

1 Discussion Pre-Check

- 1.1 After calling a function and having that function return, the **t** registers may have been changed during the execution of the function, while **a** registers cannot.

False. **a0** and **a1** registers are often used to store the return value from a function, so the function can set their values to its return values before returning.

- 1.2 In order to use the saved registers (**s0-s11**) in a function, we must store their values before using them and restore their values before returning.

True. The saved registers are callee-saved, so we must save and restore them at the beginning and end of functions. This is frequently done in organized blocks of code called the “function prologue” and “function epilogue.”

- 1.3 The stack should only be manipulated at the beginning and end of functions, where the callee-saved registers are temporarily saved.

False. While it is a good idea to create a separate ‘prologue’ and ‘epilogue’ to save callee registers onto the stack, the stack is mutable anywhere in the function. A good example is if you want to preserve the current value of a temporary register, you can decrement the **sp** to save the register onto the stack right before a function call.

2 Calling Conventions

Let’s review what special meaning we assign to each type of register in RISC-V.

Register	Convention	Saver
x0	Stores zero	N/A
sp	Stores the stack pointer	Callee
ra	Stores the return address	Caller
a0 - a7	Stores arguments and return values	Caller
t0 - t6	Stores temporary values that <i>do not persist</i> after function calls	Caller
s0 - s11	Stores saved values that <i>persist</i> after function calls	Callee

To save and recall values in registers, we use the **sw** and **lw** instructions to save and load words to and from memory, and we typically organize our functions as follows:

Prologue

addi sp, sp, -8 # Room for two registers. (Why?)

2 *Precheck: RISC-V Calling Convention*

```
sw s0, 0(sp)    # Save s0 (or any saved register)
sw s1, 4(sp)    # Save s1 (or any saved register)

# Code omitted

# Epilogue
lw s0, 0(sp)    # Load s0 (or any saved register)
lw s1, 4(sp)    # Load s1 (or any saved register)
addi sp, sp, 8  # Restore the stack pointer
```

3 Calling Conventions in Code Example

Below is an example of calling conventions in a RISC-V function.

The callee-saved registers (like s0) are saved at the start of the function and restored before returning, as these registers must be preserved by the function.

The caller-saved registers (like t1 and ra) are saved by the caller before invoking another function, as the callee can modify these registers. **Note:** Although ra is a caller-saved register, it is usually saved at the very beginning and end of the function by convention, as shown below.

```
func_a:
    # Prologue: Save callee-saved registers & the return address
    addi sp, sp, -8 # Allocate stack space
    sw ra, 0(sp)    # Save return address
    sw s0, 4(sp)    # Save s0

    addi t1, x0, 10 # Modify t1
    addi s0, x0, 20 # Modify s0

    # Save caller-saved registers before function call
    addi sp, sp, -4 # Allocate more stack space
    sw t1, 0(sp)    # Save t1 (caller-saved register)

    jal func_b      # Call another function

    # Restore caller-saved registers after function call
    lw t1, 0(sp)    # Restore t1 (caller-saved register)
    addi sp, sp, 4  # Deallocate space for caller-saved register
```

```
addi t1, t1, 5    # Modify t1
addi s0, s0, 5    # Modify s0

# Epilogue: Restore callee-saved registers
lw ra, 0(sp)      # Restore return address
lw s0, 4(sp)      # Restore s0
addi sp, sp, 8     # Deallocate stack space

ret               # Return from func_a
```