TECHNISCHE UNIVERSITÄT DRESDEN

DRESDEN concept

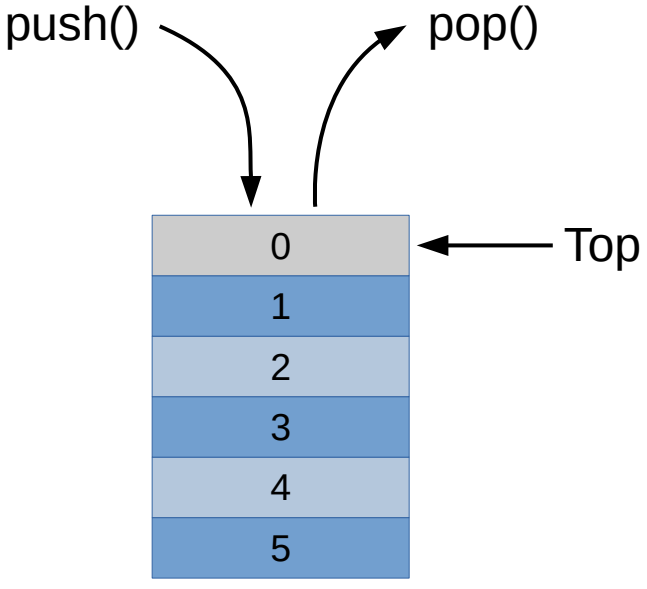
Faculty of Computer Science, Institute of Systems Architecture, Chair of Systems Engineering

RoboLab Autumn Course – Stack Machines

by M.Sc. Samuel Knobloch

What is a Stack?

- Conceptual structure
  - ➔ Commonly used abstract data type
  - ➔ Consists of homogeneous elements
- Is based on the Last In First Out (LIFO) principle
  - ➔ Also called *pushdown* stack
- Two major operations for access
  - ➔ push() - Add an element to the stack
  - ➔ pop() - Remove the top element from the stack
- Commonly used in programming and memory organization in computers



What is a Stack?

- A stack
  - ➔ … is a linear data structure format
  - ➔ … represents a sequence of objects or elements
  - ➔ … can be fixed in size or dynamically implemented
  - ➔ … has a limited set of operations allowed to work with (depends on implementation)

- Can be implemented in:
  - ➔ Software: Arrays, Linked Lists
  - ➔ Hardware: Stack Registers for memory allocation and access

- Some commonly used operations are:
  - ➔ duplicate(), peek(), swap() / exchange(), rotate() / roll()

What is a Stack Machine?

- In general, a specific type of computer or virtual machine
  - ➔ Can also refer to a software schema to simulate a Stack Machine
- Works with an instruction set on a *pushdown* stack
  - ➔ Does not use registers for variables (see Register Machines for more information)
- For programming Stack Machines ***Reverse Polish Notation*** (RPN) is used
  - ➔ https://en.wikipedia.org/wiki/Reverse_Polish_notation

Advantages of Stack Machines

- Very compact object code, which means having smaller instructions compared to other styles of machines
- Compilers are faster, simpler and quicker to build than compilers for other machines
  - ➔ Code generation is trivial
  - ➔ Code generated is independent of prior or subsequent code
- Interpreters are simple and fast
- Operand access is fast, there is no need to decode any operand fields
  - ➔ Instructions and its operands are fetched at the same time

Disadvantages of Stack Machines

- Stack Machines sometimes have performance issues:
  - ➔ More memory references required because of temporary values and local variables
    - E.g. compared to register machines
  - ➔ Factoring out common sub-expressions has high costs
    - We don't store the result for later use (default; there are some ways to do so)
- The more instructions the slower the interpreters
  - ➔ Compared to register machines, way more instructions have to be executed when compiling a program
    - For every variable, there is a separate *Load* instruction instead of bundling with other instructions

Stack Machines – Examples

- Commercial stack machines
  - ➔ UCSD Pascap p-machine (Pascal MicroEngine)
  - ➔ Inmos transputers
  - ➔ ZPU
- Virtual stack machines
  - ➔ Java Virtual Machine (Dalvik, 8-bit stack)
  - ➔ Lua (before v5.0)
  - ➔ Rubinius
  - ➔ Whetstone ALGOL 60
- More: https://en.wikipedia.org/wiki/Stack_machine#Commercial_stack_machines

Stack-oriented programming

- Operates on one or more stacks
- Most stack-oriented languages operate in ***postfix notation*** (also called *RPN*)
  - ➔ Arguments or parameters for an operation are stated before the operation

    Examples: `2 * 3` → `2 3 * (postfix)`
    → `* 2 3 (prefix)`
    → `2 * 3 (infix)`

- We use Reverse Polish Notation here
  - ➔ Mathematical notation
  - ➔ Operators follow their operands (postfix)
  - ➔ We don't have parentheses, so take care of sub-expressions

Reverse Polish Notation – Example

- Given the following expression: `30 – 4 * 5`
- To solve this we need to apply simple maths and write it down in the correct order
  - ➔
    ```
    a:      4 * 5
    b:      30 – a
    ```
- In RPN, the solution looks like this:
  - ➔
    ```
    30
    30    4
    30    4    5
    30    4    5    *    -
    ```
  - ➔ In short, we push 30, 4 and 5 to the stack, pop twice afterwards and multiply both operands
  - ➔ Then, we push back the result and pop twice for the subtraction
  - ➔ Finally, we push back the final result

Assignment 3 – Stack Machine (Theory)

Assignment 3 – Stack Machine (Theory)

- Task 1
  - ➔ Do some research on the given topics and answer the questions afterwards
- Task 2
  - ➔ Make yourself more familiar with RPN
  - ➔ Transform the given expressions into RPN without pre-calculating sub-expressions
- Task 3
  - ➔ Make yourself familiar with our stack machine specification
  - ➔ Create binary representations of the given sequences
  - ➔ Simulate the execution of both sequences and write down the final stack results
    - It is advised to use a table for writing down the state after each step