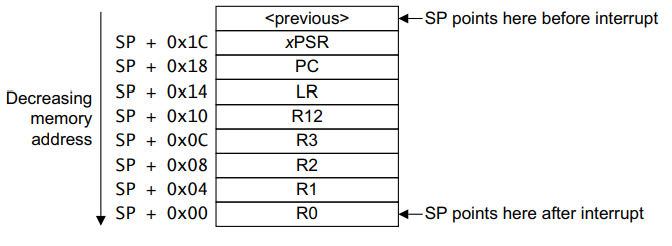
Context Switching

1. Register from R0 – R15 and xPSR values are considered as the context of a task. (17 registers)
2. Out of these registers R0 – R3, R12, R14 are called caller saved registers, which means these registers are saved as part of the stacking operation. And these registers are called as a stack frame.



1. When the currently running thread is interrupted by an exception handler for context switching. By default 7 registers from task context will be saved automatically before entering into the exception.
2. In the above there are 8 (7 register needed for task context) registers, in LR register EXEC\_RETURN instruction is stored

Context Switching Steps

* + - Before entering into the exception handler above mentioned 7 register will be saved automatically
    - Now Control is entered into the exception handler so, the first step is to terminate the currently running task
    - For that we have store the remaining unsaved register from the context of a task.
    - So, inside the the exception handler we have to *PUSH* register from R4 – R11 on the stack. Because R4-R11 are callee saved registers if the callee wants to alter these register it has to save these register before doing so.
    - Till now the stack pointer is pointing in the currently running task’s stack area.
    - Store the current stack pointer to currently running task’s task control block.
    - The currently running task’s context has been stored successfully
    - Now update the Stack pointer value of the processor with the next task’s stack pointer value.
    - If you look closely inside this exception handler, before terminating a task we are making all task’s current stack pointer to point to pushed R11 register, remember we pushed R4-R11
    - By doing POP from (R4-R11), these register values are restored (remember now it is pointing to the next task’s stack pointer)
    - As part of exception exit sequence, (R0 – R3), xPSR will be restored and using PC value the control will go the point where it left off in that task.

Starting the First Thread

1. In the below image we have created a array of uint32\_t for a stack of the task.
2. If we point the stack pointer to address of R0 of the array at the end of the function, as part of the unstacking operation, it will unstack the register R0-R3 and the control will go the address mentioned in the PC.

