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

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=0xffffffffffff8

**Push a word-size value in %rax on the stack**

(assume %rax = 0x000000002030405)

**push %rax**

%rsp=0xffffffff0002030405
**pop dest**

1. Move contents of top of stack to the `dest`
   
   \[
   \text{dest} \leftarrow (\%\text{rsp})
   \]

2. Release space on the stack by incrementing `%rsp`.
   
   \[
   \%\text{rsp} \leftarrow \%\text{rsp} + 8
   \]

<table>
<thead>
<tr>
<th>Initial State of Stack</th>
<th>Pop a word-size value from the stack.</th>
<th>Pop %rbx</th>
<th>(%rbx gets 0x0000000002030405)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Srsp=0x ffffffff8</td>
<td>0x02030405</td>
<td>Srsp=0x ffffffff8</td>
<td>0x02030405</td>
</tr>
</tbody>
</table>
call function 1. Pushes the *return address* on stack (return address is the address of the instruction *following* the function call)

\[
\text{%rsp} \leftarrow \text{%rsp} - 8
\]

\[
(\text{%rsp}) \leftarrow \text{%rip} \text{ (already updated for next instruction)}
\]

2. Puts the starting address of the *function* in \text{%rip}:

\[
\text{%rip} \leftarrow \text{starting address of function}
\]

ret 1. Pops the return address from the top of the stack into \text{%rip} (to resume execution of the *calling* function).

\[
\text{%rip} \leftarrow (\text{%rsp})
\]

\[
\text{%rsp} \leftarrow \text{%rsp} + 8
\]
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
%rbx  Callee saved
%rdx  Argument #4 – Caller saved
%rsi  Argument #2 – Caller saved
%rdi  Argument #1 – Caller saved
%rsp  Stack pointer
%rbp  Callee saved
%rbx  Callee saved
%rx8  Argument #5 – Caller saved
%rx9  Argument #6 – Caller saved
%rx10 Caller saved
%rx11 Caller saved
%rx12 Caller saved
%rx13 Caller saved
%rx14 Callee saved
%rx15 Callee saved
```
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 $\%\text{rsp}$ with a subscript of 0 is pointing to the top of the current stack frame.

```
Current Stack frame
\%rsp_0 \rightarrow ret addr in calling program
```

- Trace the effect on the stack of executing each instruction in the program by moving the position of the $\%\text{rsp}$ 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.