# PROGRAMMING CONCEPTS AND EMBEDDED PROGRAMMING IN C, C++ and JAVA:

# Lesson-4: Data Structures: Stacks

# STACK

- A structure with a series of data elements with last sent element waiting for a delete operation.
- Used when an element is not to be accessible by the index with pointer directly, as in an array, but only through LIFO (Last in first out) mode through a stack-top pointer.

# Push and Pop onto a **STACK**

- A data-element pushed (inserted) only at the from front (stack-head) in the series of elements waiting for an operation and popped (deleted) also from front (stack-head).

- Only one pointer, for *deleting* after the read operation from stack-top and other for *inserting* at stack-top. Pointer if increments after a push operation then it decrements after a POP operation.

# Standard Functions used in a Stack

1. SELInsert – Pushes a data-element into the stack as pointed by item and increment the item pointer address
2. SELReturn – Pop an element from the stack as pointed by *item and the element deletes from stack on decrement in the item pointer address

3. isSNotEmpty– Return true or false after the check for the stack not empty.

# Stack Pointer

- SP (stack pointer): SP is pointer to a memory block dedicated to saving the context on context switch to another ISR or routine.

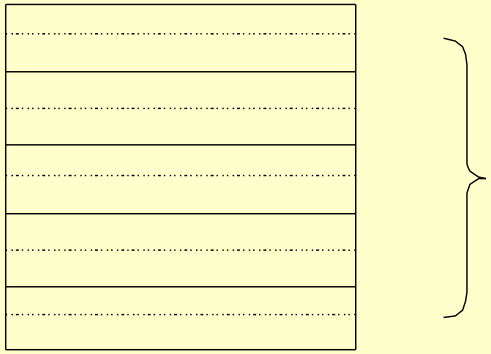
# Stack Pointer

- Each processor at least one stack pointer so that the instruction stack can be pointed and calling of the routines can be facilitated
- Each task or process or thread executing in control of the OS– Separate stack pointer for saving the context
- Stack frame – A set of stacks in a system

# Register for Pointing Return Address

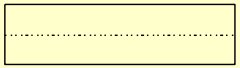
- RIP (return instruction pointer) in some processors ─ a register for saving the return address of the program counter when a routine calls another routine or ISR
- RIP called link register (LR) in ARM processor

# A stack due to nested function calls and pushing of program counters

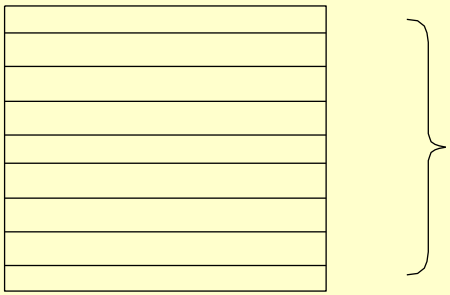


Stack
Holding Return
Addresses on
Nested calls

2 Byte
Address that PC
Acquires



# *A program-thread stack having pointers and parameters pushed on to stack before the context switch*



Stack
Holding Data
Retrievable in
LIFO Mode

A Memory Block
with Start and End

# FP (data frame pointer)

- A pointer to a memory block dedicated to saving the data and local variable values
- FIP in certain processors

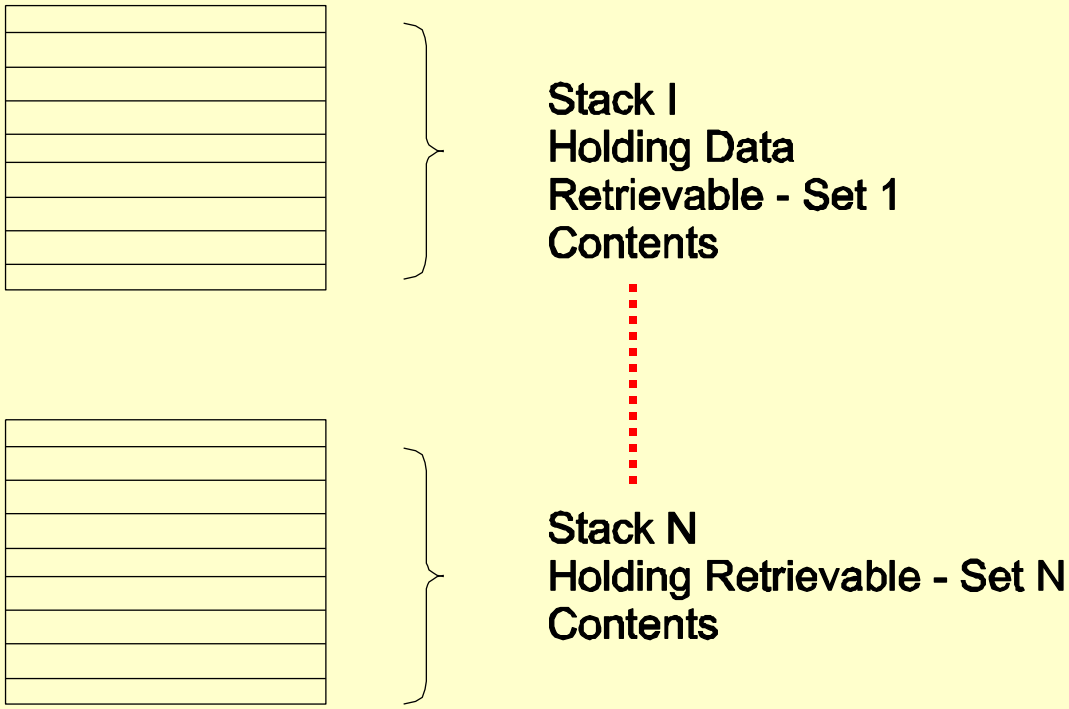
# PFP (previous program frame pointer)

- A pointer to a memory block dedicated to saved the program data frame, in certain processors

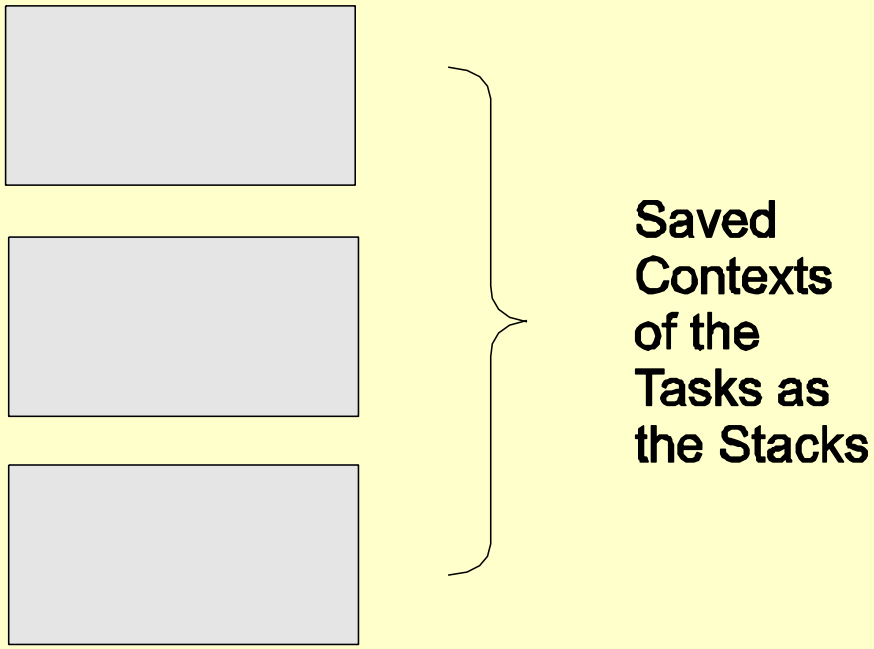
# Multiple stack frames for Multiple Threads

- OS defines processes or threads such that each process or thread is allocated one one task stack pointer or thread stack pointer

# *Multiple stacks of multiple tasks or threads pushed on the stacks with each having separate pointer*



# *Multiple stacks of CPU registers for the multiple tasks which are pushed on to stack before context switches*



# Motorola MC68010 processor

- USP (user Stack Pointer) and SSP (Supervisory Stack Pointer).
- Program runs in two modes: user mode and supervisory mode.
- In supervisory mode the operating system functions execute.
- Switch from user mode to supervisory mode after every tick of the system clock

# MC68040

- USP (User Stack Pointer), SSP (Supervisory Stack Pointer), (MSP) Memory Stack frames pointers, and Instruction Stack Pointer (ISP).

# Application Examples

1. Ease in saving of the data-elements in case of interrupts or function calls
2. Nested set of operations like function calls
3. A program thread, which blocks pushes on the stack and then the last pushed thread pops first.

# Summary

We learnt

- Using of *stack* is very helpful for saving the data in case of interrupts or function calls and saving a context.

# We learnt

- Stack related functions are 'constructing' a stack, 'pushing' an element into it, popping an element from it and 'destruction' of stack.

End of Lesson 4 of Chapter 7

on

**Data Structures: Stacks**