# **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： | 李志毅 |
| 学 号： | 2018211582 |
| 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 29 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 SVCall 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**

**如何触发 SVCall 异常？**

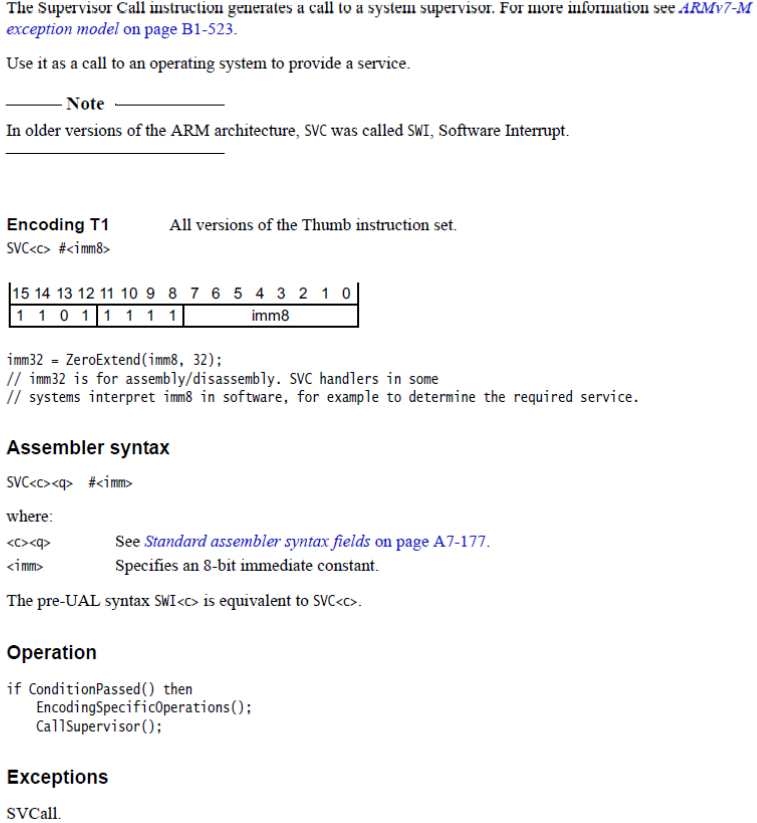
使用 svc 系统调用指令。

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**")

指令格式见手册：

《DDI0403E\_d\_armv7m\_arm Arm®v7-M Architecture Reference Manual.pdf》



**Cortex-M4 处理异常的流程是怎样的？即进入异常和退出异常时处理器做了什么？**

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”到底指什么？需要保存哪些内容？**

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。

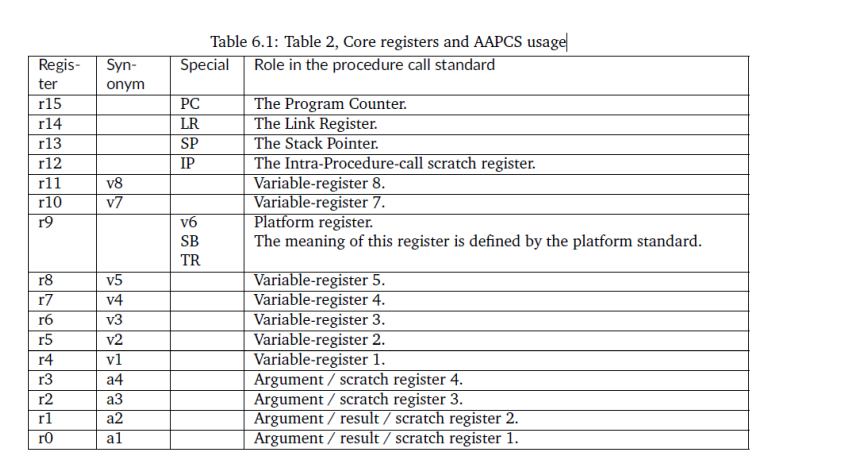
**进入异常时，硬件保存了哪些寄存器？**

查手册《ARMv7-M Architecture Reference Manual.pdf》



**如何准备好调用 C 的堆栈环境？**

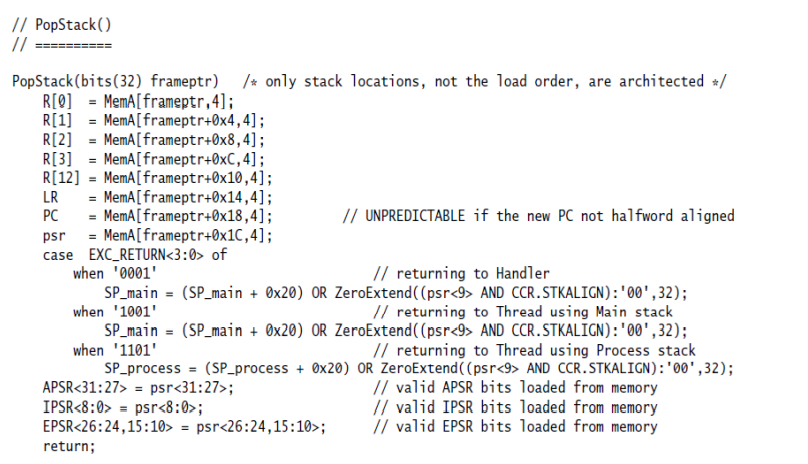
函数调用保存哪些寄存器是有约定的（ARM ABI AAPCS），见资料《aapcs32 Procedure Call Standard for the Arm® Architecture — ABI 2018Q4 documentation.pdf》

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。

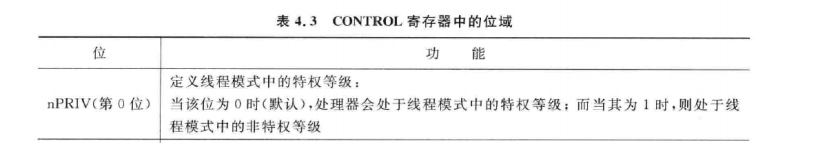
所以，我们作为 caller，需要保存 **R0-R3 R12**。

**退出异常时发生了什么？**

查手册《ARMv7-M Architecture Reference Manual.pdf》

**从哪儿可以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:



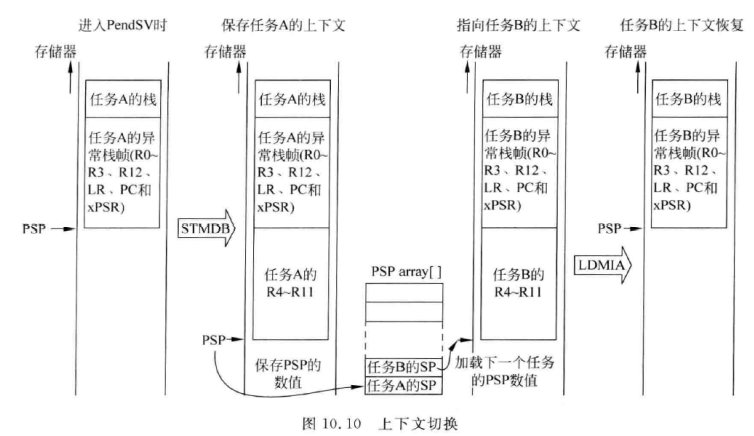
EXC\_RETURN 寄存器保存了异常发生前的模式和栈类型：



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。

**如何通过 SVC 系统调用实现函数上下文切换？**

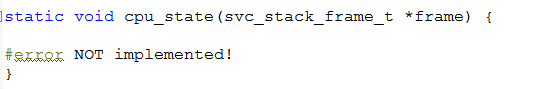
资料《ARM Cortex-M3Cortex-M4 权威指南.pdf》中 10.5 节使用 pendSV 实现了任务的上下文切换，使用 svc 指令是基本类似的：



# **实验步骤**

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、 41 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）

给出的示例代码中缺少对于cpu\_state(svc\_stack\_frame\_t \*frame)函数的编写，在该函数中添加相应代码即可实现对异常发生前后的处理器现场状态打印

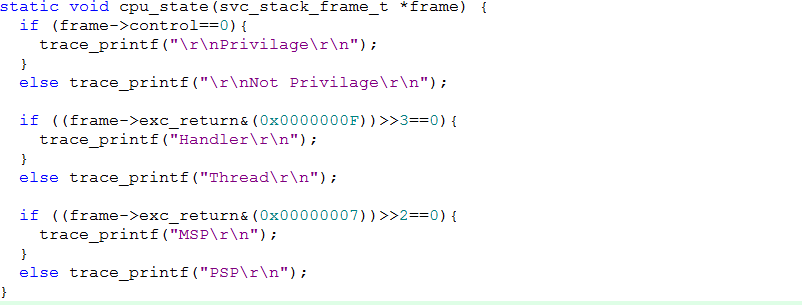


# **实验方案与实现**

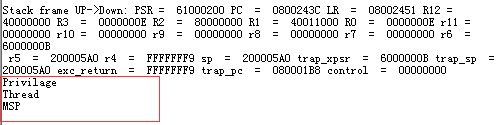
## 软件结构

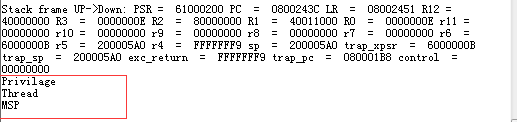


## 源代码



# **实验结果与分析**





实验结果截图如上，可以看出触发异常时，其处理器等级为特权级，返回模式为线程模式，使用主栈MSP

# **实验总结**

通过该实验，加深了我对于cortex-M4体系结构中寄存器和异常的使用方式的理解和运用，掌握了基本的软件编写和调试方法，理解了处理器异常上下文切换的实现方式和多任务的实现方式。