

Laboratory 9 Notes

X86 Stack

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

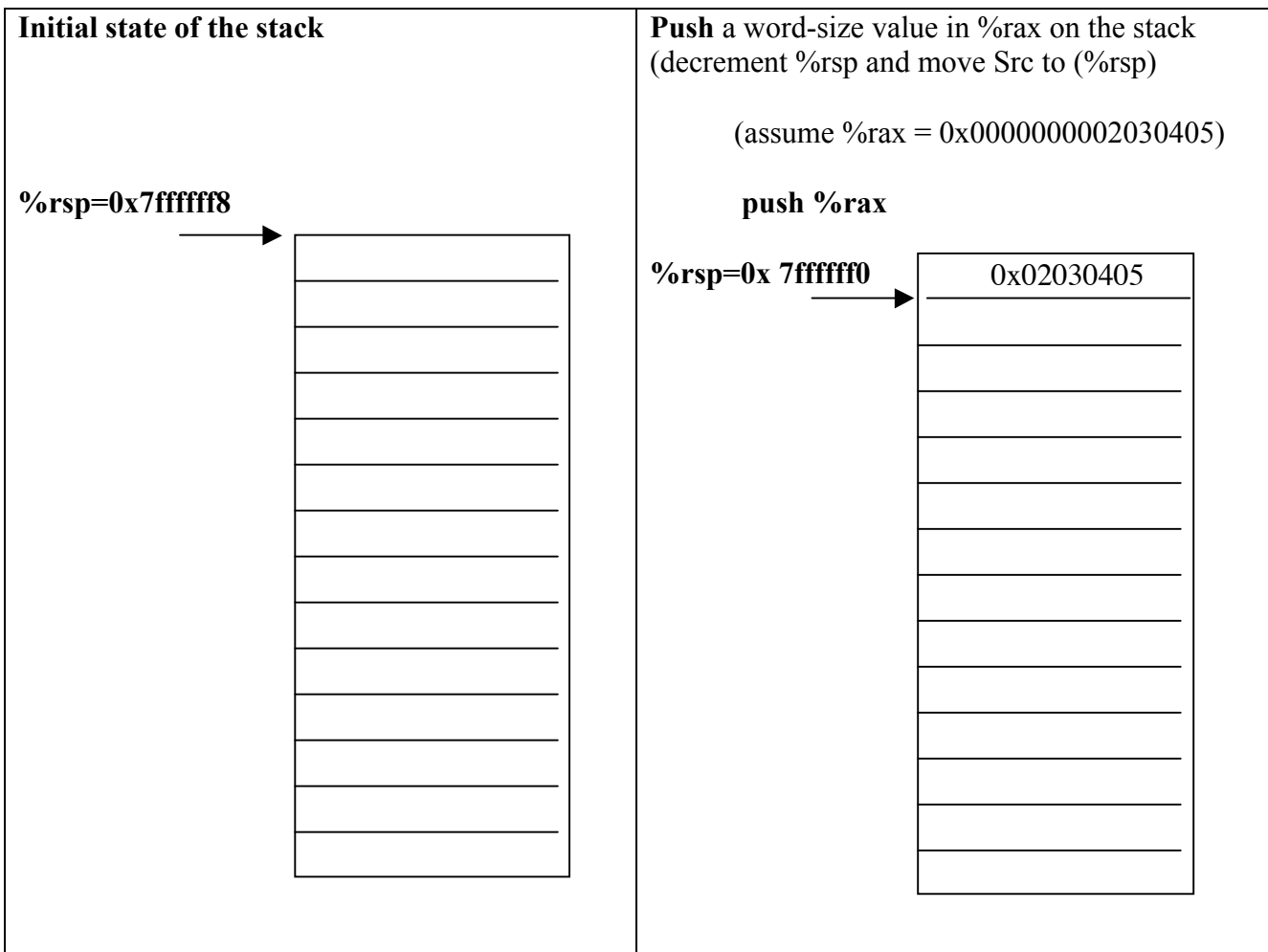
push *src*

1. Make space on the stack by decrementing `%rsp`:

$$\%rsp \leftarrow \%rsp - 8$$

2. Move *src* to the stack:

$$(\%rsp) \leftarrow \text{src}$$



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)

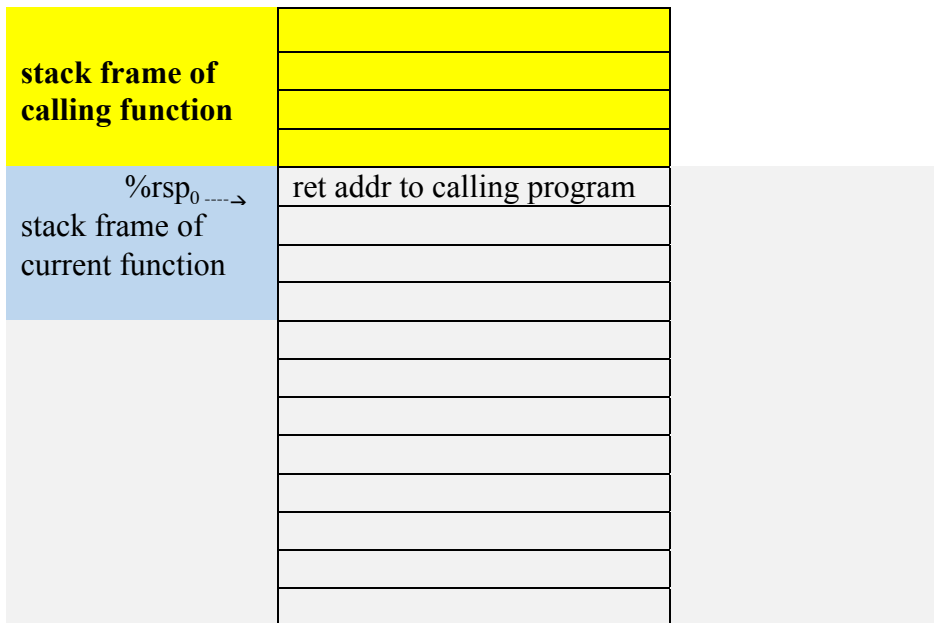
2. Puts the starting address of the *function* in $\%rip$:
 $\%rip \leftarrow$ starting address of *function*

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

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 an 8-byte value. 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.