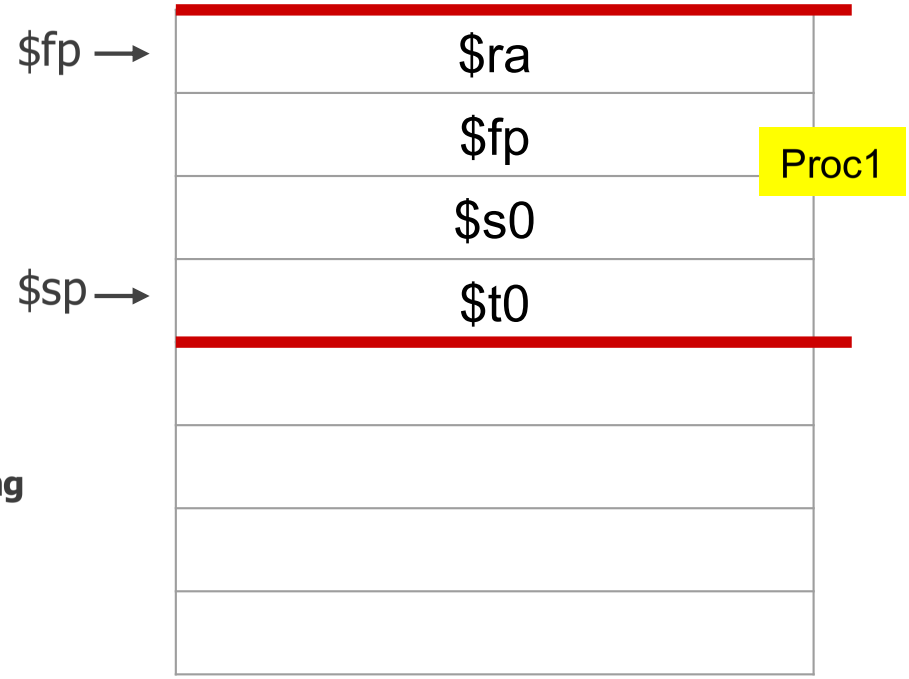
1. Who is the CALLER and CALLEE?
2. What registers need to survive the procedure call?
3. Contractually, who is responsible for which registers?
4. Who needs to create a stack frame?
5. What register values need to be stored in each stack frame?

```
proc1:
        addi $t0, $0, 10
        addi $s0, $0, 5
        add $a0, $t0, $s0
        jal proc2
        add $t1, $v0, $s0
        add $t2, $t0, $v0
        jr $ra
proc2:
        sll $s0, $a0, 2
        add $v0, $0, $s0
        jr $ra
```

Proc1 is both a CALLER and CALLEE!

1. **Contractually**, CALLEE required to store $fp and $ra register CALLER (main) values, and $s register values its in stack frame
2. **Contractually**, CALLER is required to store $t0 in its stack frame (survive the procedure call)



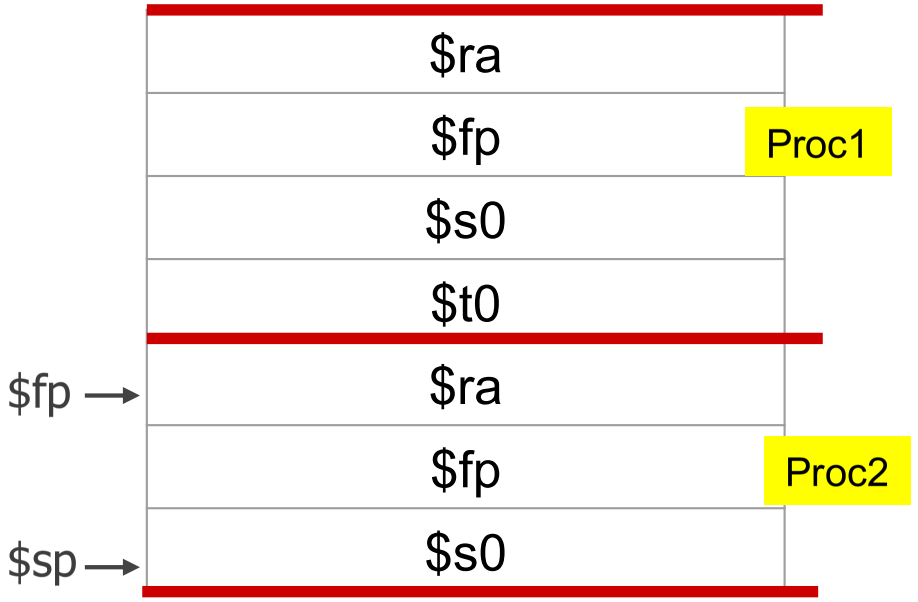


Proc2 is only a CALLEE

1. **Contractually**, CALLEE required to store $fp and $ra register CALLER (proc1) values, and $s register values in its stack frame.
2. Is a leaf, i.e., does not call another procedure.

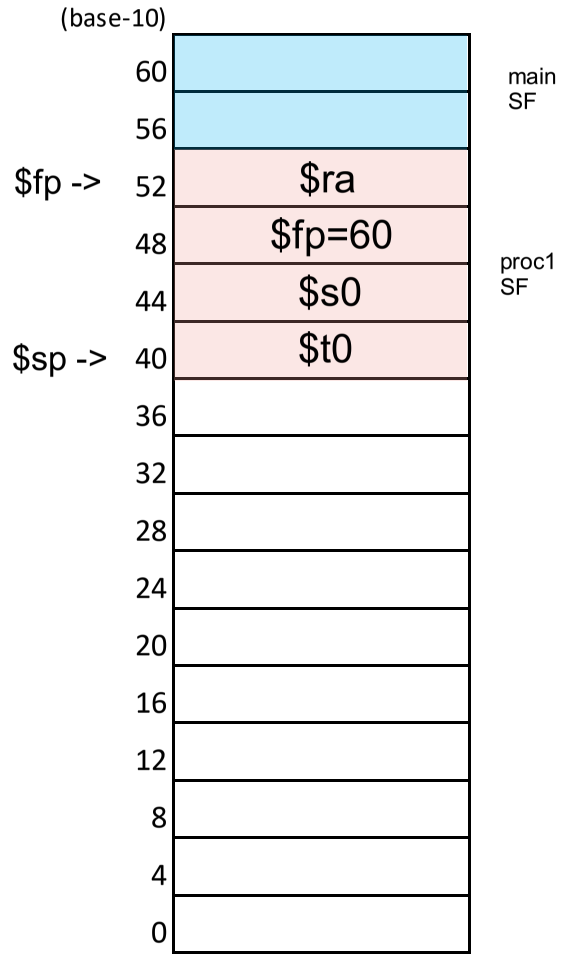
Proc1 and Proc2: Stack Frames

Stack after proc2 is called by proc1, but before proc2 returns.



Update the proc1 MIPS code

```
proc1:
        addi $sp, $sp, -8       # allocate memory for $ra and $fp

        sw $ra, 4($sp)          # save $ra
        sw $fp, 0($sp)          # save $fp
        addi $fp, $sp, 4        # set $fp

        addi $sp, $sp, -8       # allocate memory for $s0 and $t0
        sw $s0, 4($sp)          # save $s0

        addi $t0, $0, 10
        sw $t0, 0($sp)          # save $t0

        addi $s0, $0, 5
        add $a0, $t0, $s0
        jal proc2

        lw $t0, -12($fp)        # restore $t0
        add $t1, $v0, $s0
        lw $s0, -8($fp)         # restore $s0
        add $t2, $t0, $v0

        addi $sp, $fp, 4        # Restore $sp
        lw $ra, 0($fp)          # Restore $ra
        lw $fp, -4($fp)         # Restore $fp

        jr $ra
```



Update the proc2 MIPS code

```
proc2:
        addi $sp, $sp, -8       # allocate memory for $ra and $fp

        sw $ra, 4($sp)          # save $ra
        sw $fp, 0($sp)          # save $fp
        addi $fp, $sp, 4        # set $fp

        addi $sp, $sp, -4       # allocate memory for $s0 and $t0
        sw $s0, 0($sp)          # save $s0

        sll $s0, $a0, 2
        add $v0, $0, $s0

        lw $s0, -8($fp)         # restore $s0
        addi $sp, $fp, 4        # restore $sp
        lw $ra, 0($fp)          # restore $ra
        lw $fp, -4($fp)         # restore $fp

        jr $ra
```