Calling Convention

What is a calling convention?

- Procedure that outlines how functions and subroutines interact with each other
  - Passing parameters
  - Return result
  - Making sure no values are lost
- Responsible for
  - Transfer of values
  - Saving and restoring state
  - Distinct stack frames

States (for LC-2200)

- State: the contents of the registers at a specific point in time
- Why is it important?
  - Program uses both memory and the registers in execution
  - States need to be saved to ensure consistency before and after a subroutine call
  - We can specialize registers to make our lives easier!

Specialized Registers

- $s0 - $s2 are caller saved registers
- $t0 - $t2 are temporary registers
- $a0 - $a2 are param passing registers
- $v0 is a return value register
- $ra is a return address/link register
- $at is a target address register
- $sp is a stack pointer register
- $fp is a frame pointer register
- $k0 is reserved for OS use and traps (not for user space!)
- $zero is the zero register and has a fixed value

Saving State?

- Option 1: Shadow Registers
  - Multiple sets of registers
  - Switch to a different set on a subroutine call using hardware
- Option 2: Stack

- Saving and restoring registers in memory
- Grows down (high address => low address)

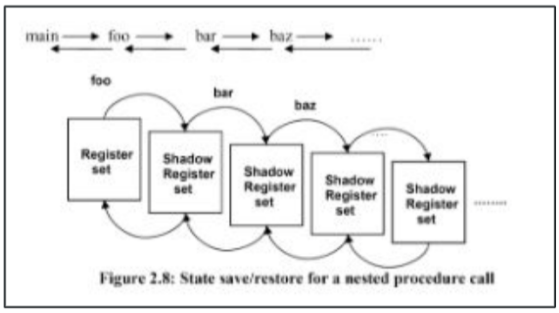
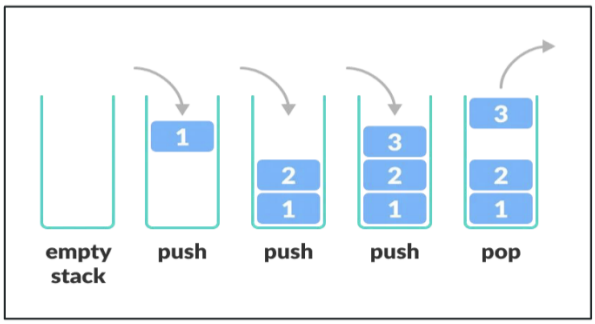


Figure 2.8: State save/restore for a nested procedure call



Shadow Register	Stack
Multiple sets of registers, Switch to a different set on a subroutine call using hardware	Saving and restoring registers in memory, Grows down (high address → low address)
Very fast	No additional hardware required, flexible
Complicates hardware, problems with recursion (unlimited register sets?)	Can be slow accessing memory, more complicated to write

Activation Register

- The part of the stack relevant to the currently executing procedure
- Allows for communication between caller and callee (if we have a convention for that!)
- There are be multiple activation records on the stack (nested function calls), but only one is active at a time (the current function)
- Sometimes called a frame too

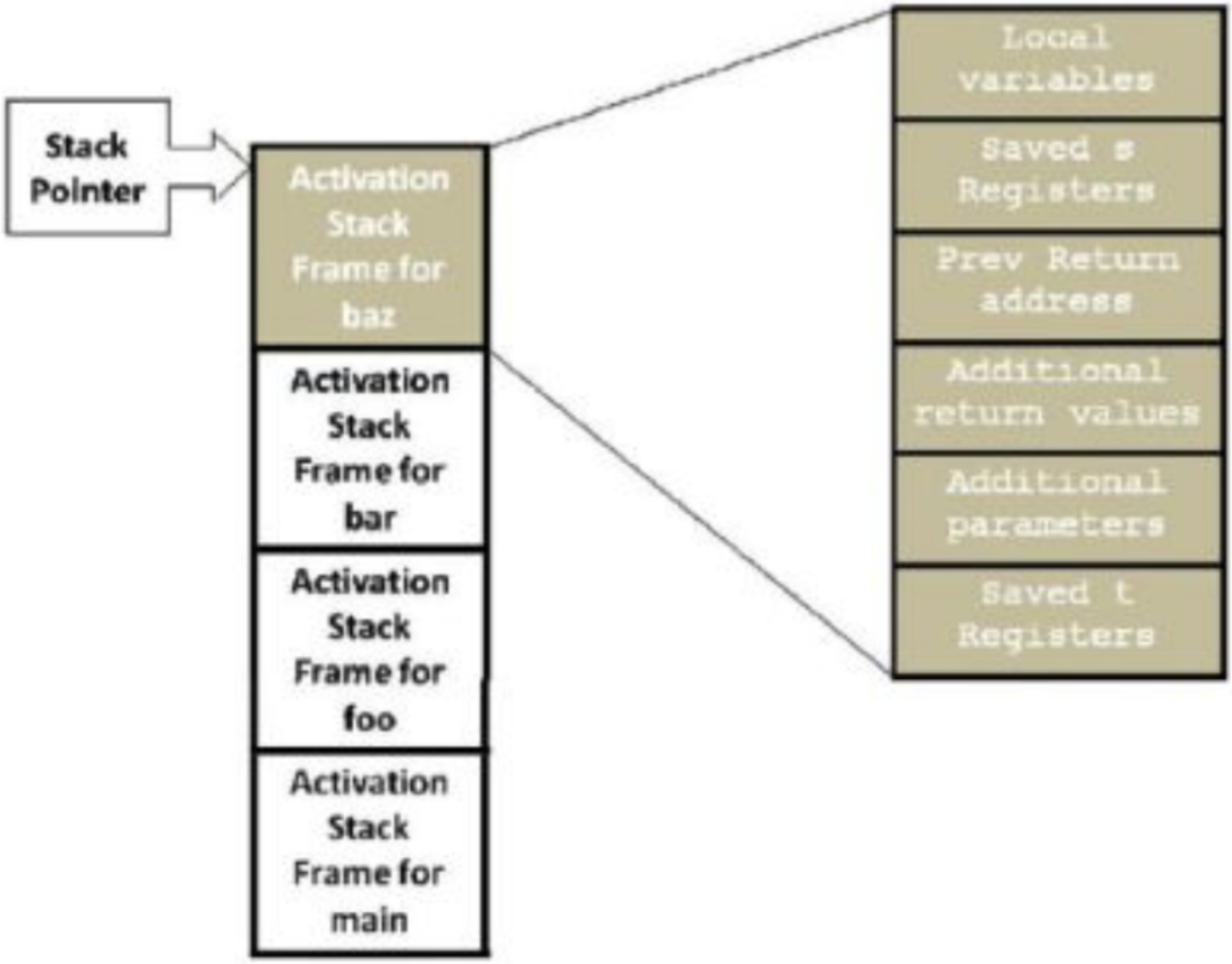


Figure 2.21: Activation records for a sequence of calls

Frame Pointer

- Constant, nonmoving reference in our activation record
  - Stack pointer moves whenever you push to or pop from the stack, so that's not going to work
- Point to the previous frame pointer on the stack, as saved by the callee
- Store current frame pointer in the $fp register in LC-2200

Caller/Callee Division of Responsibilities

Caller

- $t0 – $t2: temp registers
- $ra: return address

Callee

- $s0 – $s2: saved registers
- $fp: frame pointer

Stack Buildup and Teardown

1. Caller saves any of the registers t0-t2 on the stack (if it needs the values in them upon return)
2. Caller places the function parameters in registers a0-a2. If there are additional parameters, the caller puts them on the stack.
3. Caller allocates space for any additional return values on the stack. This is only done if the return value cannot fit in v0.
4. Caller saves previous return address stored in register `ra` onto the stack.
5. Caller executes `JALR $at`, `$ra`. This has no effect on the stack
6. Callee saves the previous frame pointer on stack. Then it copies the current value of the stack pointer (`$sp`) into the frame pointer (`$fp`)
7. Callee saves any of the `s0`-`s2` registers it plans to overwrite onto the stack. For simplicity, we can say that it always saves `s0`-`s2` onto the stack.
8. Callee allocates space on the stack for any local variables. This is the reason the frame pointer is important. At this point, Callee does all its stuff, and stores the return values as necessary
9. Prior to return, Callee restores the `s0`-`s2` registers with the values on the stack
10. Prior to return, Callee restores the old frame pointer from the stack.
11. Callee executes jump to return address: `JALR $ra`, `$zero`. No change to the stack
12. Upon return, Caller restores the previous return address to `ra`
13. Upon return, Caller stores the additional values as desired
14. Upon return, Caller moves stack pointer to discard additional parameters