# Today
- Stack ops

# Assignments
- HW6 out
The Memory Stack

Stack Basics:

- Like an array-based stack data structure (push and pop operations)
- The “top” of the stack holds last item pushed (pop from the “top”)
- Stack allocated in some region of memory assigned to the program
- Default stack size on ada is:

  \[ 8,192 \text{ Kbytes} = \approx 8 \text{ MB} \]

  can check on (linux-based) system using \texttt{ulimit -s} (or \texttt{ulimit -a})

- Used for (more later):
  - accessing more memory in a program (e.g., “local” variable stack)
  - Used for parameter passing (for functions with large number of params)
pushq src

- src is an immediate, register, or another memory value
- Pushes value in src onto top of stack
- Example:

```assembly
.global _start
.text

_start:  pushq  $97          # push immediate
         movq  $97, %rax
         pushq  %rax         # push register
         pushq  m1           # push memory value

.data
m1:     .quad 97
```

**Note:** 8 bytes pushed onto stack

**Note:** pushw is 16-bit (2-byte) version, where only 2-bytes placed on stack

- generally discouraged to mix pushq and pushw
popq dst

- dst is a register
- Pops 8-bytes off stack and puts into dst
- Example:

  ```
pushq $97
pushq $122
...
popq %rax  # rax = 122
popq %rbx  # rbx = 97
  ```

**Note:** popw is 16-bit (2-byte) version
How the Stack is Implemented

- contiguous sequence of memory cells (like an array ...)
- %rsp holds address of first byte at the “top” of the stack (the stack pointer)
- stack grows “downward” – towards lower addresses

push and pop are equivalent to lower-level instructions

\[
\begin{align*}
\text{pushq } & \$97 \quad \Rightarrow \quad \text{subq } \$8, \%rsp \\
& \text{movq } \$87, (\%rsp) \\
\text{popq } & \%rax \quad \Rightarrow \quad \text{movq } (\%rsp), \%rax \\
& \text{addq } \$8, \%rsp
\end{align*}
\]
Can access items in stack w/out pop (without modifying stack pointer)

- e.g., to access the element after the top element

```
pushq $122
pushq $90
pushq $97

movq 8(%rsp), %rax # get second item from stack
incq %rax
movq %rax, 8(%rsp) # change second item from stack

popq %rax # rax = 97
popq %rax # rax = 91
popq %rax # rax = 122
```

We’ll see more examples of the stack next time ...

- as we move on to function encodings in assembly