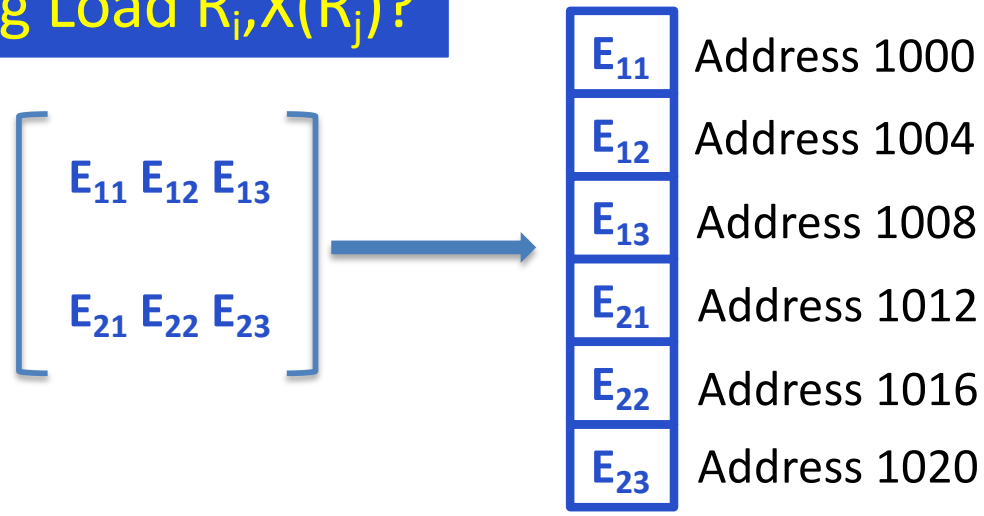
Accessing Memory using Pointers

Move E_{11},E_{12},E_{13} to R1,R2,R3 using Load R_i,X(R_j)?

```
1.   Mov    R5,#0
2.   Load   R1,1000(R5)
3.   Add    R5,R5,#4
4.   Load   R2,1000(R5)
5.   Add    R5,R5,#4
6.   Load   R3,1000(R5)
```

…**Need many Adds to move the pointer**

$$\begin{bmatrix} E_{11} & E_{12} & E_{13} \\ E_{21} & E_{22} & E_{23} \end{bmatrix}$$



A 2 X 3 Matrix is stored in the Memory

What would be a better way to manipulate the PTR?

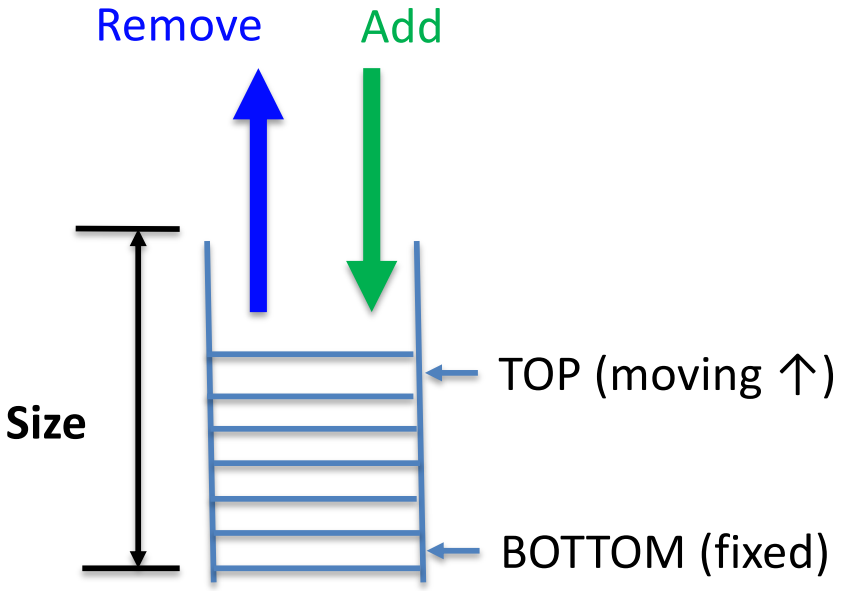
Load register and then increment the PTR as one (pseudo) instruction

Adjust Pointer after Load

- IDEA: Load and Update Register
- LDMIA Rb!, { Rlist }
  - Load the registers specified by Rlist, starting at the base address in Rb. **Write back the new base address.**
- Similar operation:
  - Load only one register
  - Then Rb = Rb + word size

2.6 Stacks

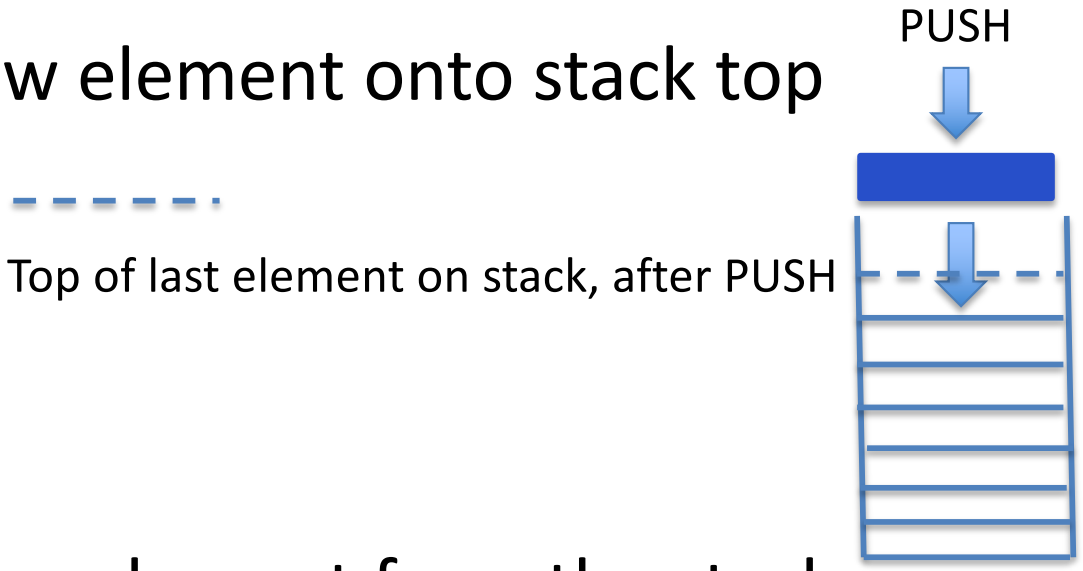
- A **stack**
  - Also called push-down stack
  - Example: a list of data elements
    - Add and Remove at top end only
  - Usage:
    - Easier to store/load data
    - Information passing in Subroutines and Interrupts



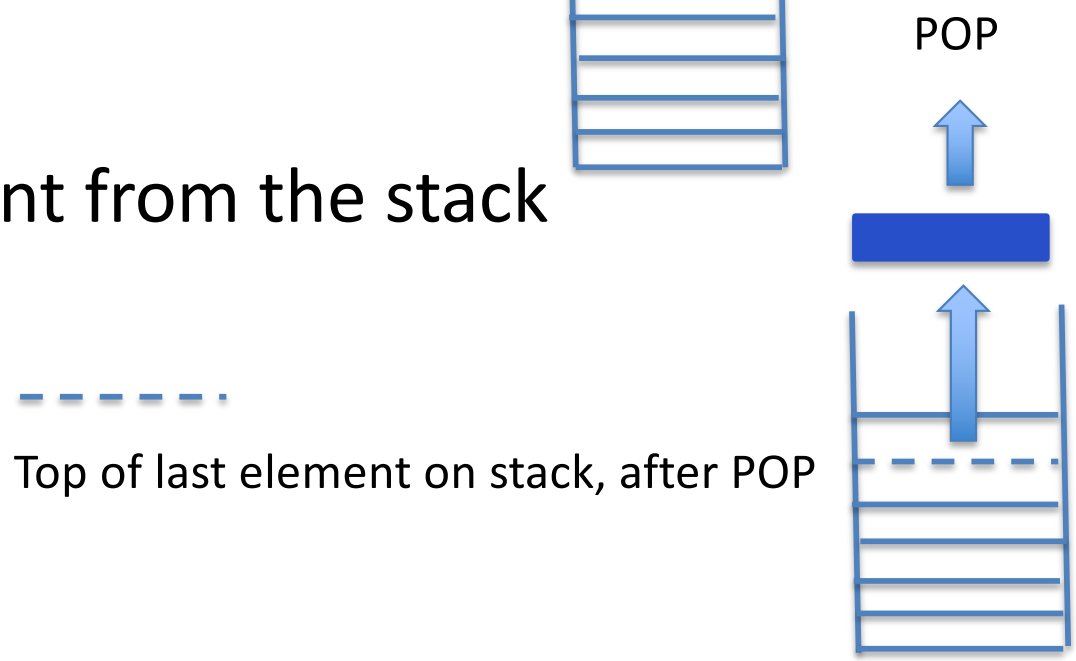
Stack is First-In-First-Out (FIFO) or Last-In-First-Out (LIFO) ?

Stacks: Push and Pop

- **Push** a new element onto stack top

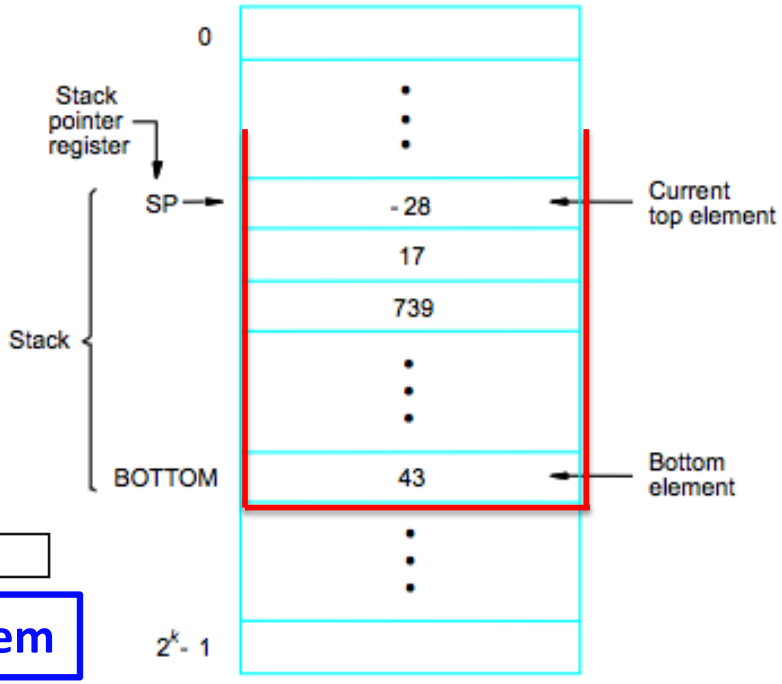


- **Pop** the top element from the stack

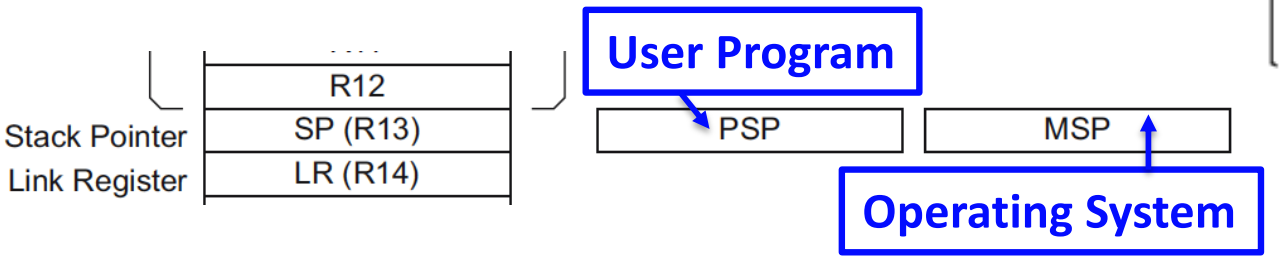


2.7.1 Stack Implementation

- Stack is implemented in memory
  - Note latency accessing memory



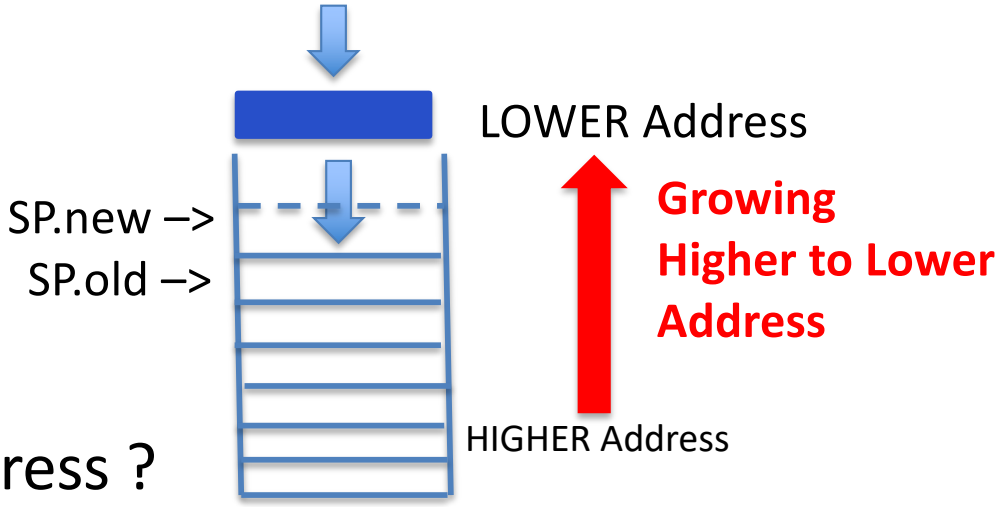
- **Stack pointer (SP)** register



  - Points to top of the stack
    - Contains top-of-stack address

Stack Pushing

- Push operation involves two operations/instructions, for example:
  1. Subtract SP, SP, #4 ; move SP to create space
  2. Store Rj, (SP) ; store Rj in [SP]
- SP.old: original top address
- SP.new: new top address
- PUSH - Stack growth:
  - Lower address to higher address ?
  - Higher address to lower address ?

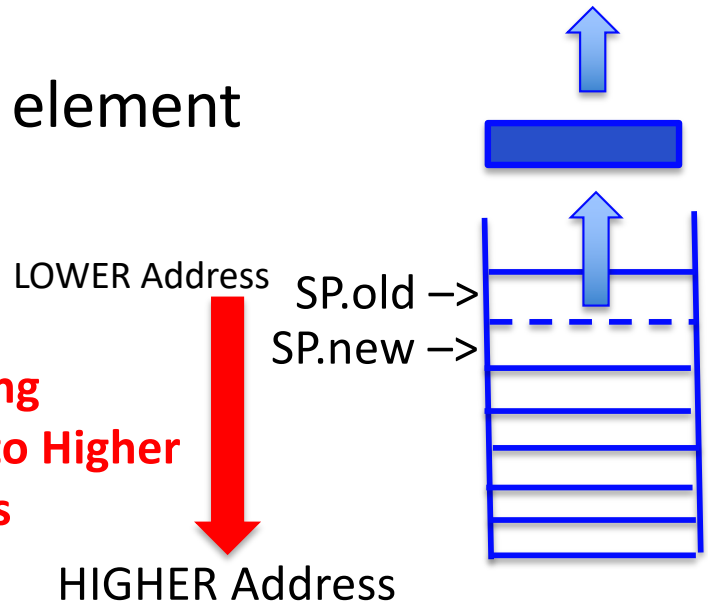


Stack Popping

- Pop operation also involves two operations/instructions, for example:

  1. Load Rj, (SP) ; store Rj in [SP]
  2. Add SP, SP, #4 ; move SP to top element

- SP.old: original top address
- SP.new: new top address



- POP - Stack shrinkage:
  - Lower address to higher address ?
  - Higher address to lower address ?

STM32 Cortex-M0 (PM-11)

Stacks

The processor uses a full descending stack. This means the stack pointer indicates the last stacked item on the stack memory. When the processor pushes a new item onto the stack, it decrements the stack pointer and then writes the item to the new memory location.

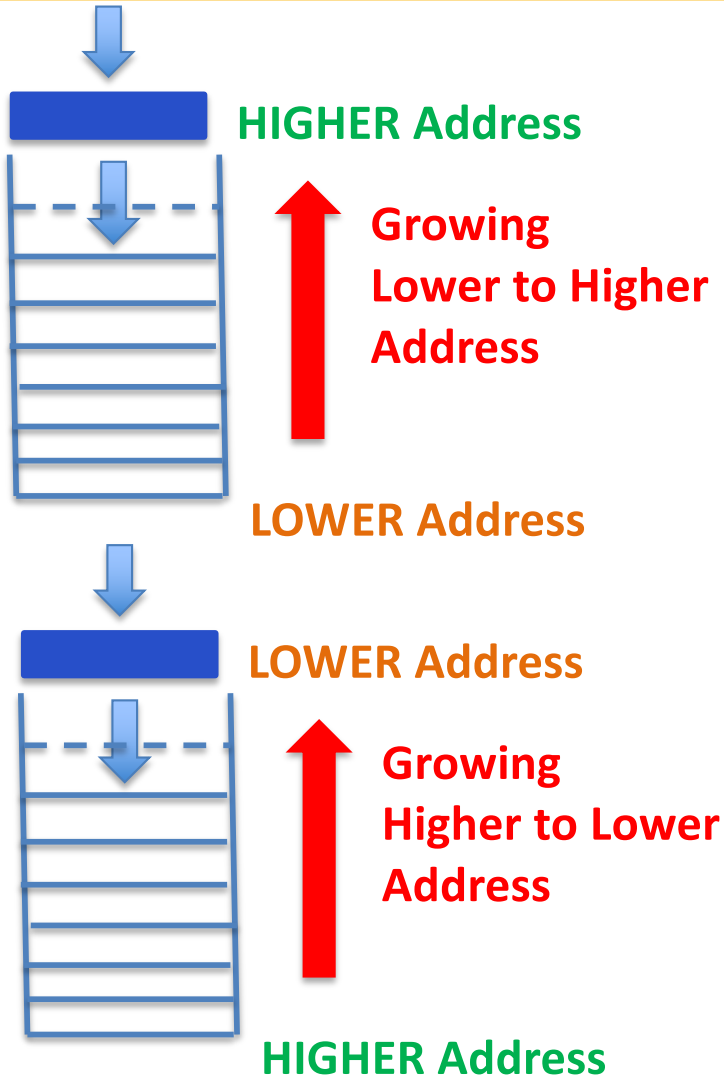
Stack Growth

- **Stack Growth**: two possible directions
  - **Ascending** : lower to higher address

Which one is better ?
Who decides on the direction ?

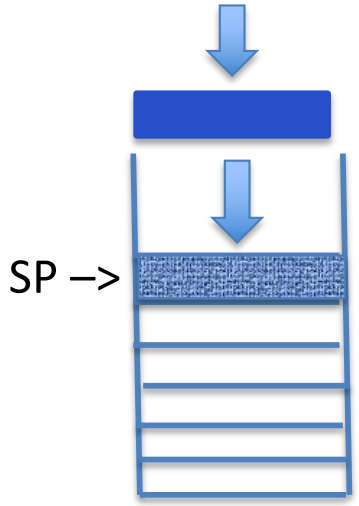
  - **Descending** : higher to lower address

Both work well if the indexing is done correctly
The computer architect makes decision



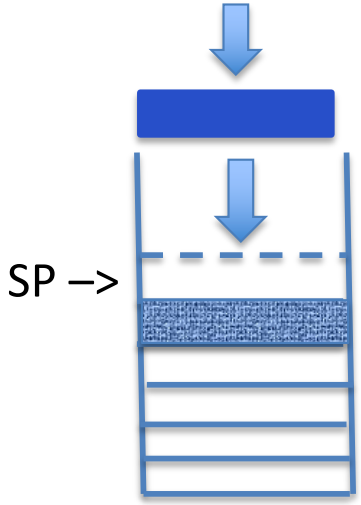
Stack Pointer

- **Stack Pointer**: point to two possible spaces
  - **Full** stack : SP points to the last item pushed (last stacked item)



Which one is better ?
Who decides on the pointer ?

  - **Empty** stack : SP points to the available space



Both work well if the indexing is done correctly
The computer architect makes decision

Four Stack Varieties

- **Stack Growth**: two possible directions
  - **Ascending** : lower to higher address
  - **Descending** : higher to lower address
- **Stack Pointer**: two possible spaces
  - **Full** stack : SP points to the last item pushed (last stacked item)
  - **Empty** stack : SP points to the available space

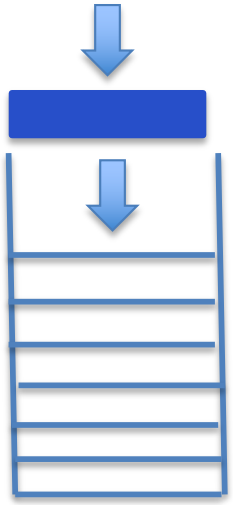
Pre/Post - Inc/Dec Pointers

Pre/Post and Inc/Dec: 4 combinations in using PTR

- **Pre-decrement**: – (SP) : dec. first, then use
- **Pre-increment**: + (SP) : inc. first, then use
- **Post-decrement**: (SP) – : use first, then dec.
- **Post-increment**: (SP) + : use first, then inc.

Given Ascending Full Stack: How to Push? Pop?

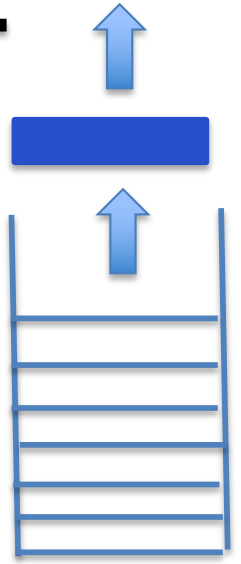
PUSH



- Stack Growth: which?
  - Ascending – lower to higher address
  - Descending – higher to lower address
- Stack pointer: which?
  - Full stack – SP points to last item pushed
  - Empty stack – SP points to the next space

PUSH R1
=
Store R1,+(SP)

POP

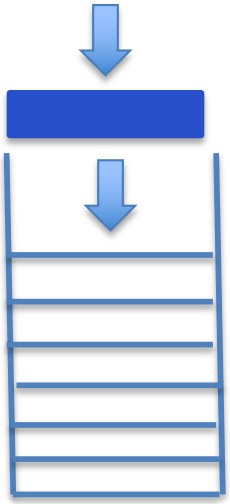


- Stack Growth: which?
  - Ascending – lower to higher address
  - Descending – higher to lower address
- Stack pointer: which?
  - Full stack – SP points to last item pushed
  - Empty stack – SP points to the next space

POP R1
=
Load R1,(SP)-

Given Ascending Empty Stack: How to Push? Pop?

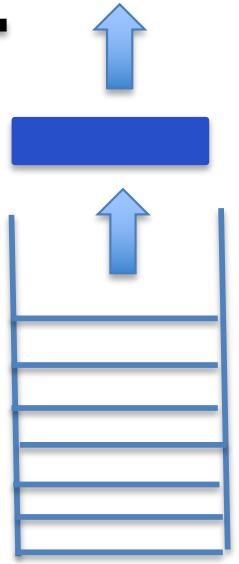
PUSH



- Stack Growth: which?
  - Ascending – lower to higher address
  - Descending – higher to lower address
- Stack pointer: which?
  - Full stack – SP points to last item pushed
  - Empty stack – SP points to the next space

PUSH R1
=
Store R1,(SP)+

POP

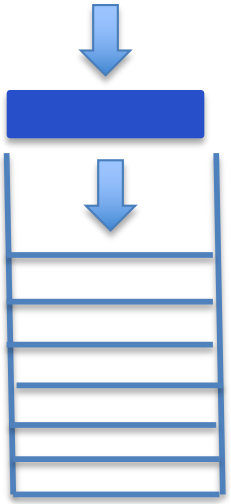


- Stack Growth: which?
  - Ascending – lower to higher address
  - Descending – higher to lower address
- Stack pointer: which?
  - Full stack – SP points to last item pushed
  - Empty stack – SP points to the next space

POP R1
=
Load R1,-(SP)

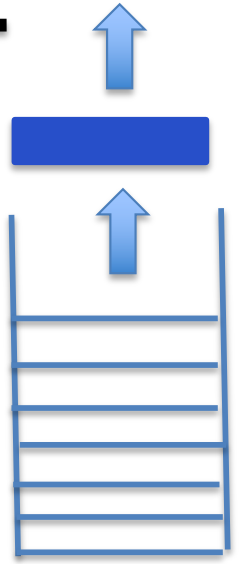
Given Descending Full Stack: How to Push? Pop?

PUSH



- Stack Growth: which?
  - Ascending – lower to higher address
  - Descending – higher to lower address
- Stack pointer: which?
  - Full stack – SP points to last item pushed
  - Empty stack – SP points to the next space

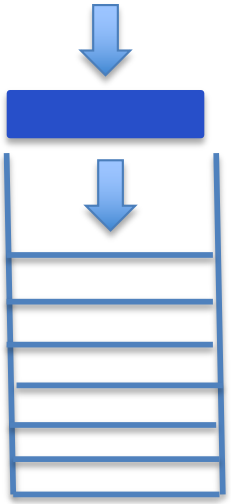
POP



- Stack Growth: which?
  - Ascending – lower to higher address
  - Descending – higher to lower address
- Stack pointer: which?
  - Full stack – SP points to last item pushed
  - Empty stack – SP points to the next space

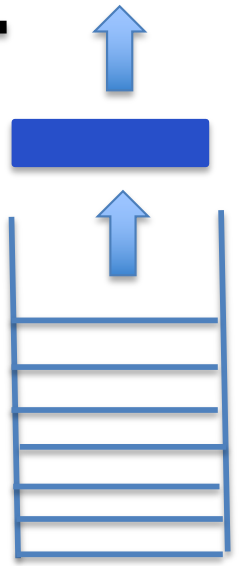
Given Descending Empty Stack: How to Push? Pop?

PUSH



- Stack Growth: which?
  - Ascending – lower to higher address
  - Descending – higher to lower address
- Stack pointer: which?
  - Full stack – SP points to last item pushed
  - Empty stack – SP points to the next space

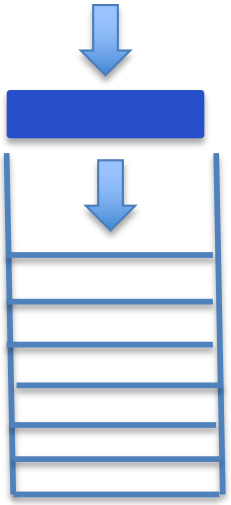
POP



- Stack Growth: which?
  - Ascending – lower to higher address
  - Descending – higher to lower address
- Stack pointer: which?
  - Full stack – SP points to last item pushed
  - Empty stack – SP points to the next space

Given –(SP) in Push/Pop: Stack Type?

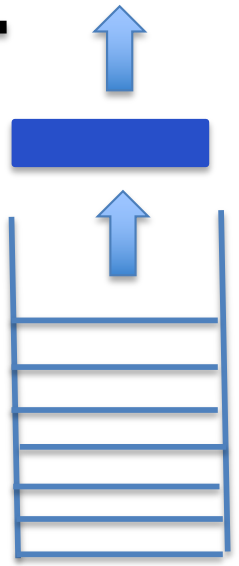
PUSH



- Stack Growth: which?
  - Ascending – lower to higher address
  - Descending – higher to lower address
- Stack pointer: which?
  - Full stack – SP points to last item pushed
  - Empty stack – SP points to the next space

PUSH R1
=
Store R1,-(SP)

POP

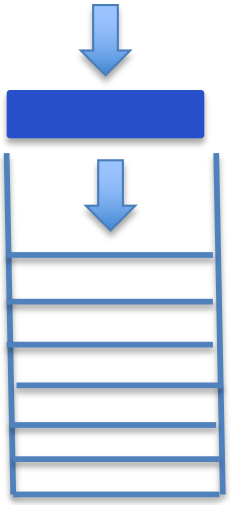


- Stack Growth: which?
  - Ascending – lower to higher address
  - Descending – higher to lower address
- Stack pointer: which?
  - Full stack – SP points to last item pushed
  - Empty stack – SP points to the next space

POP R1
=
Load R1,-(SP)

Given +(SP) in Push/Pop: Stack Type?

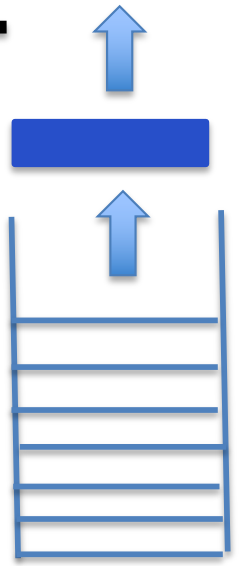
PUSH



- Stack Growth: which?
  - Ascending – lower to higher address
  - Descending – higher to lower address
- Stack pointer: which?
  - Full stack – SP points to last item pushed
  - Empty stack – SP points to the next space

PUSH R1
=
Store R1,+(SP)

POP

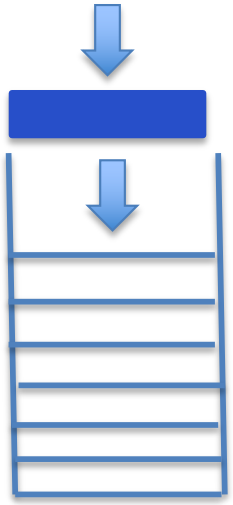


- Stack Growth: which?
  - Ascending – lower to higher address
  - Descending – higher to lower address
- Stack pointer: which?
  - Full stack – SP points to last item pushed
  - Empty stack – SP points to the next space

POP R1
=
Load R1,+(SP)

Given (SP)– in Push/Pop: Stack Type?

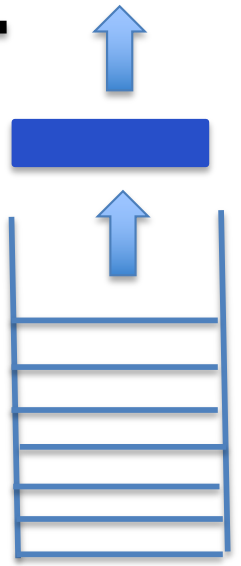
PUSH



- Stack Growth: which?
  - Ascending – lower to higher address
  - Descending – higher to lower address
- Stack pointer: which?
  - Full stack – SP points to last item pushed
  - Empty stack – SP points to the next space

PUSH R1
=
Store R1,(SP)-

POP

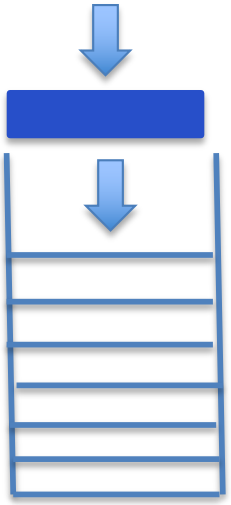


- Stack Growth: which?
  - Ascending – lower to higher address
  - Descending – higher to lower address
- Stack pointer: which?
  - Full stack – SP points to last item pushed
  - Empty stack – SP points to the next space

POP R1
=
Load R1,(SP)-

Given (SP)+ in Push/Pop: Stack Type?

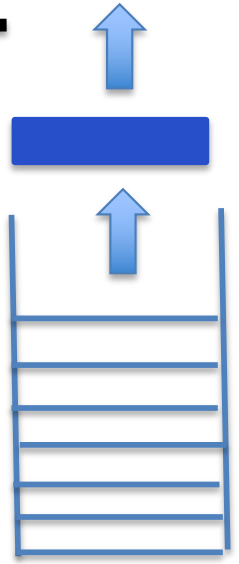
PUSH



- Stack Growth: which?
  - Ascending – lower to higher address
  - Descending – higher to lower address
- Stack pointer: which?
  - Full stack – SP points to last item pushed
  - Empty stack – SP points to the next space

PUSH R1
=
Store R1,(SP)+

POP



- Stack Growth: which?
  - Ascending – lower to higher address
  - Descending – higher to lower address
- Stack pointer: which?
  - Full stack – SP points to last item pushed
  - Empty stack – SP points to the next space

POP R1
=
Load R1,(SP)+