Real-World Linkage Convention (using a real stack frame)

A real-world linkage convention puts parameters onto the stack into a stack frame, and allows the subroutine to return a structured type (a struct or class object). There may be variations from this plan (e.g., functions with a variable number of parameters), but this holds all of the key components of a real stack frame calling convention in a compiled language like C. This convention, however, does not really deal with floating-point registers.

Caller prolog (done by the caller):

1. Push any registers $t0-$t9 that contain values that must be saved. Push the registers in ascending numerical order.
2. **Stack frame begins here.** If returning a structure (a struct or class object) from the subroutine, allocate space for the structure, $sp = $sp - space_for_structure.
3. Allocate space for parameters, $sp = $sp - space_for_parameters, and put argument values into parameters. Subtraction from $sp may be combined with subtraction from step 2. DO NOT pass arguments in the $a registers.
4. Call the subroutine using jal.

Subroutine Prolog (done by the subroutine):

5. Push $ra (always).
6. Push the caller's frame pointer $fp (always).
7. Push any registers $s0-$s7 that the subroutine might alter. Also push any registers $a0-$a3 you might alter. Push the registers in ascending numerical order.
8. Initialize the frame pointer: $fp = $sp - space_for_variables. The "space for variables" is normally four times the number of local (scalar) variables. (Remember that subtracting from $sp grows the stack). **Stack frame ends here, with $fp pointing to bottom of stack frame.** If no local variables, set $fp = $sp and skip the next step.
9. If not the same already, initialize the stack pointer: $sp = $fp.

Subroutine Body:

10. The subroutine may alter any $t register, or any $s or $a register that it saved in the subroutine prolog.
11. The subroutine refers to structure return area, parameters and local variables using `disp($fp)`.
12. The subroutine may push and pop temporary variables and other values on the stack using $sp.
13. If the subroutine calls another subroutine, then it does so by following these rules.

Subroutine Epilog (done at the end of the subroutine):

14. If returning a structure, copy it into the structure return area, otherwise, put return value in $v0-$v1 (a floating-point value may be returned in $f0-$f1) – DO NOT return a scalar on the stack.
15. $sp = $fp + space_for_variables.
16. Pop into $a and $s registers any values were previously saved in the stack frame, in reverse order.
17. Pop the caller's frame pointer into $fp (always).
18. Pop $ra (always).
19. Return to the caller using jr $ra.

Regaining Control from a Subroutine (caller epilog):

20. Deallocate space for parameters, $sp = $sp + space_for_parameters.
21. If returning a structure, save to destination and then reclaim space from stack, $sp = $sp + space_for_structure. Or, returned structure may be used as a temporary variable, and de-allocated after use.
22. Pop any registers $t0-$t9 that the caller previously pushed, in reverse order.

The stack frame during the subroutine call looks like this: