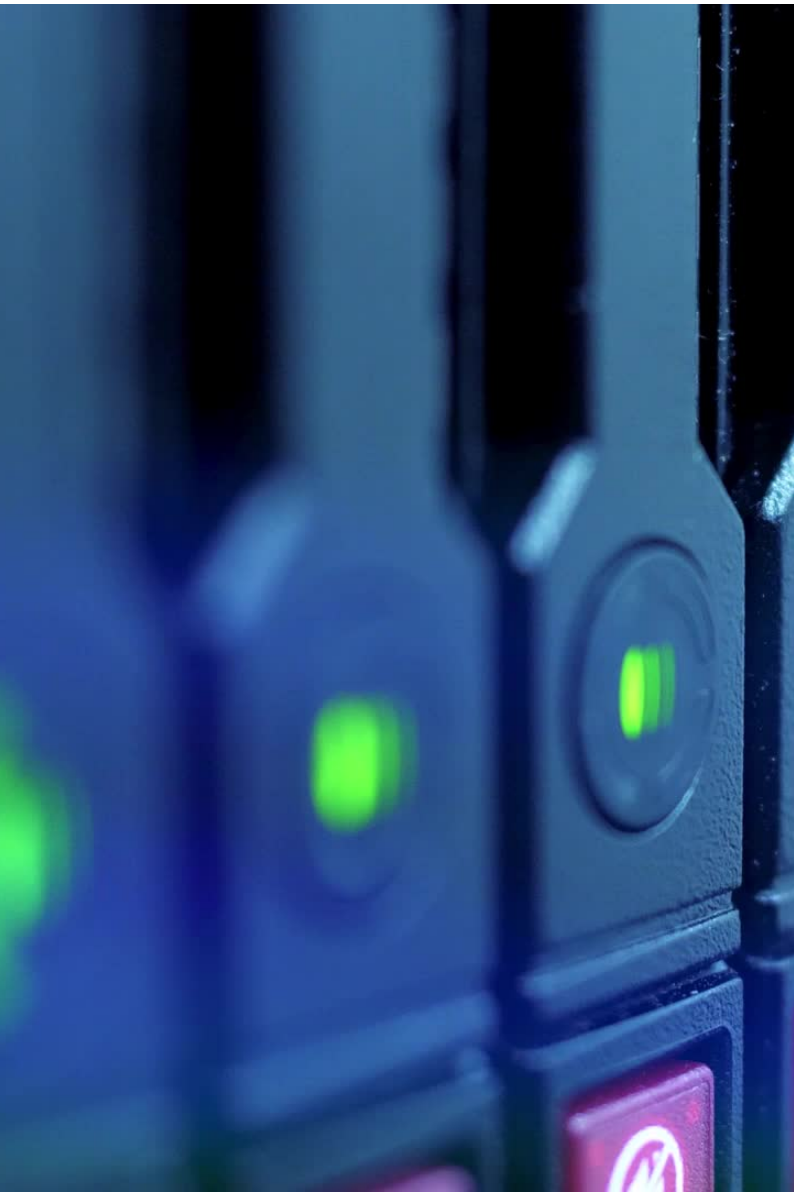
Stack Implementation in Operating Systems

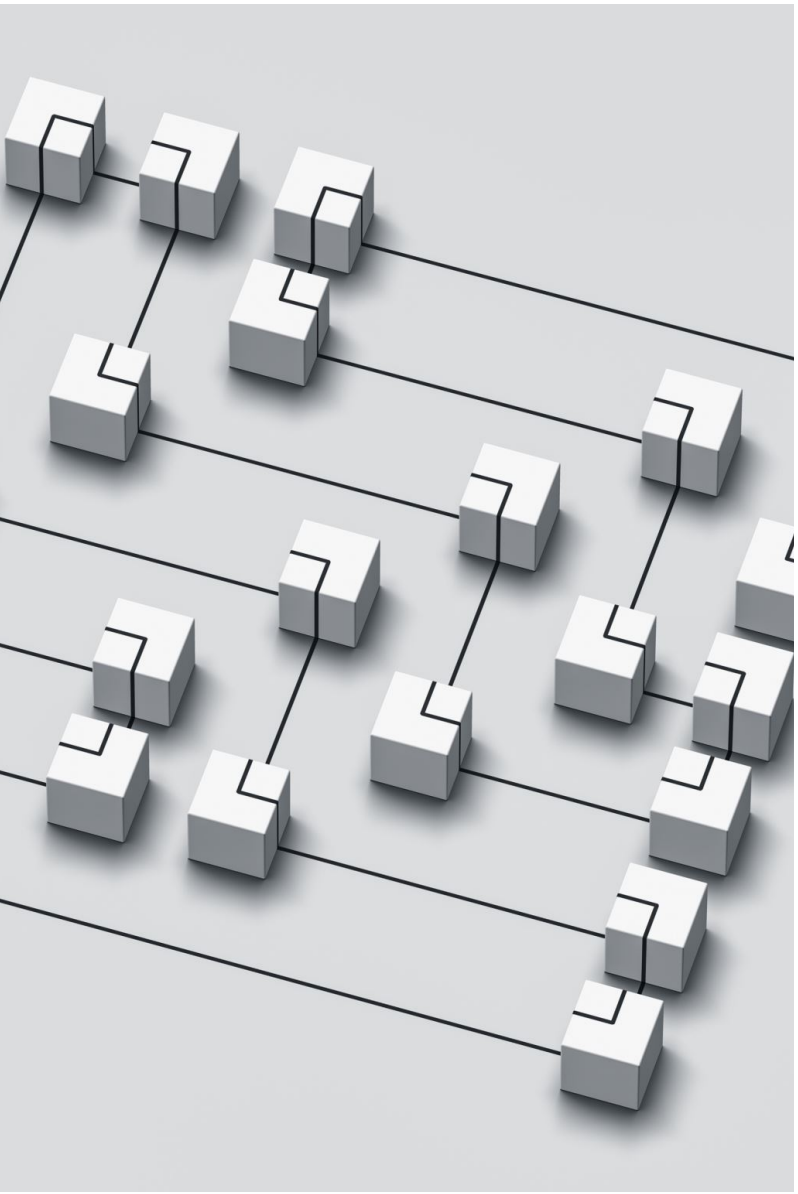


Introduction to Stack

Definition: A stack is an ordered set of components with Last-In-First-Out (LIFO) access.

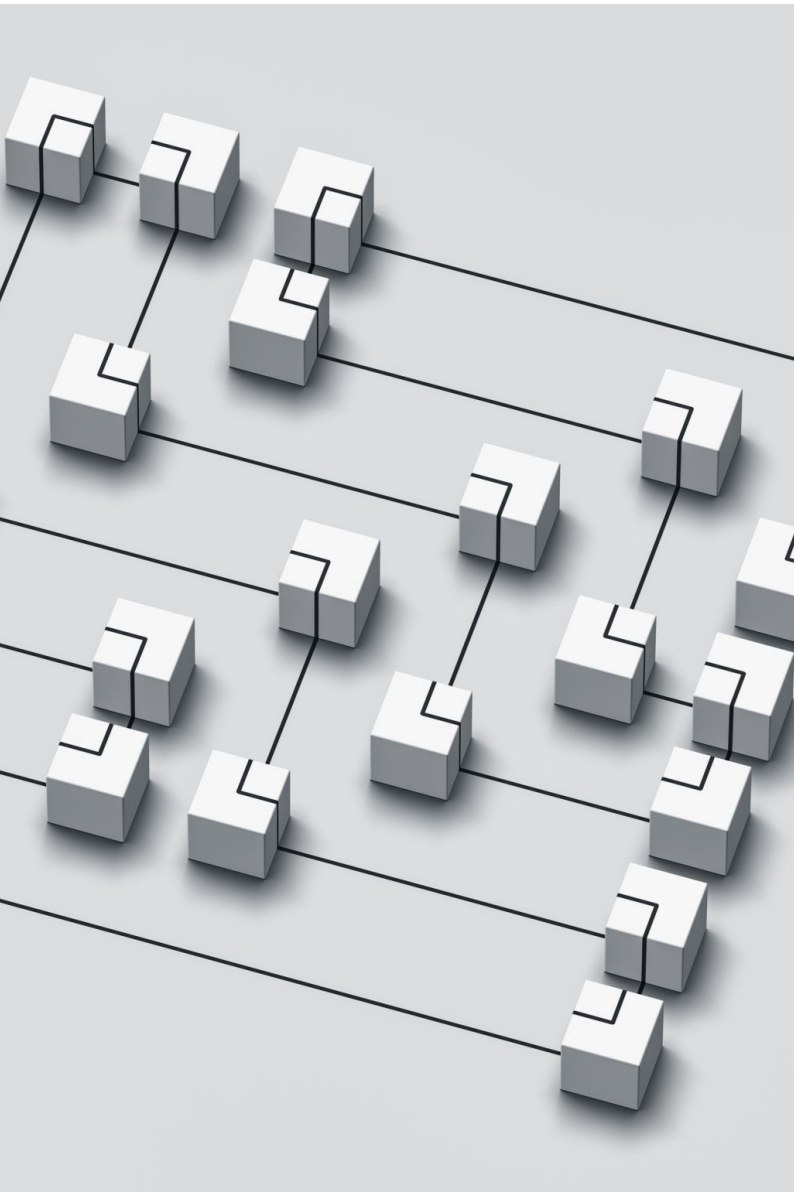
Key Characteristics:

- Only the last added item (top of the stack) can be accessed.
- Items can be added (PUSH) or removed (POP) only from the top.
- Also known as a pushdown list or LIFO list.



Stack Components

- **Addresses Required:**
  - **Stack Pointer:** Contains the address of the top of the stack.
  - **Stack Base:** Contains the address of the lowest location in the reserved block.
  - **Stack Limit:** Contains the address of the top end of the reserved block.



Address Details

- **Stack Pointer:** Updated on PUSH (decrement) and POP (increment) operations.
- **Stack Base:** First location used when adding to an empty stack.
- **Stack Limit:** Prevents pushing an item when the stack is full.

Stack Growth

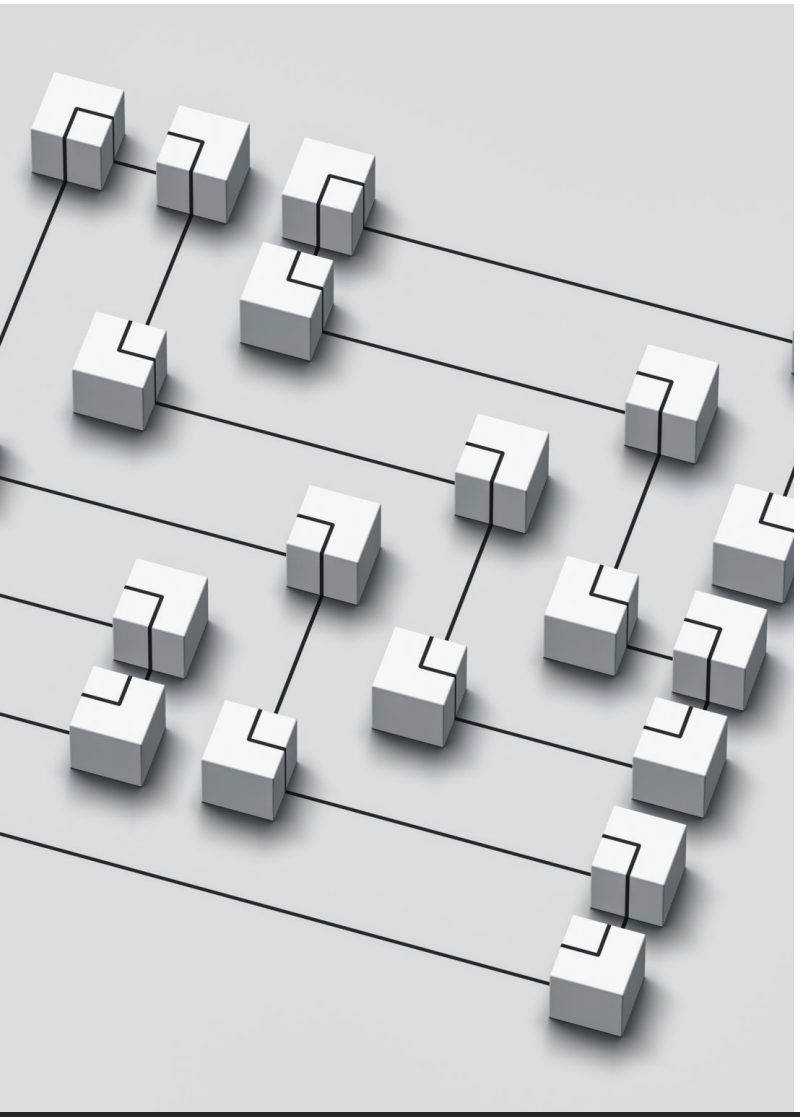
- Traditionally, the base of the stack is at the high address end.
- The stack grows from higher addresses to lower addresses.

Role of Stack in Program Execution

Program Calls:
Processor pushes function parameters and return addresses onto the stack.

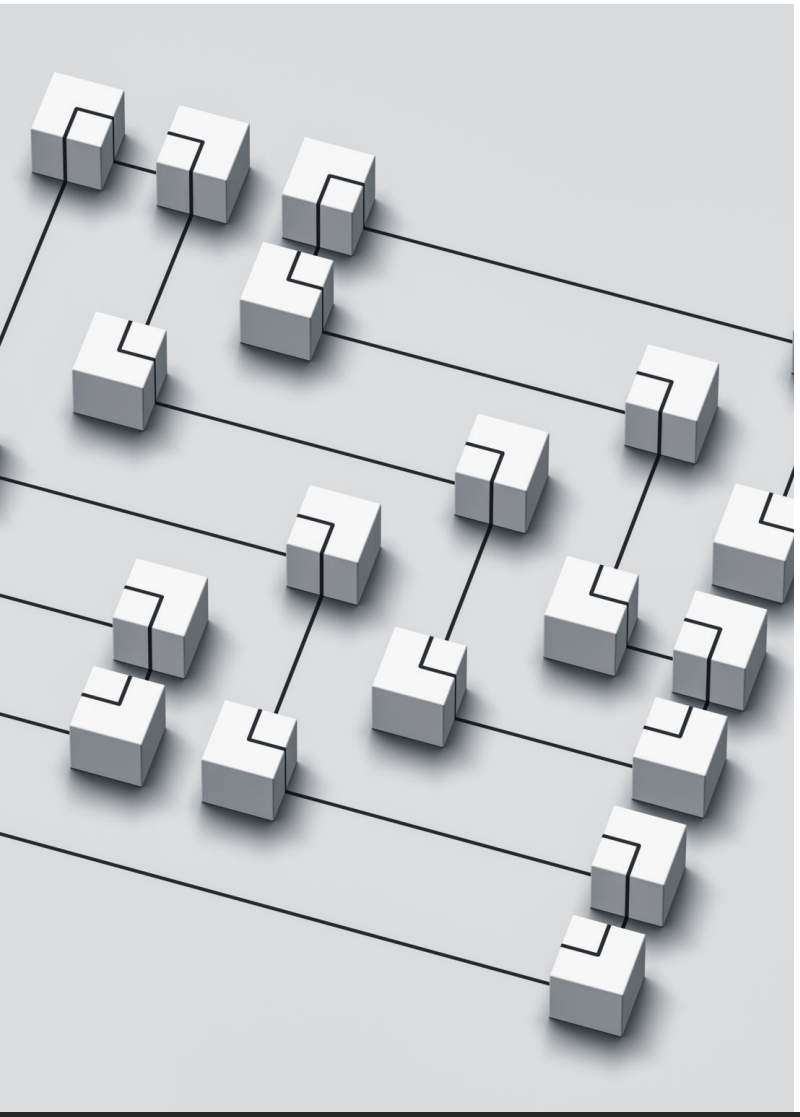
Function Usage:
Stack stores local variables and intermediate results.

Return Process:
Processor pops return address and local variables from the stack.



Stack in Interrupts and Exceptions

- **Interrupts/Exceptions:**
  - Processor pushes the current program counter and status onto the stack.
  - Jumps to the interrupt or exception handler.
- **Handler Usage:** Stack stores the context of the interrupted program and restores it afterward.



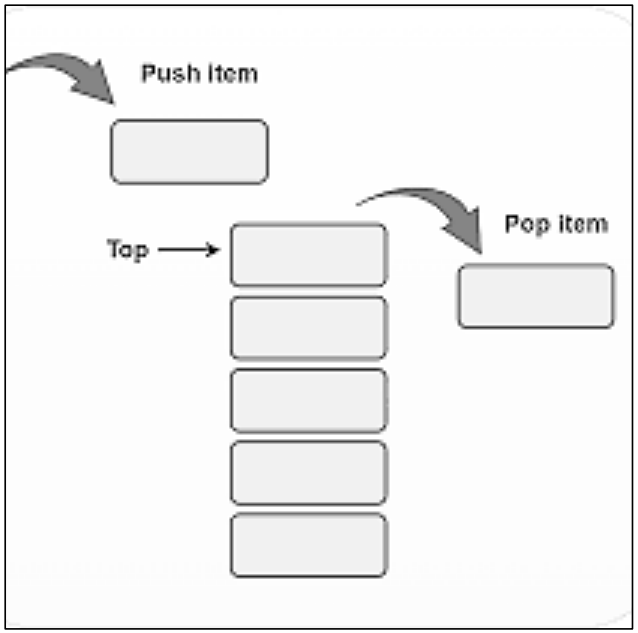
Implementation Considerations

Factors to Consider:

- **Stack Size:** Must accommodate maximum stack usage.
- **Stack Pointer:** Keeps track of the current stack position.
- **Stack Frame Layout:** Organizes function parameters, local variables, and intermediate results.

Introduction to Push and Pop

Push and pop are fundamental stack operations used in various aspects of operating systems.



They are crucial for:

- Function Call Management
- Interrupt Handling
- Context Switching

Stack Pointer (SP)

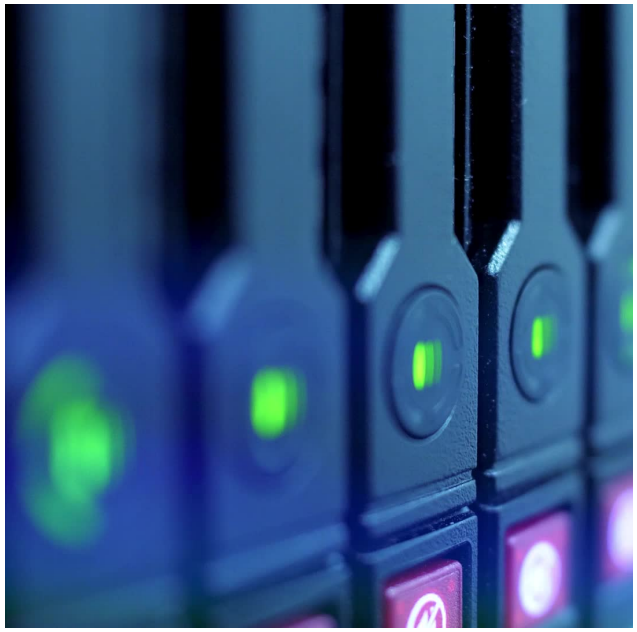
Definition: A register that holds the address of the top of the stack.

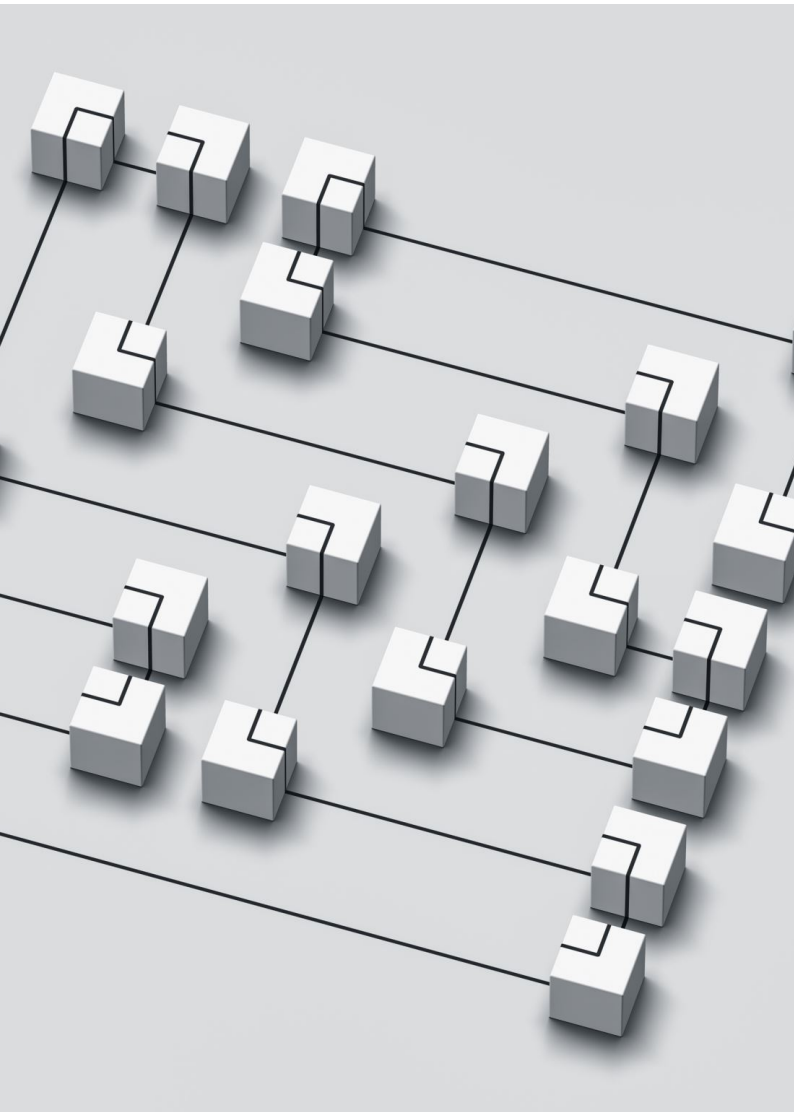
Updates on Operations:

- **PUSH:** Decremented to add a new item.
- **POP:** Incremented to remove the top item.

Importance:

- **Efficient Access:** Allows quick access to the top of the stack.
- **Error Handling:** Helps in detecting stack overflow (when SP goes below stack base) and stack underflow (when SP exceeds stack limit).



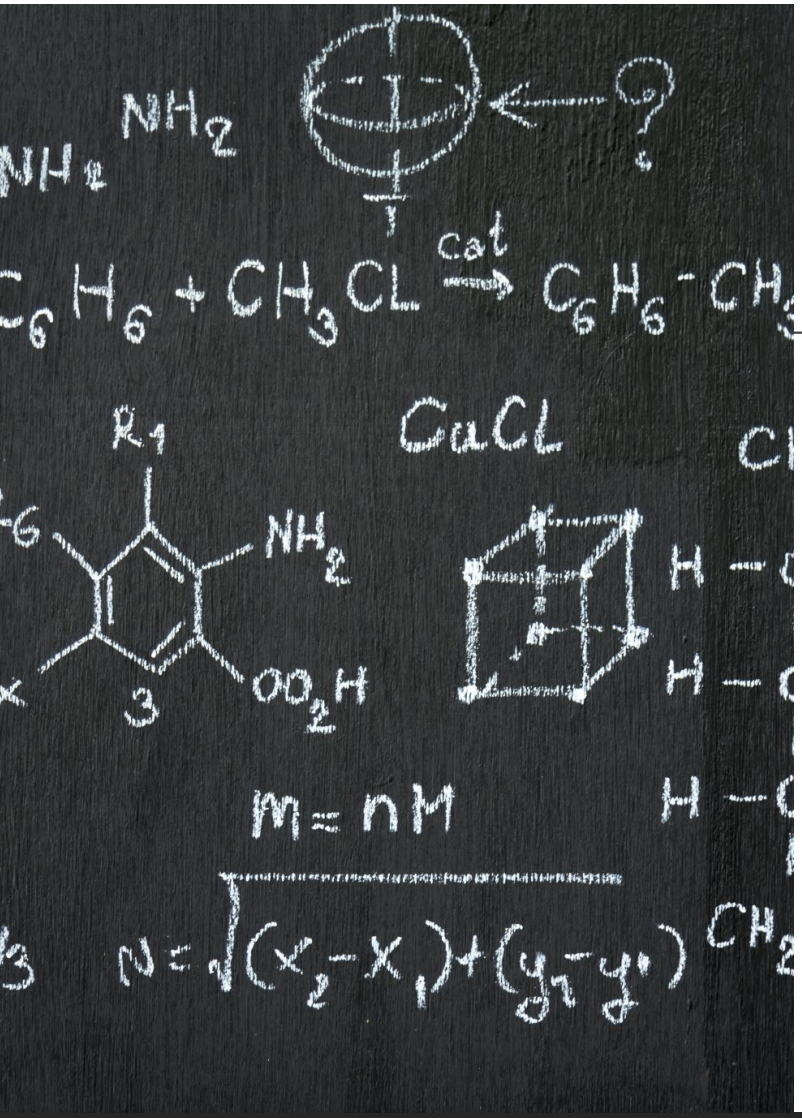


PUSH Operation

Definition: Adds an item to the top of the stack.

Process:

1. **Check Stack Limit:** Ensure the stack is not full.
2. **Update Stack Pointer:** Decrement the stack pointer to point to the next available position.
3. **Store Item:** Place the item at the position the stack pointer indicates.

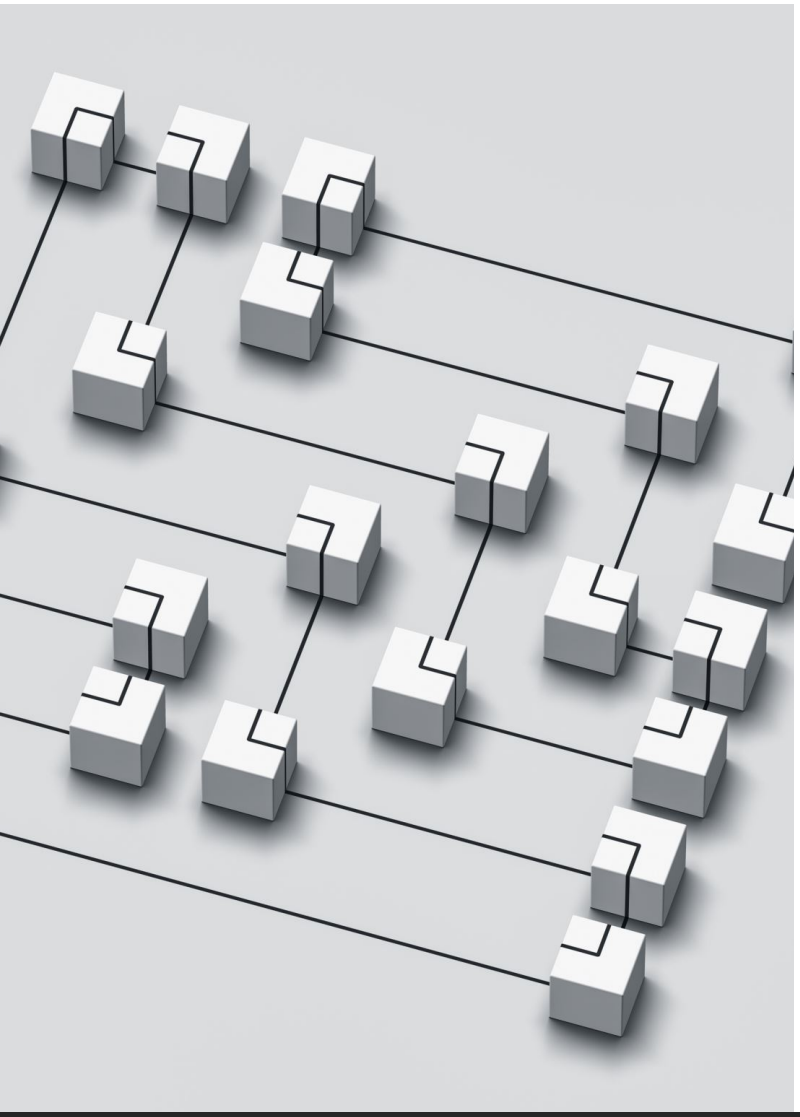


PUSH Operation Example

- Initial Stack:
- SP = 4
- Stack: [1, 2, 3] (Top = 3)
- PUSH(4):
- SP is decremented (SP = 3).
- Item 4 is placed at position 3.
- Updated Stack: [1, 2, 3, 4] (Top = 4)

Visual Representation:

- Initial State: [1, 2, 3]
- After PUSH: [1, 2, 3, 4]



POP Operation

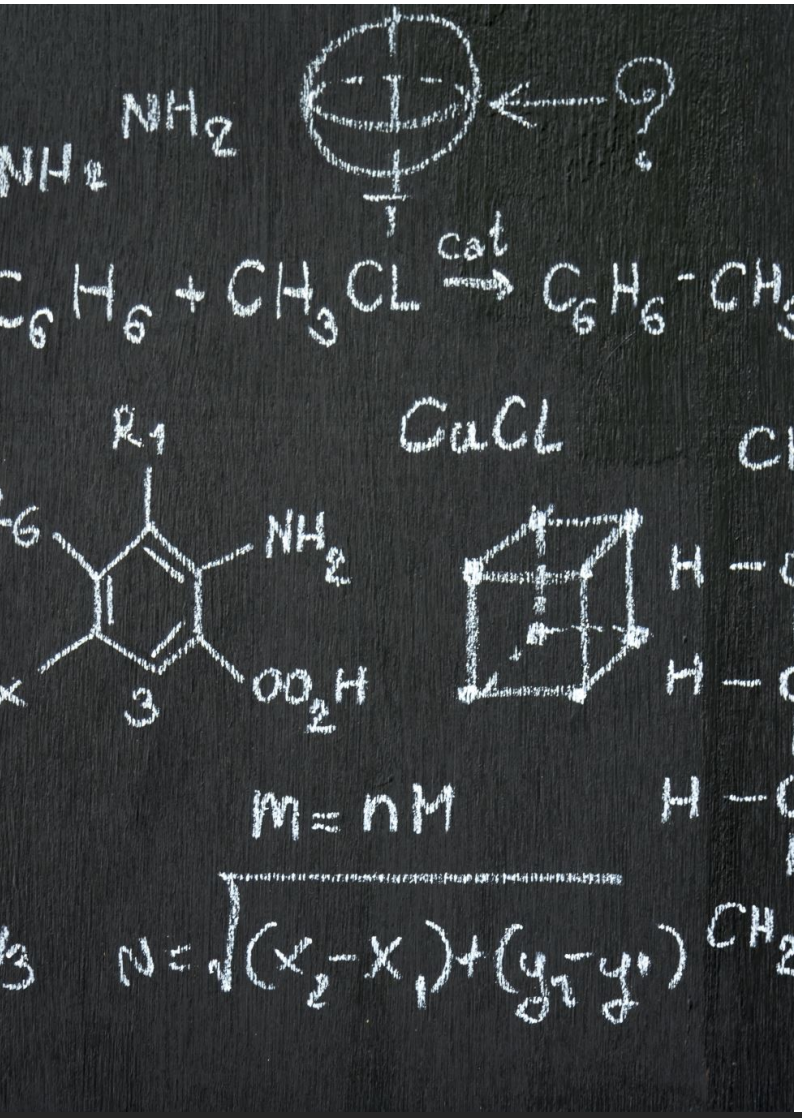
Definition: Removes an item from the top of the stack.

Process:

1.Check Stack Base: Ensure the stack is not empty.

2.Retrieve Item: Get the item from the position indicated by the stack pointer.

3.Update Stack Pointer: Increment the stack pointer to point to the new top of the stack.

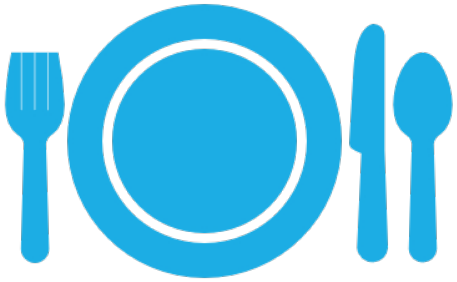


POP Operation Example

- Initial Stack:
- SP = 3
- Stack: [1, 2, 3, 4] (Top = 4)
- POP():
- Item 4 is retrieved from position 3.
- SP is incremented (SP = 4).
- Updated Stack: [1, 2, 3] (Top = 3)

Visual Representation:

- Initial State: [1, 2, 3, 4]
- After POP: [1, 2, 3]



Understanding Push and Pop Operations Using Dish Cleaning Analogy

Push Operation in Dish Cleaning

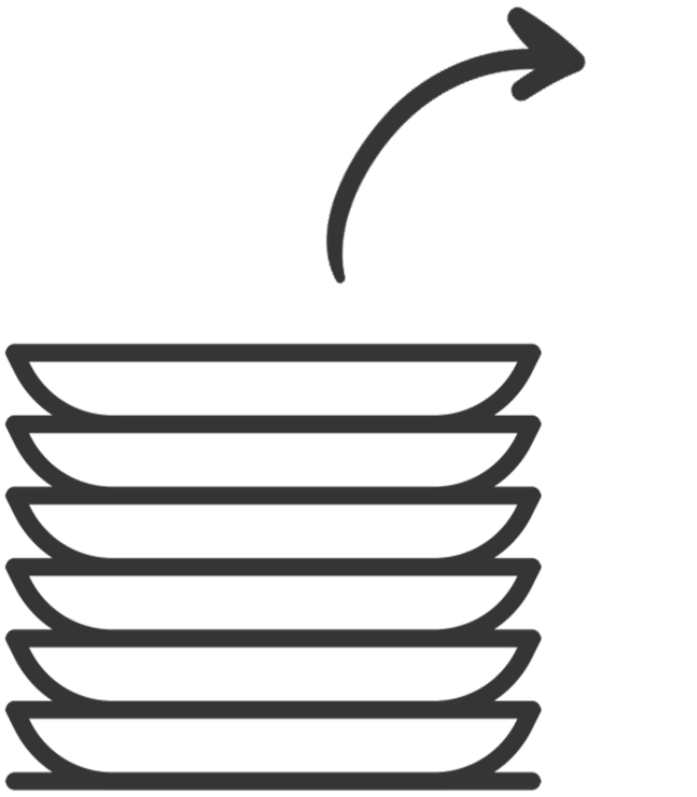
- **Analogy: Adding a dish to the stack**
  - You wash a plate and place it on the stack of clean plates.
  - The newly washed plate is added on top of the stack.
- **Push Operation in OS**
  - Adding data (e.g., function context) to the stack.
  - Example: When a function is called, its context is pushed onto the stack.

Pop Operation in Dish Cleaning

- **Analogy: Removing a dish from the stack**
  - When you need a clean plate, you take the top one from the stack.
  - The top plate is removed from the stack.
- **Pop Operation in OS**
  - Removing data (e.g., function context) from the stack.
  - Example: When a function call is completed, its context is popped from the stack.

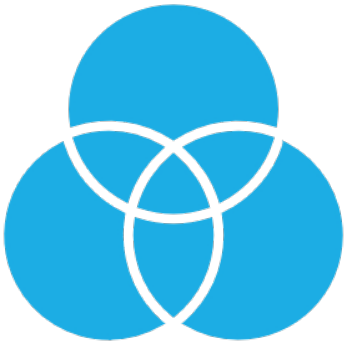
Example Scenario - Push

- **Starting with no dishes (empty stack)**
- **Washing dishes (Push):**
  - Wash Plate 1 (Push Plate 1 onto the stack)
  - Wash Plate 2 (Push Plate 2 onto the stack)
  - Wash Plate 3 (Push Plate 3 onto the stack)



Example Scenario - Pop

1. **Using dishes (Pop):** Take Plate 3 (Pop Plate 3 from the stack)
2. Take Plate 2 (Pop Plate 2 from the stack)
3. Take Plate 1 (Pop Plate 1 from the stack)



What other real-life analogy do you think exists?

Real-life Analogies

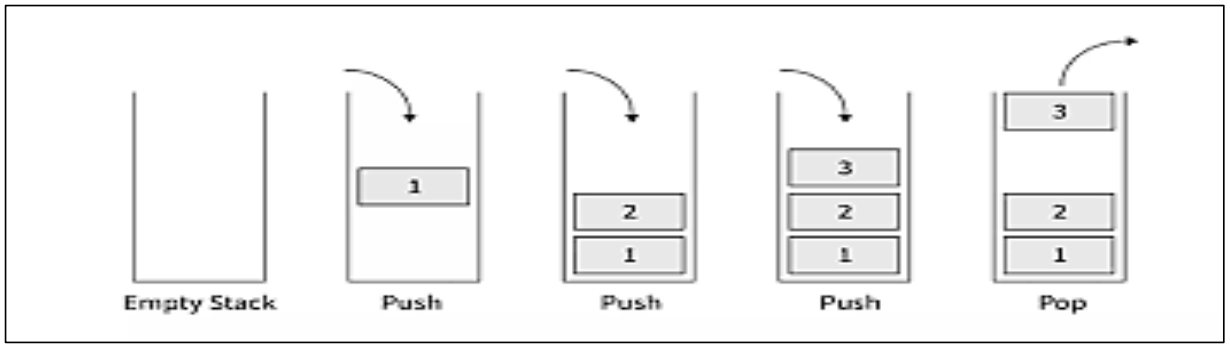
1. **Stack of Pancakes**: Imagine a stack of pancakes where you can only add new pancakes to the top of the stack (push operation) and remove the top pancake (pop operation).
2. **Stack of Building Blocks**: Children often play with building blocks, stacking them one on top of the other. Each time a new block is added, it goes on top of the stack, and when a block is removed, it's taken from the top.
3. **Line at a Ticket Counter**: In a line at a ticket counter, people join the line (push) and are served in the order they joined. When someone is served, they are removed from the front of the line (pop).
4. **Stack of Music CDs**: Stack CDs on top of each other; when you want to listen to one, you take the top CD (pop), and when you buy a new CD, you add it to the top of the stack (push).
5. **Stack of Nested Boxes**: Imagine a set of nested boxes where each box can contain smaller boxes. When you add a new box to the stack, it goes on top, and when you remove a box, you take the top one off.
6. **Stack of Coins in a Coin Dispenser**: In a coin dispenser, coins are stacked on top of each other, and when you dispense a coin, it's taken from the top of the stack.
7. **Stack of Luggage at an Airport**: Luggage is stacked on top of each other as it arrives at the airport. When someone collects their luggage, it's taken from the top of the stack.

Function Call Management

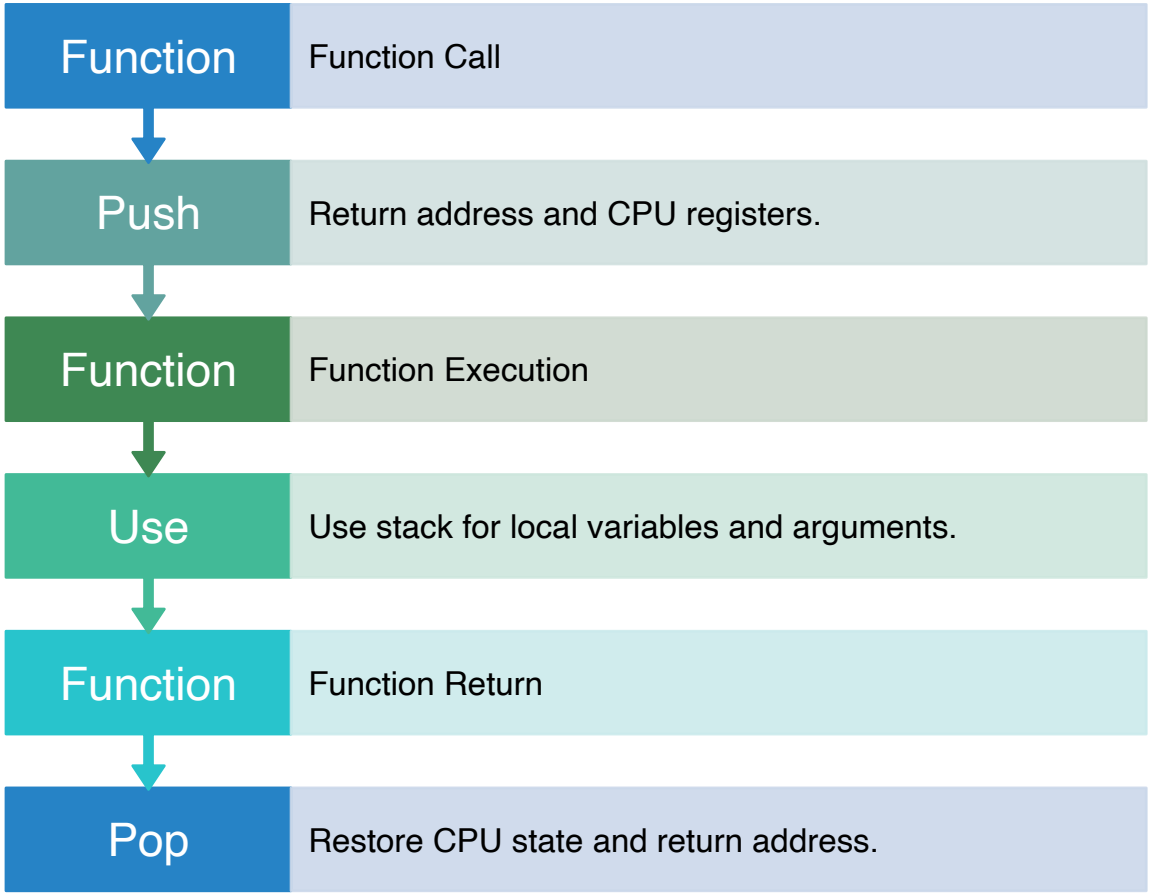
When a function is called:

- **Push**: Save return address and registers.
- **Pop**: Restore return address and registers.

This allows the function to execute without losing the context of the caller.



Function Call Management - Detailed Process



Interrupt Handling

During an interrupt:

- **Push**: Save program counter and status register.
- **Pop**: Restore program counter and status register.

This allows the interrupt service routine (ISR) to execute.

Interrupt Handling - Detailed Process

Interrupt Occurs

- Push: Program counter and status register.

ISR Execution

- Handle the interrupt.

Return from Interrupt

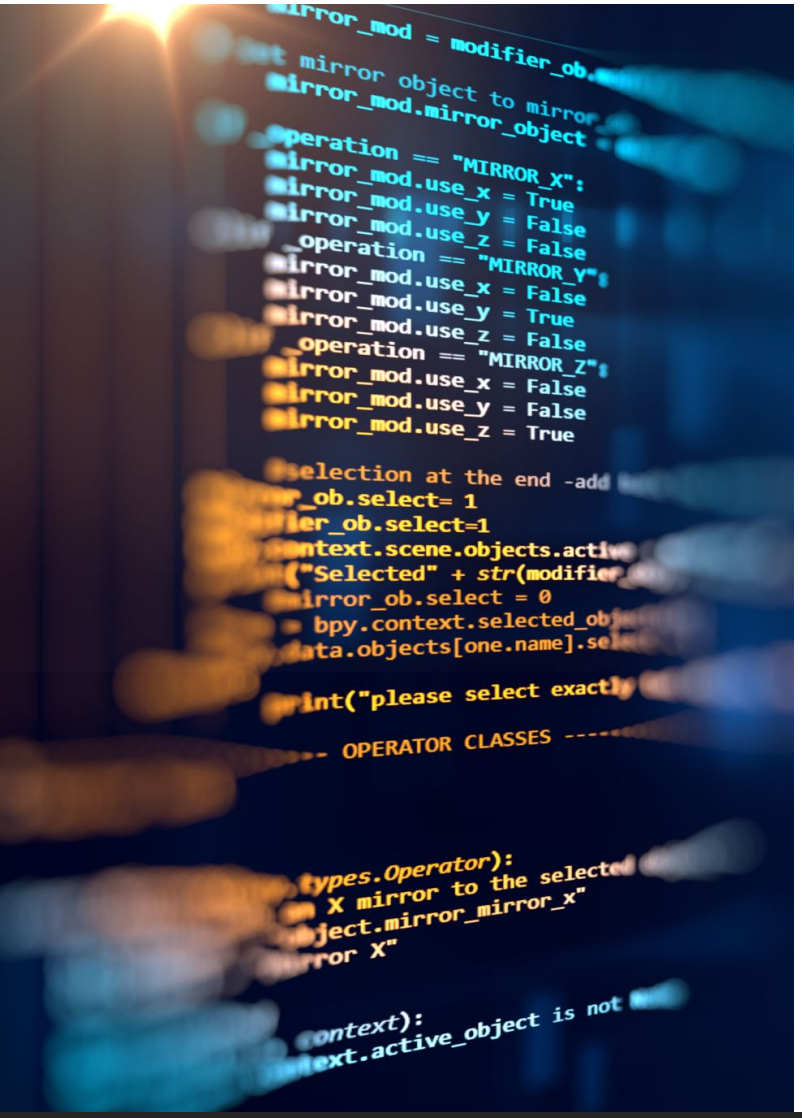
- Pop: Restore CPU state and program counter.

Context Switching

In multitasking systems:

- **Push**: Save current process state.
- **Pop**: Restore next process state.

This allows the CPU to switch between processes.



Context Switching - Detailed Process

1. **Save Current Process State:**
   - Push: CPU registers and program counter.
2. **Scheduler Switches Context:**
   - Select next process to run.
3. **Restore Next Process State:**
   - Pop: CPU registers and program counter.

Example in Assembly Language

Example of push and pop in x86 assembly:

1. **Push**: Save register states.
2. **Operation**: Perform tasks.
3. **Pop**: Restore register states.
4. **Exit**: End program.

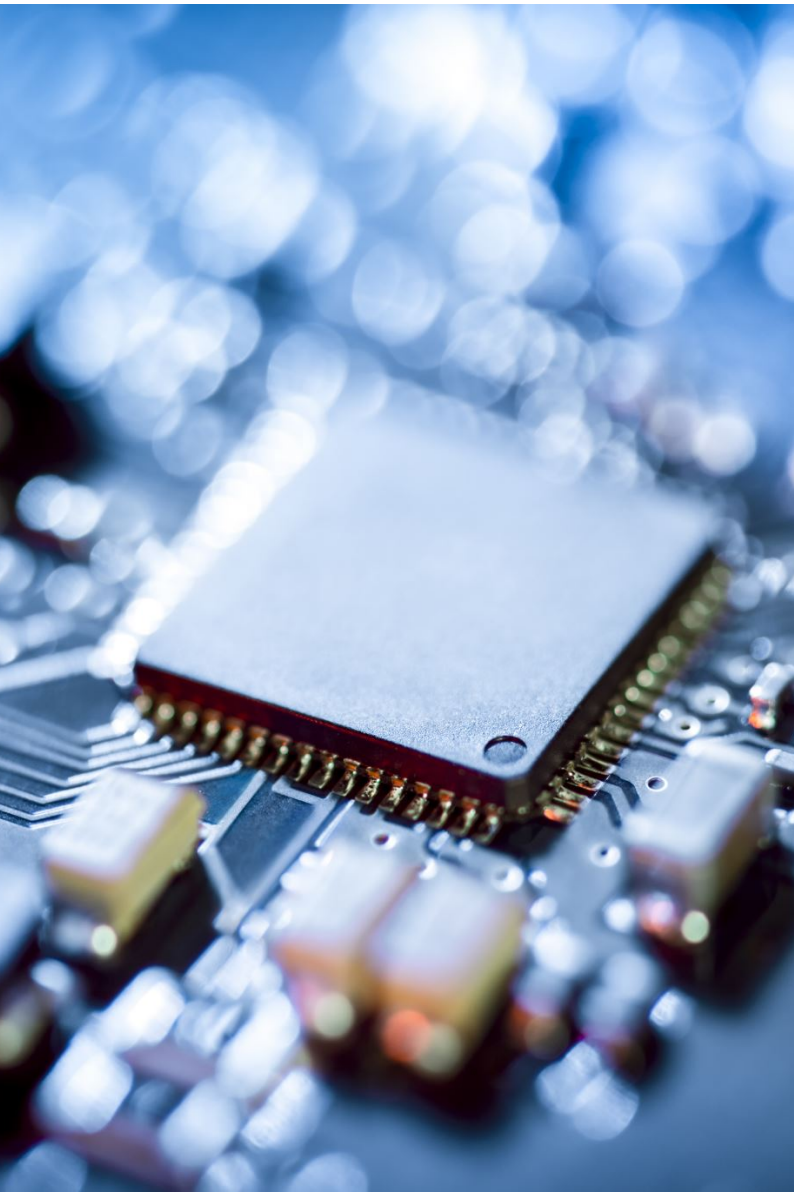
Example in Assembly Language - Code

```
section .data
    msg db 'Hello, world!', 0

section .bss

section .text
global _start

_start:
    push eax
    push ebx
    ; perform operations
    pop ebx
    pop eax
    mov eax, 1
    xor ebx, ebx
    int 0x80
```



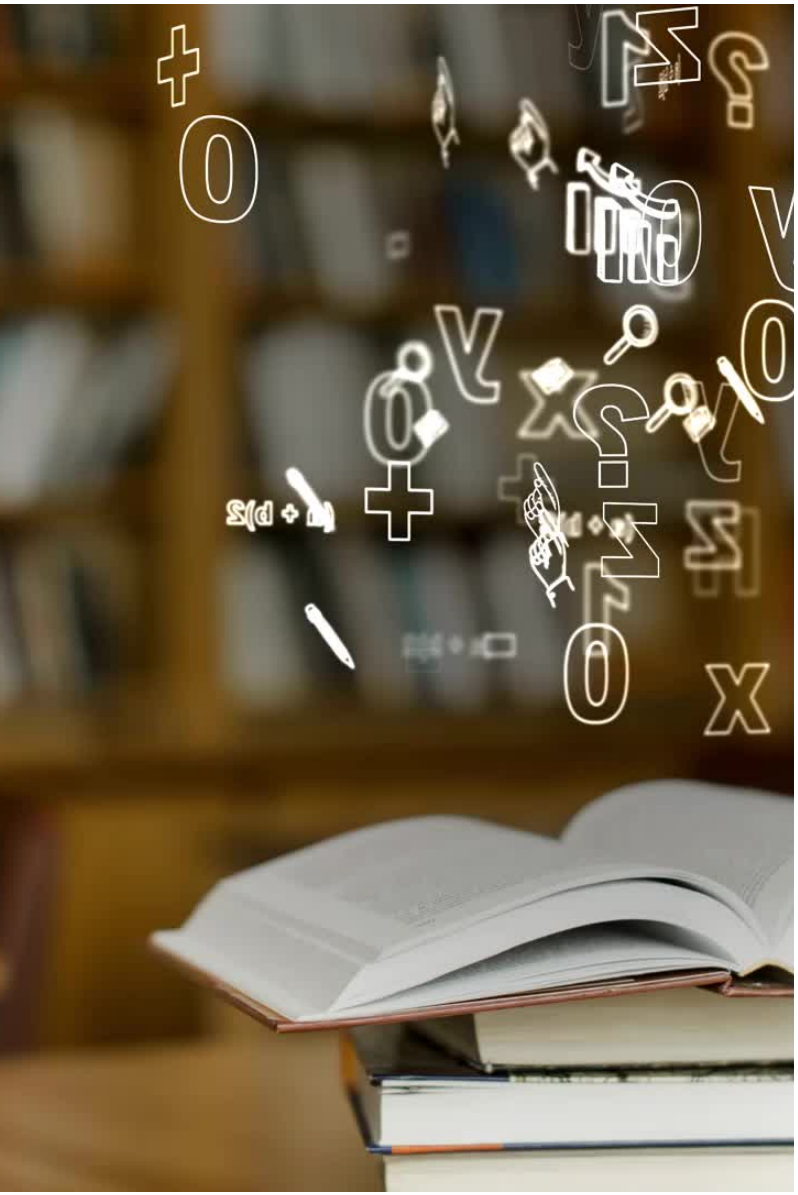
Importance of Push and Pop

- **Memory Management:** Essential for efficient memory allocation and deallocation.
- **Function Calls:** Facilitates storing and retrieving local variables, parameters, and return addresses.
- **Recursion:** Prevents stack overflow errors by managing recursive function calls.
- **Interrupt Handling:** Utilized for saving and restoring CPU states during interrupts.
- **Context Switching:** Enables efficient process state saving and loading during multitasking.
- **Resource Management:** Crucial for allocating and deallocating system resources in a stack-like manner.

Activity

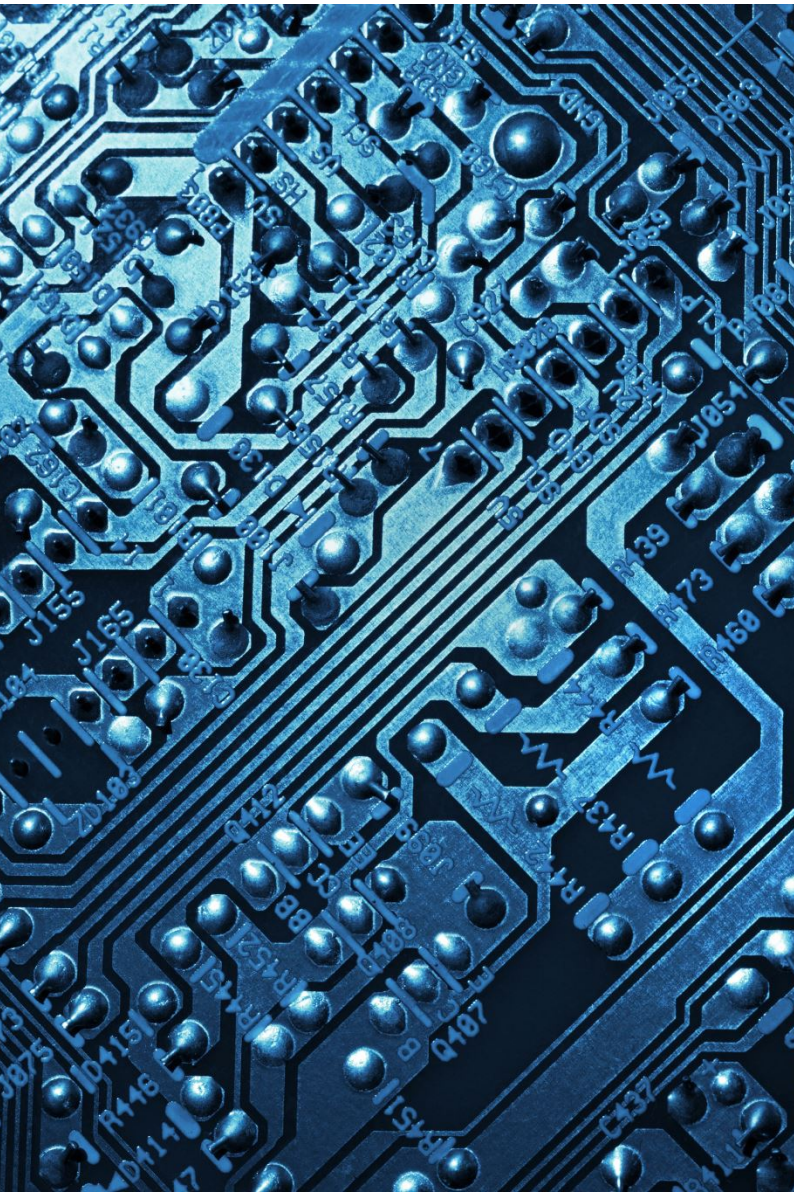
https://www.sigcis.org/files/A%20brief%20history.pdf

- Scan this QR code or copy the link above to read the article titled ‘A brief history of the stack’ by Sten Henriksson.
- Then, be prepared to answer multiple-choice questions that the professor will ask in class.



What are the key axioms that define the behavior of a stack?

- Axiom 1: isempty(create()) - A newly created stack is empty.
- Axiom 2: ¬ isempty(push(x, S)) - A stack with an added element is not empty.
- Axiom 3: pop(push(x, S)) = S - Pushing and then popping an element leaves the stack unchanged.
- Axiom 4: top(push(x, S)) = x - The top of a stack after pushing an element is the pushed element.
- Axiom 5: ¬isempty(S) ⇒ (push(top(S),pop(S))=S) - Removing and then replacing the top element of a non-empty stack leaves it unchanged.

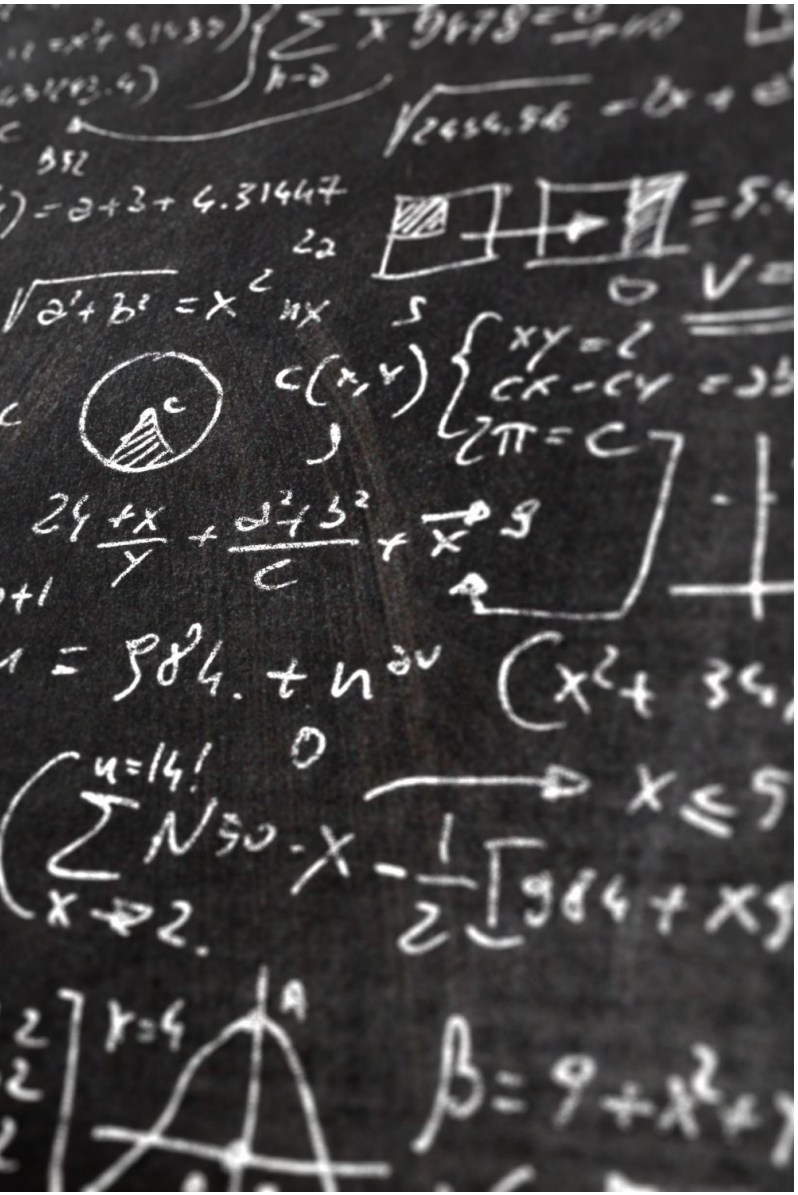


Who is credited with emphasizing the importance of the stack in the development of computer science?

Michael Mahoney highlighted the role of stacks in the theory of automata and formal languages, noting their foundational importance between 1955 and 1970.

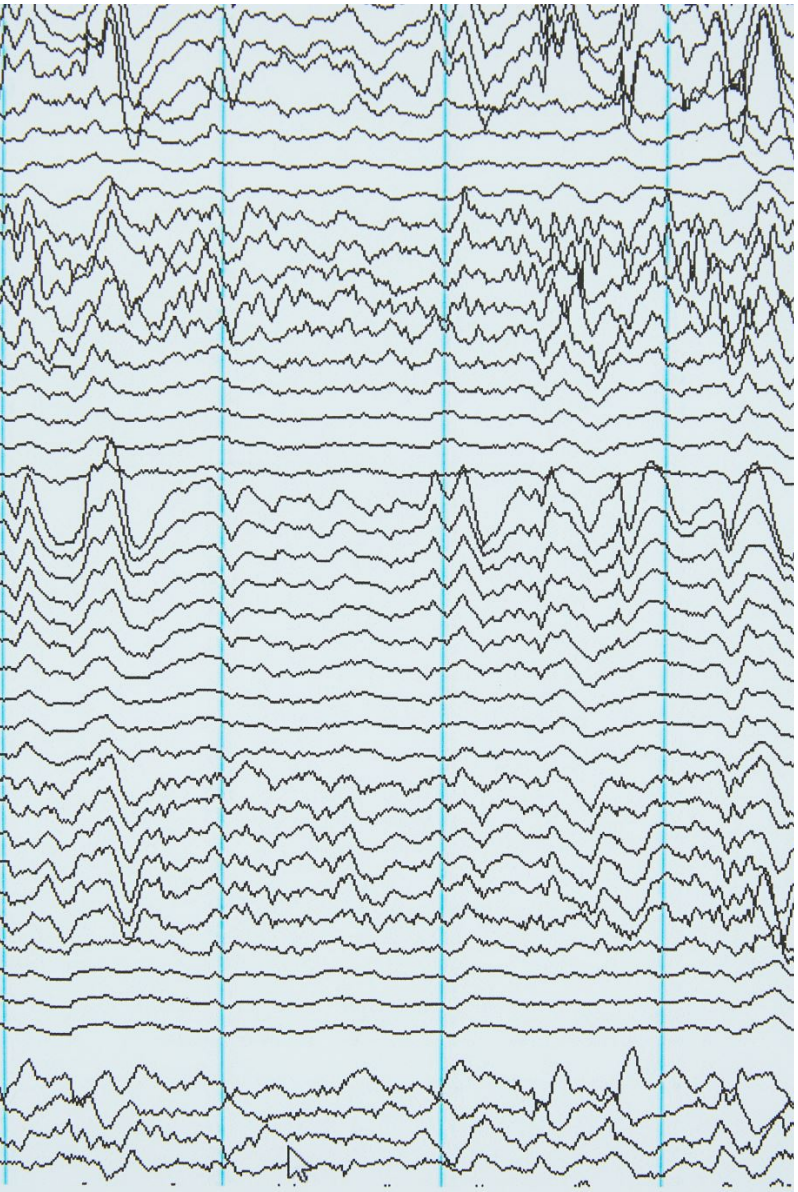
How did stacks facilitate the development of programming languages like Algol 60?

Stacks allowed Algol 60 to handle procedure calls and block structures effectively, enabling recursion and proper variable scope management.



What is the historical significance of the "cellar principle" in stack development?

The cellar principle, introduced by F.L. Bauer and K. Samelson, involved storing intermediate results in reverse order, facilitating the translation of infix to postfix notation and influencing the design of recursive languages.



What is a cumulative tale, and how does it illustrate the stack principle?

A cumulative tale, like "The Old Woman and Her Pig," adds incidents sequentially and resolves them in reverse order, akin to how a stack operates.

Who is credited with the first use of a stack in a computer science context according to the OED?

a) Donald Knuth

b) F.L. Bauer

c) E.W. Dijkstra 

d) John Backus