

Laboratory 10 Notes

X86 Stack and Procedures

- Certain instructions implicitly modify the stack pointer (**push**, **pop**, **call**, **ret**)
- `%rsp` (*stack pointer*) always holds a pointer into the current stack frame

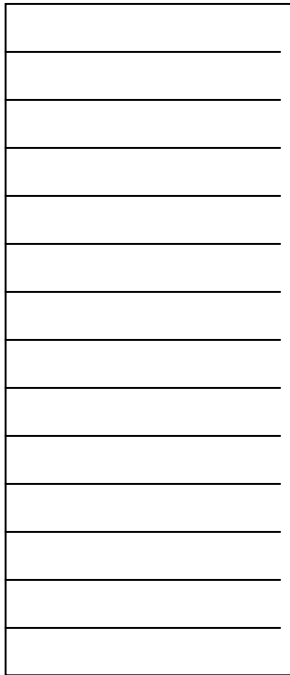
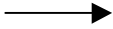
push *src*

1. Make space on the stack by decrementing `%rsp`:
 $\%rsp \leftarrow \%rsp - 8$

2. Move *src* to the stack:
 $(\%rsp) \leftarrow src$

Initial state of the stack

`%rsp=0xfffffffff8`

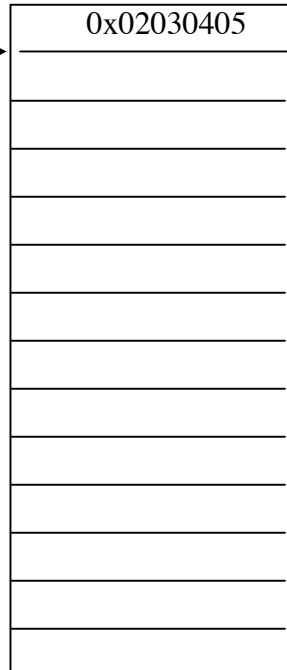
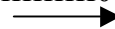


Push a word-size value in `%rax` on the stack
(decrement `%rsp` and move `Src` to `(%rsp)`)

(assume `%rax = 0x000000002030405`)

push `%rax`

`%rsp=0xfffffffff0`



pop *dest*

1. Move contents of top of stack to the *dest*
 $dest \leftarrow (\%rsp)$
2. Release space on the stack by incrementing $\%rsp$.
 $\%rsp \leftarrow \%rsp + 8$

Initial State of Stack

Pop a word-size value from the stack.

Pop $\%rbx$

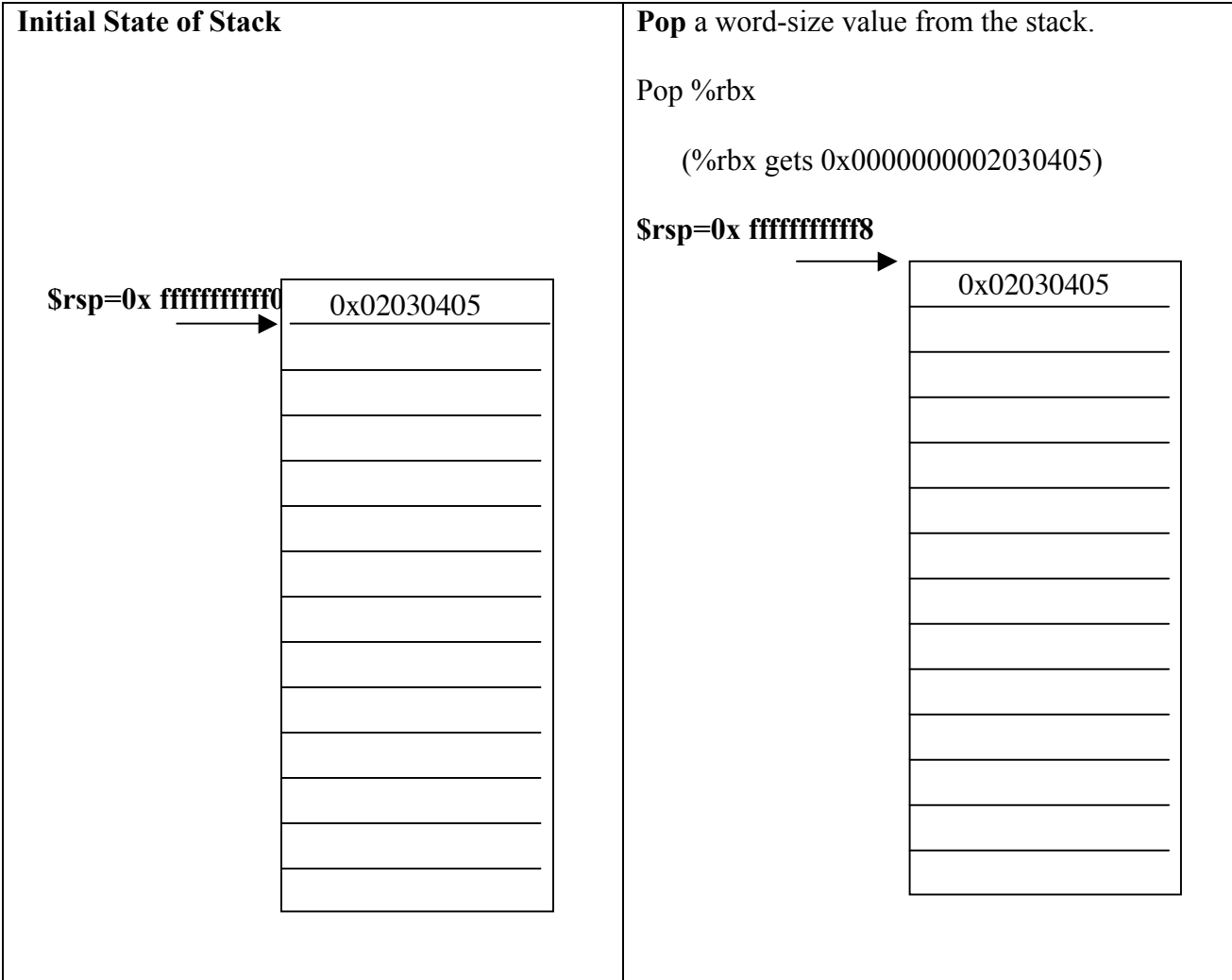
($\%rbx$ gets 0x0000000002030405)

$\$rsp=0x\text{ ffffffff8}$

$\$rsp=0x\text{ ffffffff0}$

0x02030405

0x02030405



call *function* 1. Pushes the *return address* on stack (return address is the address of the instruction *following* the function call)
 $\%rsp \leftarrow \%rsp - 8$
 $(\%rsp) \leftarrow \%rip$ (already updated for next instruction)

2. Puts the starting address of the *function* in $\%rip$:
 $\%rip \leftarrow$ starting address of *function*

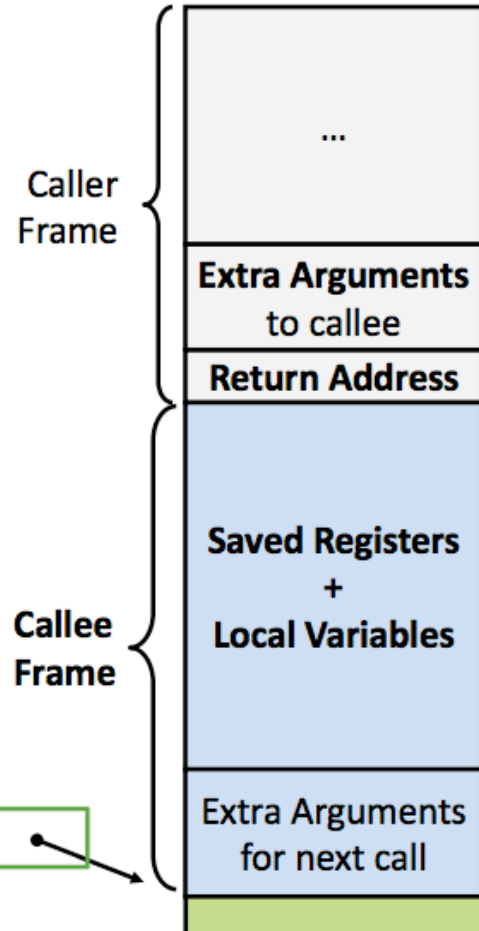
ret 1. Pops the return address from the top of the stack into $\%rip$ (to resume execution of the *calling* function).
 $\%rip \leftarrow (\%rsp)$
 $\%rsp \leftarrow \%rsp + 8$

Procedure Summary

- *call, ret, push, pop*
- Stack discipline fits procedure call / return.*
 - If P calls Q: Q (and calls by Q) returns before P
- Conventions support arbitrary function calls.
 - Register-save conventions.
Stack frame saves extra args or local variables.
Result returned in **%rax**

%rax Return value – Caller saved	%r8 Argument #5 – Caller saved
%rbx Callee saved	%r9 Argument #6 – Caller saved
%rcx Argument #4 – Caller saved	%r10 Caller saved
%rdx Argument #3 – Caller saved	%r11 Caller Saved
%rsi Argument #2 – Caller saved	%r12 Callee saved
%rdi Argument #1 – Caller saved	%r13 Callee saved
%rsp Stack pointer	%r14 Callee saved
%rbp Callee saved	%r15 Callee saved

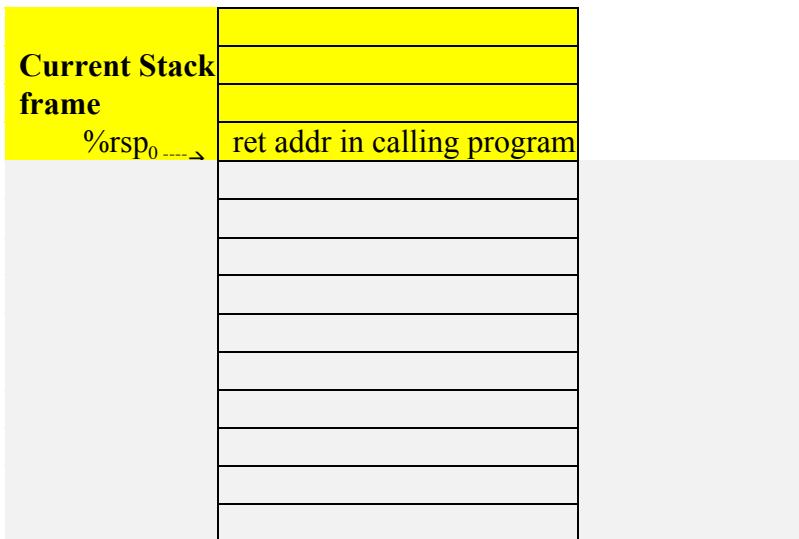
Stack pointer
%rsp



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.