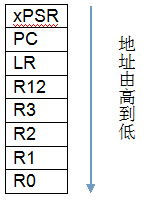
**文章转载自：**https://www.cnblogs.com/yeshenmeng/p/9590814.html

**1 入栈序列**

1.PendSV中断自动保存的数据入栈序列   
   
2. 加上手动保存的R4-R11的入栈序列

数组[N-1]= 任务环境数据栈顶  


数组[N-16]=任务环境数据栈底

数组[N-17]=运行中间变量栈顶

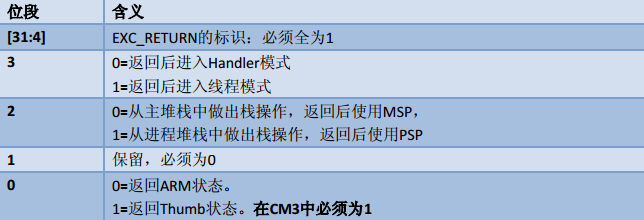
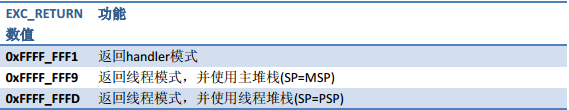
||

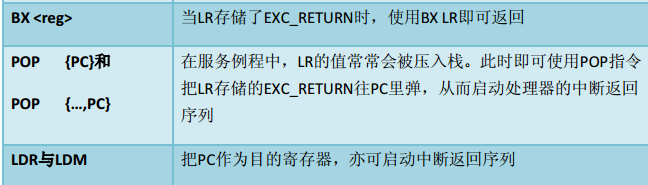
数组[0]= 运行中间变量栈底

**2 PendSV在Cortex-M3中的应用**

Systick为嵌入到内核中，优先级比一般中断优先级高。若在一般中断的ISR执行过程中，发生了Systick异常，则Systick会抢占该ISR。若此时Systick做上下文切换，在M3中将触发用法fault（在中断活跃时尝试切入线程模式）。即使在别的内核体系下不发生硬fault，ISR也会被延迟，这对于任一讲究实时性的系统是不能接受的。   
所以Systick只是在其服务程序中触发一个PendSV中断（事先将PendSV中断优先级设置的很低）。Systick返回后将继续执行被抢占的中断ISR。执行完后，程序跳转到PendSV服务程序中执行任务切换。

**3 M3中堆栈的切换**

3.1 在中断返回时，通过修改LR中的EXC\_RETURN   
3.1.1 EXC\_RETURN   
在出入ISR的时候，LR的值将得到重新的诠释，这种特殊的值称为“EXC\_RETURN”，   
在异常进入时由系统计算并赋给LR，并在异常返回时使用它。EXC\_RETURN的二进制值除了最低4位外全为1，而其最低4位则有另外的含义。   
表1 EXC\_RETURN位段详解   
   
表2 合法的EXC\_RETURN值及功能   


3.2 异常返回和异常返回序列   
M3 提供的异常返回指令   
   
出栈：先前压入栈中的寄存器在这里恢复。内部的出栈顺序与入栈时的相对应，堆栈指针的值也改回先前的值。   
只要ISR没有更改过CONTROL[1]，就依然使用发生本次异常的瞬间正在使用的SP指针来执行出栈操作。

3.3 修改EXC\_RETURN，进行模式和堆栈的选择。   
在异常ISR中，处理器处于特权模式，可以访问所有存储器（除MPU规定）。通过修改LR的值来达到不同模式和堆栈的切换。

**4、再论M3双堆栈机制**

已经知道M3堆栈分为MSP和PSP，CONTROL[1]决定如何选择。当CONTROL[1]为0时，只使用MSP，此时用户程序和异常handler共享一个堆栈。   
当CONTROL[1]为1时，线程模式将不再使用MSP，而改用PSP。这样做的好处在OS内核中防止用户程序 的堆栈破坏OS的堆栈。在在这种情况下进入异常的自动压栈使用的是进程堆栈，进入异常handler后才自动改为MSP，退出异常时切换回PSP，并且从进程堆栈上弹出数据

**5、理解M3中的R14**

1 在中断   
在出入ISR的时候，LR的值将得到重新的诠释，这种特殊的值称为“EXC\_RETURN”。在异常进入时由系统计算并赋给LR，并在异常返回时使用它。

2 在函数跳转   
当呼叫一个子程序时，由R14存储返回地址。

**6、任务中的PSP MSP**

1 、执行任务的时候用的是PSP(process stack pointer)

PSP堆栈指针在任务初始化的时候指向的是任务堆栈(任务堆栈数组)而不是CPU内核堆栈（启动文件中的stack区域）。

当任务运行时，任务堆栈用来保存一些局部变量(栈顶保存16个寄存器数据，局部变量从topstkptr-17开始)。

当任务挂起时，任务堆栈负责保存任务的运行现场(16个寄存器数据)，也就是CPU寄存器的值

2、执行其他中断的时候用的是MSP(main stack pointer)

MSP指向CPU内核堆栈，就是启动文件里面分配的堆栈大小区域内存。