Laboratory 9 Notes
X86 Stack

Stack Operations

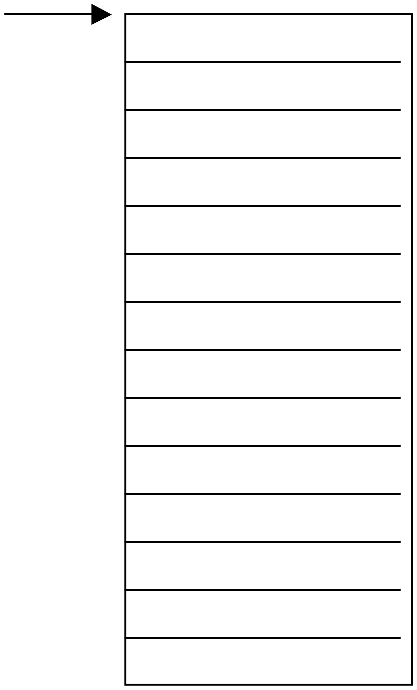
push *src*

1. Make space on the stack by decrementing %rsp (stack pointer).
2. Move *src* to the stack

%rsp ← %rsp - 8
(%rsp) ← src

Initial state of the stack

%rsp=0xfffffffffff8

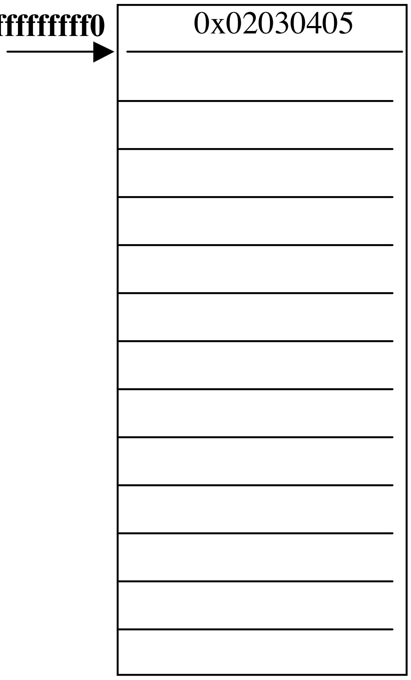


Push a word-size value in %rax on the stack (decrement %rsp and move Src to (%rsp)

(assume %rax = 0x0000000002030405)

Push %rax

%rsp=0x fffffffffff0

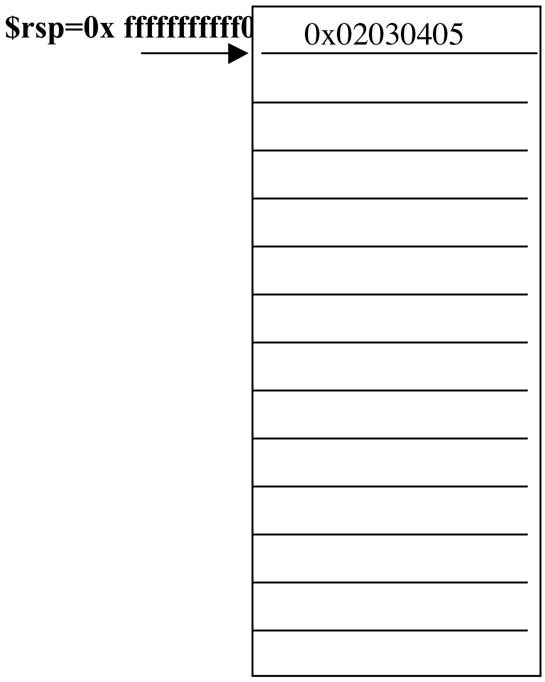


pop *dest*

1. Move contents of top of stack to the *dest*
2. Release space on the stack by incrementing %rsp.

dest ← (%rsp)
%rsp ← %rsp + 8

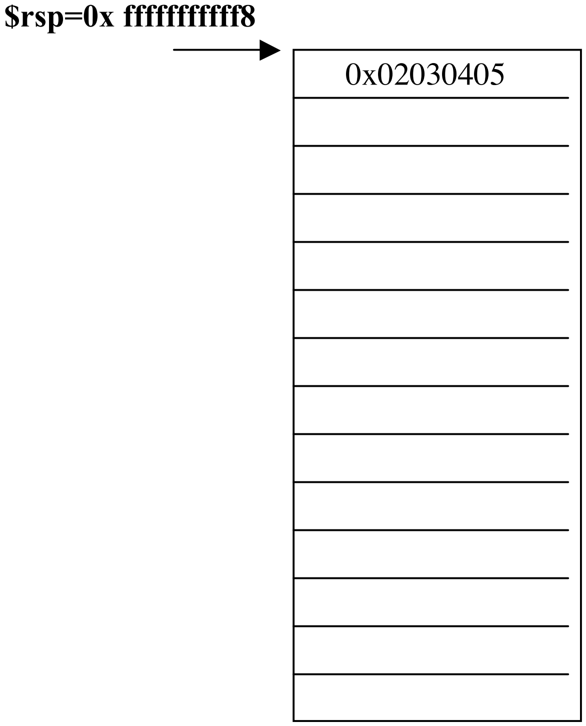
Initial State of Stack



Pop a word-size value from the stack.

Pop %rbx

(%rbx gets 0x0000000002030405)



Instructions used for Function call and return

call *function* 1. Pushes the return address on stack (the address of the instruction *following* the function call)
2. Puts the starting address of the function in %rip:

%rsp ← %rsp - 8
(%rsp) ← %rip (already updated for next instruction)
%rip ← address of function

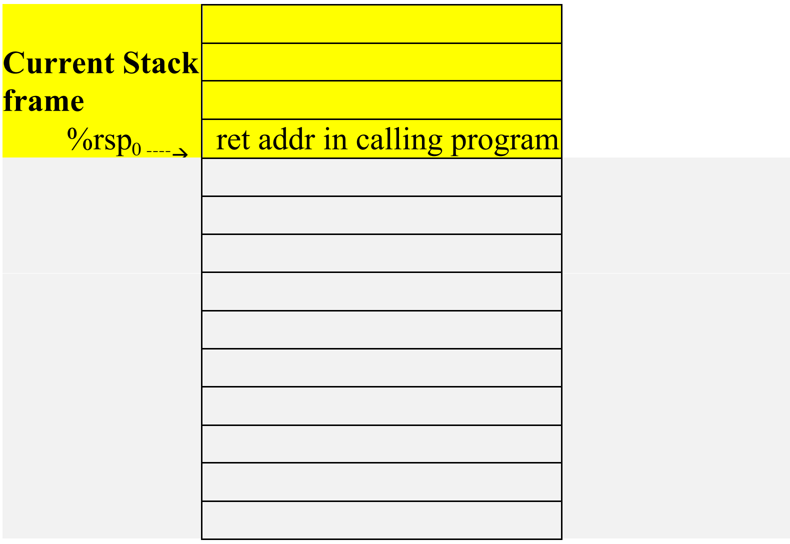
ret 1. Pops the return address off the top of the stack and puts it in %rip (resumes execution of the caller function.

%rip ← (%rsp)
%rsp ← %rsp + 8

Conventions for drawing stack diagrams

To record the contents of the stack to understand how the stack is used, using the following notation:

- We use the model of memory where the stack has low addresses at the bottom and high at the top. Each row in the stack represents a word. The initial **%rsp** with a subscript of **0** is pointing to the top of the current stack frame



- Trace the effect on the stack of executing each instruction in the program by moving the position of the **%rsp** when it changes, (incrementing the subscript for each new value), and by recording new values on the stack as they are stored there.

- When the stack starts to empty, continue with the same notation, except use the right hand side of the stack diagram to indicate the changes.

- Also record changes to relevant registers.