18ES611 Embedded System Programming

Sarath tv

Stack

stack as a temporary storage for data.

Last In First Out (LIFO), the last data stored in the stack is the first than can be retrieved and the first data stored in the stack is the last to be retrieved.

PUSH and the **POP(PULL)**.

Analogy - **pile** of **plates**. The **bottom** plate is the **first data** pushed onto the stack and the **top plate** is the **last data** pushed.

When the **top plate** is removed (**pulled**), the **one below pops** up to become the **new top**.

only the **top element** in the **stack can be accessed.**

- Stack segment is used to **store variables** which are created **inside functions** (*function could be main function or user-defined function*), variable like

  - **Local variables** of the function
  - **Arguments passed to function**
  - **Return address**

- Variables stored in the stack will be removed as soon as the function execution finishes.

Stack

The stack area contains the **program stack**, a LIFO structure.

A “**stack pointer**” register **tracks** the **top** of the stack;

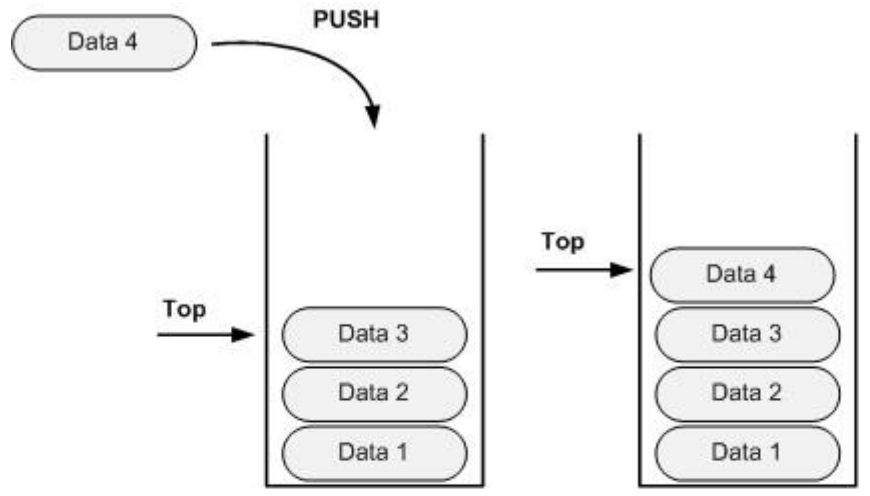
it is **adjusted** each time a value is “**pushed**” onto the stack.

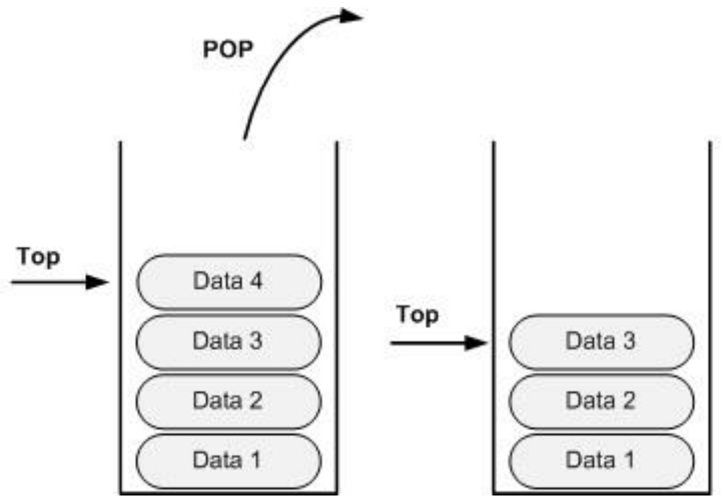
The **set of values pushed for one function** call is termed a “**stack frame**”;

A stack frame consists at **minimum of a return address**.

Each time a function is called, the address of where to **return** to and **certain** information about the **caller’s environment**.

The **newly** called **function** then **allocates room** on the stack for its **automatic** and **temporary variables**.


Data 4
PUSH
Top
Data 3
Data 2
Data 1
Top
Data 4
Data 3
Data 2
Data 1


POP
Top
Data 4
Data 3
Data 2
Data 1
Top
Data 3
Data 2
Data 1

Each function is given its own section of memory to operate in.

This section is referred to as a ***stack frame***.

It most commonly includes the following components:

- The **return address** (when the function is complete it returns back to the function call).
- **Arguments passed** to the function.
- **Local variables** of the function.
- **Saved copies** of any registers modified by the function that have to be restored.

When a program enters a function, space is allocated on top of the stack.

When the **program leaves** the function, the **space is freed**.

The **life span** of variables on the stack is limited to the **duration of the function**.

Implementation of Stack

The common hardware implementation of **stack** consists of **reserved contiguous region of memory** with a **stack pointer** into that memory.

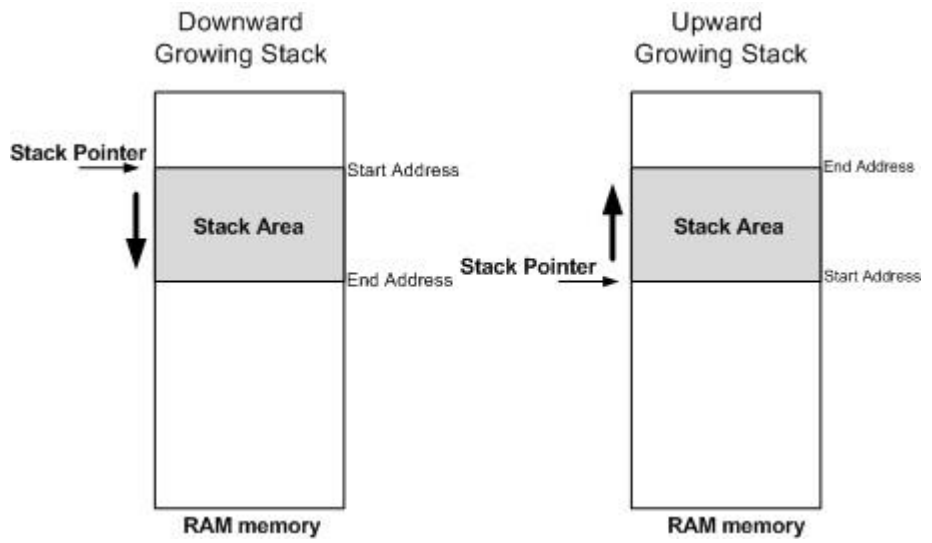
The stack has **a fixed location in memory** at which it begins.

The *stack pointer* is a **hardware register** that points to the **current extents of the stack**. There are stacks that

grow downwards and

ones that grow upwards.

The stack can technically be located anywhere in memory, depending on the system.



Types of Stack

Software to implement a stack - **different implementation** choices result different types of stacks.

There are ***two types of stack depending on how the stack grows***.

Ascending Stack **-**grows upwards. Starts from a **low memory address** and, as items are pushed onto it, **progresses to higher memory addresses**.

In a push the **stack pointer is incremented**, i.e. the stack grows towards higher address.

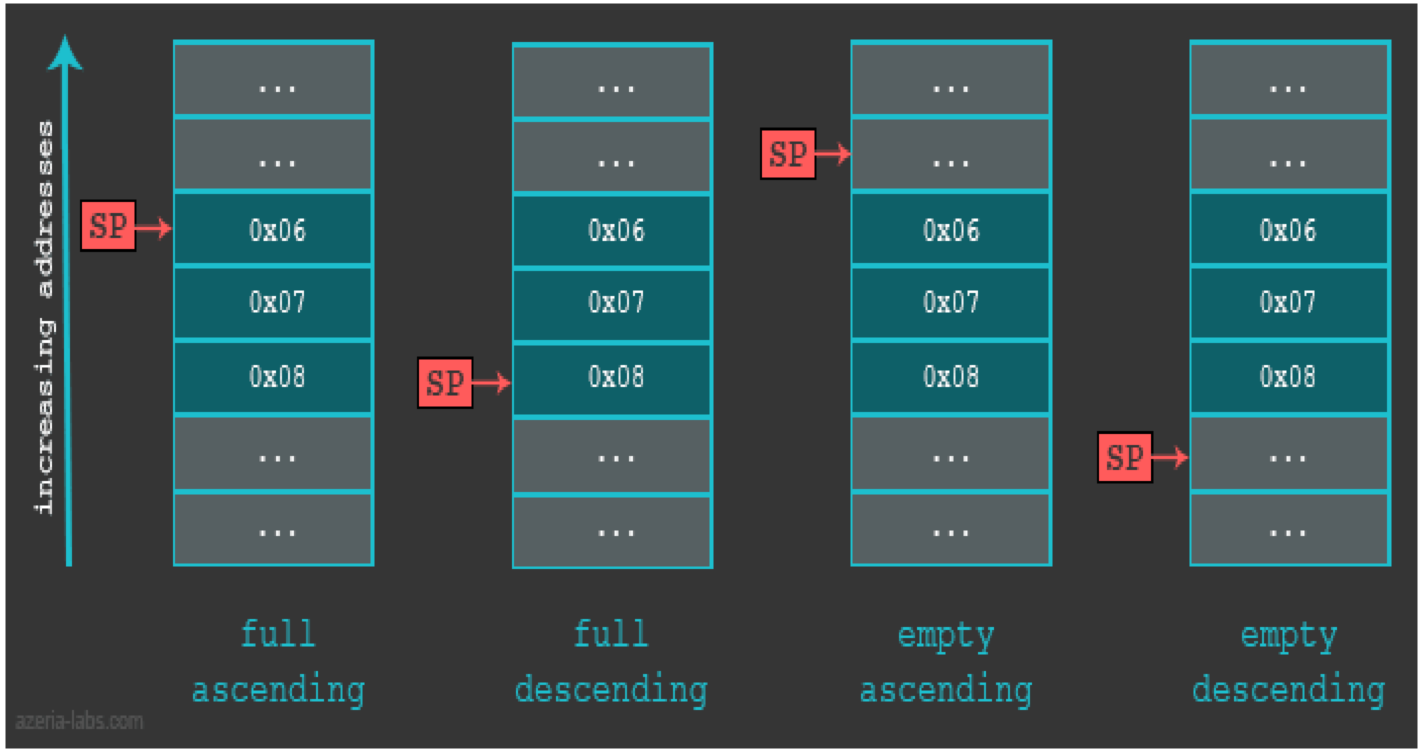
Descending Stack -grows downwards. It starts from a **high memory address**, and as items are pushed onto it, **progresses** to **lower memory addresses**. In a push the **stack pointer is decremented**, i.e. the stack grows towards lower address.

There are ***two types of stack depending on what the stack pointer points*** to.

Empty stack -Stack pointer **points** to the **location** in **which** the **next item** will be **stored**. A **push** will **store** the **value**, and **increment** the **stack pointer**. The stack pointers points to the next free (empty) location on the stack,

Full stack **-** Stack **pointer points** to the **location** in which the **last item** was **stored**. A **push** will **increment** the **stack pointer** and **store** the value. the stack pointer points to the topmost item in the stack, i.e. the location of the last item to be pushed onto the stack.

Four different stacks are possible - *full-ascending, full-descending, empty-ascending, empty-descending*.


increasing addresses
SP
...
0x06
0x07
0x08
full ascending
full descending
empty ascending
empty descending
azeria-labs.com

THANK YOU!!!!!