# 嵌入式系统实验报告



|  |  |
| --- | --- |
| 实验名称： | CPU 异常处理与上下文切换 |
| 姓 名： | 陈姝仪 |
| 学 号： | 2018211507 |
| 学 院(系)： | 计算机学院 |
| 专 业： | 网络工程 |
| 指导教师： | 刘健培 |

2020年12月31日

# 实验目的

* 了解开发环境的使用。
* 学会通过查阅文档和数据手册获取信息。
* 掌握 cortex-M4 体系结构中寄存器和异常的使用方式。
* 掌握基本的软件编写与调试方式。
* 理解处理器异常上下文切换的实现方式。
* 理解多任务的实现方式。

# 实验环境

* FS-STM32F407开发平台
* ST-Link 仿真器
* RealView MDK5.23集成开发软件
* 串口调试工具
* PC机Window7/8/10 (64bit)

# 实验要求

* 基本要求
* 在主程序中用svc指令触发SVC异常
* 编写SVCall异常处理程序，打印异常发生前的处理器现场状态（即寄存器R0-R15、XPSR）以及异常发生后发生变化的寄存器（R13/SP、R14/LR/EXC\_RETURN、R15/PC、xPSR、CONTROL），据此分析异常发生前后处理器分别处于哪种模式（handler or thread）、使用哪种栈（MSP or PSP）、特权等级（特权与非特权）
* 扩展要求
* 使用svc异常模拟系统调用，实现函数间上下文切换功能
* 如：func1->context\_switch->func2->context\_switch->func1

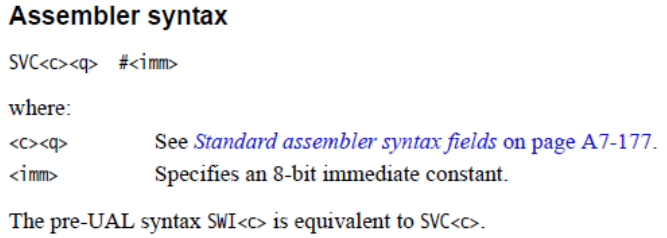
# 实验原理

* **使用svc系统调用指令出发SVCall异常**

在eclipse的gcc编译环境下，可用C嵌入汇编如下：

**#define SVC\_CALL**() **asm volatile** ("svc 0")

指令格式：



* **Cortex-M4处理异常的流程**

异常发生后，需要软件和硬件协作处理。

进入异常时，硬件会切换模式，并保存必要的寄存器。然后异常处理程序（软件）再根据需要保存需要的额外寄存器。

保护好现场后，就可以进行额外的处理。此时可以进一步准备好调用 c 函数需要的堆栈，然后调用 c 函数进行进一步处理。

退出异常时，流程基本与进入异常时相反，首先软件需要恢复部分寄存器，然后通过机器指令让硬件恢复之前硬件保存的寄存器，并跳转到异常发生时的地址继续执行（如果需要）。

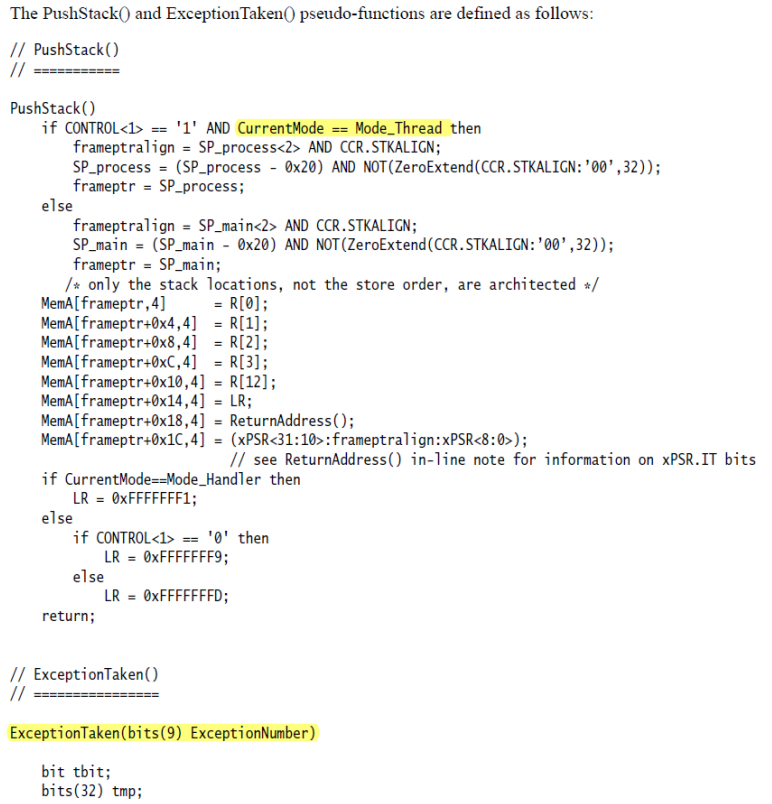
* **“现场”的含义及需要保存的内容**

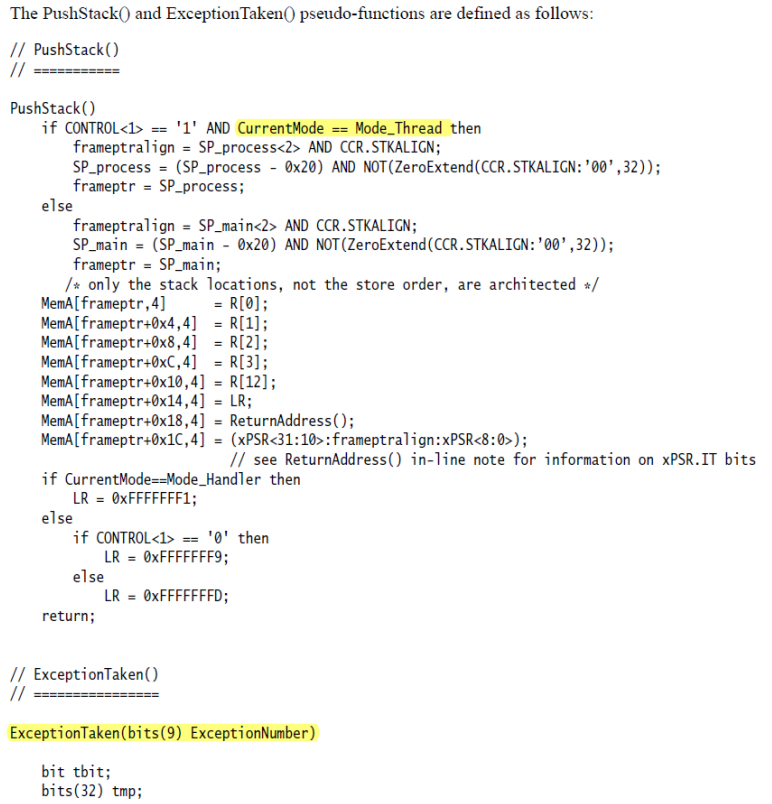
一般而言，现场指的是处理器的核内寄存器。

因为当发生异常时，处理器需要运行另外一段程序，这段程序也需要使用处理器，所以必须先把异常前处理器中已有的寄存器中的数据保护起来，相当于创建一个“还原点”，然后在处理完后，在将之前保护的数据恢复到处理器中，从而从“还原点”继续往下执行原程序。

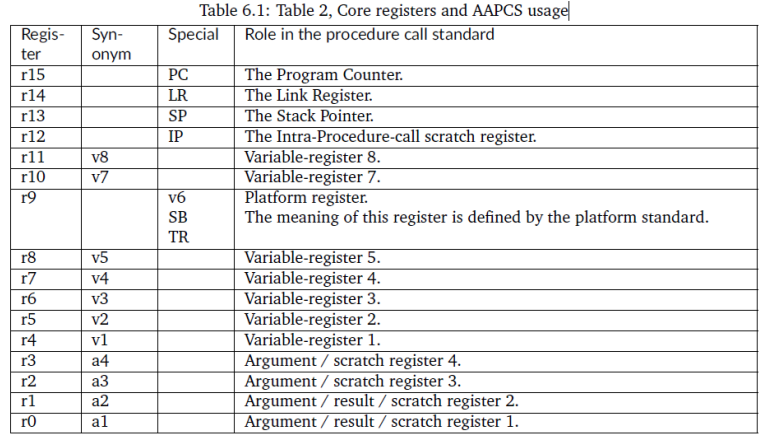
为获取异常发生前的处理器现场状态，需要知道 arm 在进入异常时，保存了哪些寄存器（意味着这些寄存器是我们在异常里写代码时可用的），保存在了哪个位置（据此才能获取 arm 保存的值）。然后在我们的代码破坏 arm 未保存的值之前，先把相关的寄存器的内容保存下来，然后准备好 c 的调用环境，就可以调用 c 函数进行处理了。

* **进入异常时，硬件保存的寄存器**





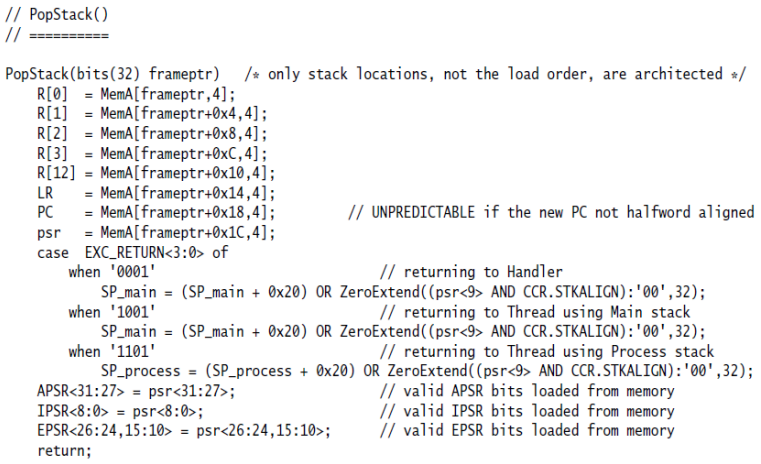
* **准备好调用C的堆栈环境**



一般，编译器处理器 CPU 中的寄存器时有 3 类主要的用法，1 类编译器不会使用的，主要是一些特殊寄存器（如调试寄存器、特殊功能寄存器等），第 2类是需要调用者保存的—即 callee 认为 caller 会保存从而 callee 可以不保存而使用，如 R0-R3 R12 这些 scratch 寄存器，第三类是被调用者 caller 会保存的，如R4-R11 这些变量寄存器和 LR 等。

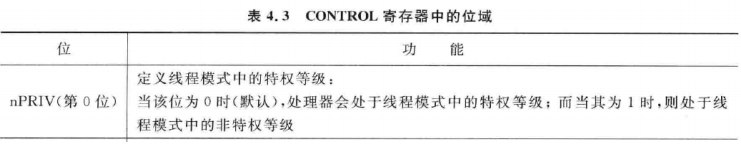
所以，我们作为 caller，需要保存 R0-R3 R12。

* **退出异常时发生的过程**



* **得知异常前后处理器的模式、栈、特权等级**

CONTROL寄存器保存了线程模式的特权等级：



EXC\_RETURN寄存器保存了异常发生前的模式和栈类型：

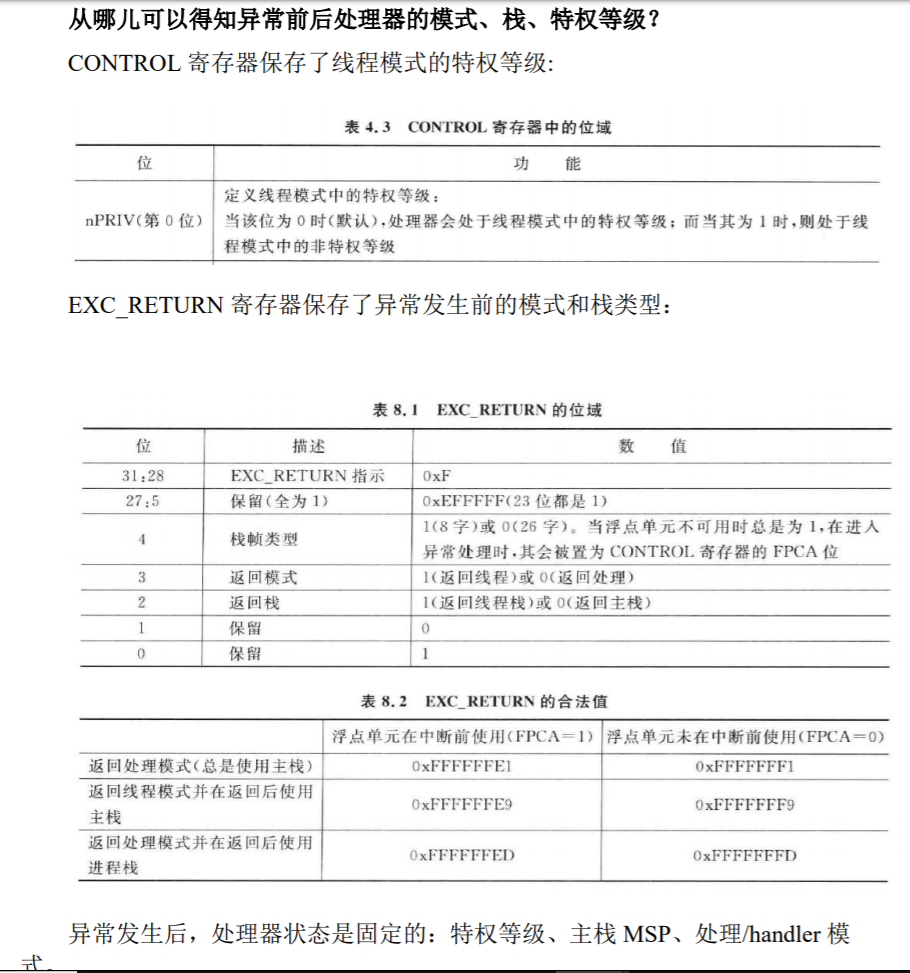


异常发生后，处理器状态是固定的：特权等级、主栈MSP、处理/handler模式。

* **通过SVC系统调用实现函数上下文切换**

资料《ARM Cortex-M3Cortex-M4 权威指南.pdf》中 10.5 节使用 pendSV 实

现了任务的上下文切换，使用 svc 指令是基本类似的：



# 实验步骤

## 基本部分

* **分析函数调用的现场保护和恢复**

1. 下载并打开老师给的代码emlab2020-lab1.rar
2. 连接好实验板的相关线路，打开电源，打开串口工具并打开响应串口。
3. 在lab\_main.c中找到实验入口lab1\_a\_main()，设置断点，build程序，然后用debug模式运行程序，从这个函数开始一步步跟着程序走，观察程序的执行过程。
4. 对照实验指导书和SVC\_Handler函数分析函数调用和退出时是如何保存和恢复现场的，保存了哪些寄存器，调用前后哪些寄存器发生了变化。

对照实验指导书，知道了CONTROL寄存器第0位保存了线程模式的特权等级，EXC\_RETURN寄存器第3、第2位分别保存了异常发生前的模式和栈类型。从frame中取，若CONTROL寄存器第0位为0，则线程模式是特权等级，为1则为非特权等级；若EXC\_RETURN寄存器第3位为1则处于thread模式，为0则处于handler模式；若EXC\_RETURN寄存器第2位为1则为PSP，为0则为MSP。

## 附加部分

1. 打开项目中的lab1\_b.c和task\_co.c，分析代码，分析任务之间通过主动调用svc现场切换系统调用让渡cpu的过程。
2. 通过分析take\_change\_state\_to函数，发现任务一共有4个状态：TASK\_DEAD,TASK\_RUNNING,TASK\_READY,TASK\_BLOCKED。根据blocked状态中，向ready和dead状态转移都需要将任务从wait\_queue中取出来，running状态转到ready状态要把任务加到ready\_queue中，转到blocked状态要把任务加到wait\_queue中，因此对应的判断，ready状态转到blocked状态也要把任务加到wait\_queue中，ready状态转到其他状态要把任务从ready\_queue中移除。
3. 根据分析填充task\_change\_state\_to函数中TASK\_READY情况的代码即可。

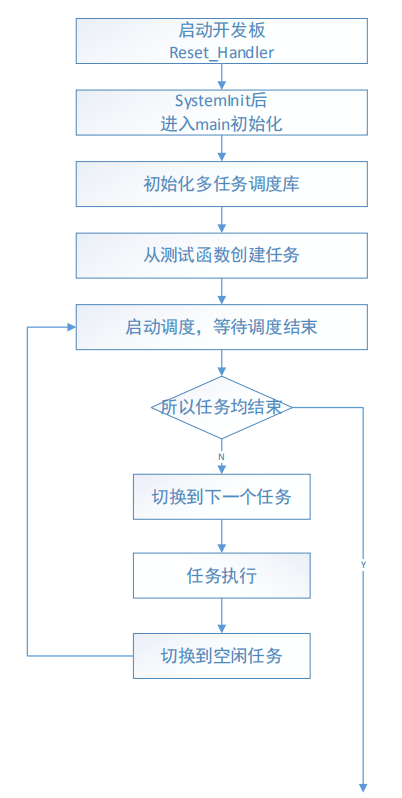
# 实验方案与实现

## 软件结构

* SVC异常处理的程序流程如下图所示：

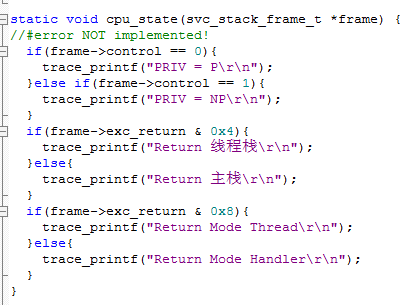


* SVC上下文切换的程序流程如下图所示：

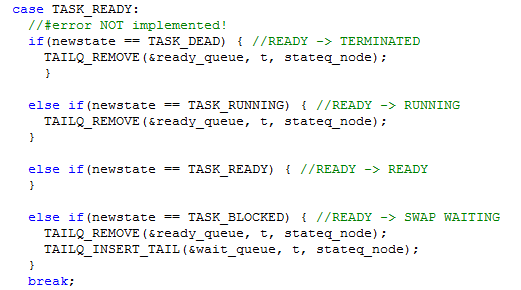


## 源代码

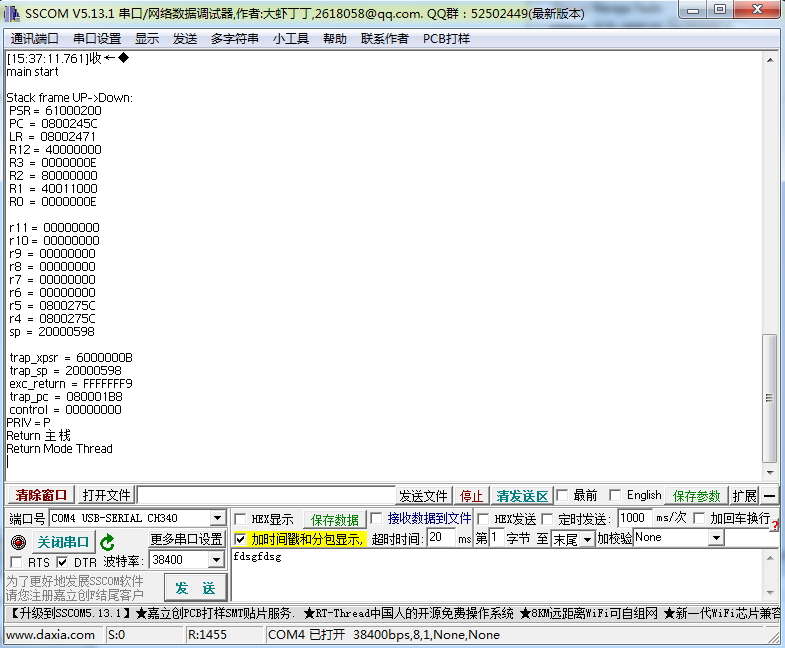
Lab1\_a:



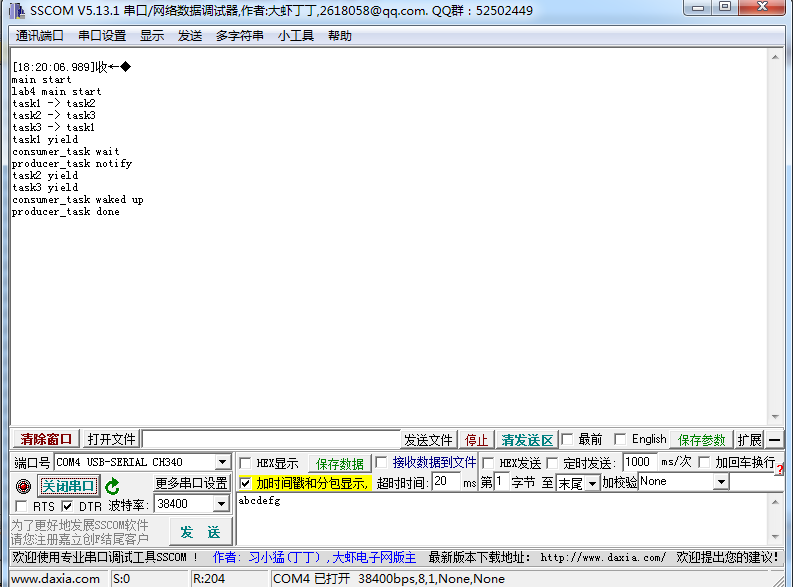
Lab1\_b:



# 实验结果与分析

Lab1\_a:从图中可知，特权等级为特权，栈为MSP，模式为线程模式

Lab1\_b:



# 实验总结

第一次做嵌入式实验，有点无从下手，后来认真看了实验指导书后有了些眉目。实验的关键在于根据实验要求，明白cpu\_state函数的作用，以及学会查找指导手册获取自己需要的信息，并据此获得对应的信息用于判断处理器分别处于何种模式（handler or thread）、使用何种栈（MSP or PSP）和何种特权等级（特权与非特权）。

在b部分的实验中，发现嵌入式系统的状态切换和我们在《操作系统》课上学的不太一样。Ready状态之后，每个状态都有可能出现，而不是像操作系统的状态转换图一样ready后只有running。