

Call/Return Protocols

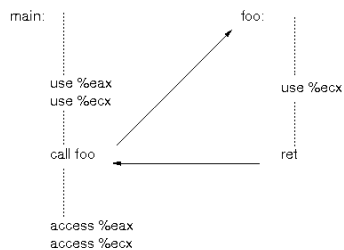
CSE 361S

Registers

- observe: register set used for efficient access to temp./intermediate data within programs
- ⇒ used extensively

Issue

- Registers are global, software development is local



- Value in %ecx is set by foo, not main, but developer of main might not know this.

One *bad* solution

- every subroutine documents its register usage
- let's count the ways this is bad!
 - # subroutines × # calls to subroutines = # of ways to goof

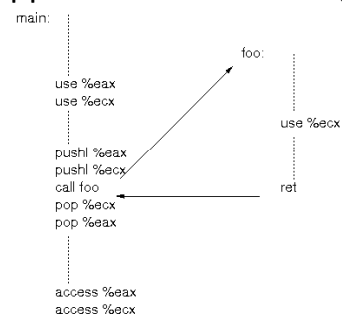
Aspect of good solution

- Like to maintain register state in calling routine so registers can be effectively used

⇒ need to save registers if they are to be used in callee routine and caller routine

- In earlier example, caller is main and callee is foo
- The stack is traditionally used for this purpose, but who is responsible for saving?

One approach: caller save registers

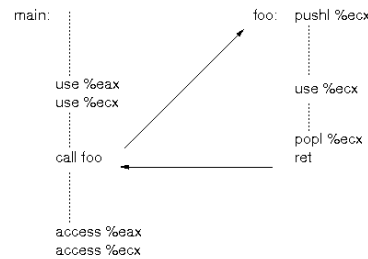


- Value in %ecx is set in main, not in foo

Caller save protocol

- Only save what needs to be kept unaltered ++
- Don't know if callee might alter, so save things that might be unnecessary --

2nd approach: callee save registers



- only save what is altered ++
- don't know if save value is live --

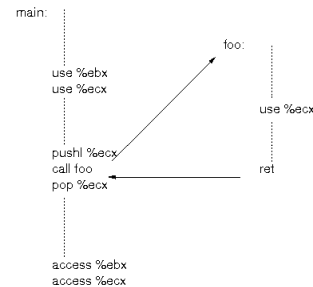
Actual x86 convention

- Mixture of callee and caller save:

caller save %eax, %ecx, and %edx
 callee save %ebx, %esi, and %edi

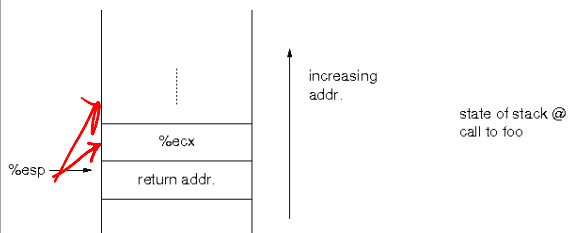
- Perform save only when needed, based on local information

E.g., move %eax usage to %ebx



- caller, callee save status is now one of the many considerations for what registers to use

Stack



Other uses for stack

- Passing parameters to subroutines
- Local variables within subroutines

⇒ motivates stack frame, region of stack associated with a particular invocation of a function/subroutine

- Top of stack always available via %esp
- Stable reference relative to call %ebp called frame pointer

