工作笔记

根据应用内容，简化

tangyapeng

# \*\*\*\*\*\*\*\*\*\*star-gather项目学习笔记\*\*\*\*\*\*\*\*\*

## 远程桌面登录信息

• 使用windows远程桌面，一般使用自己账户登录远程桌面：116.113.133.5:19014

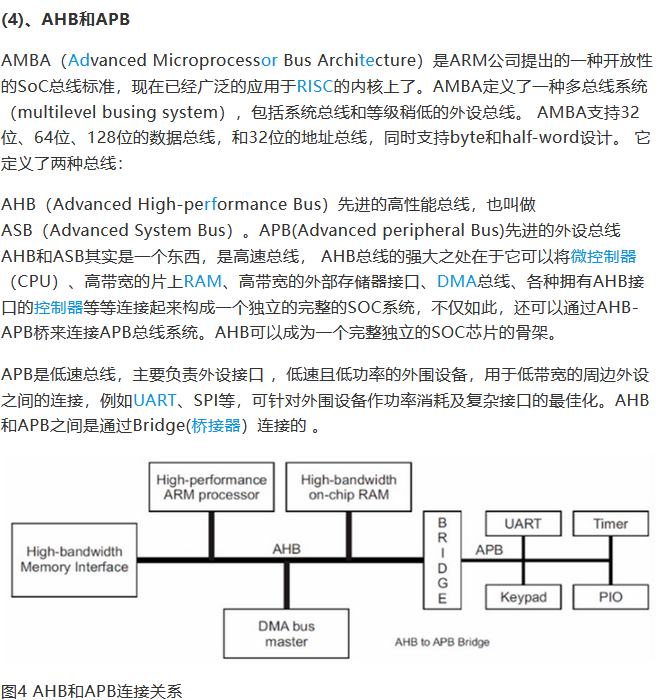
• 用户名：star-gather\<姓名的全拼>

• **密码：Typ@24061729 (其他的密码也是用这个)**

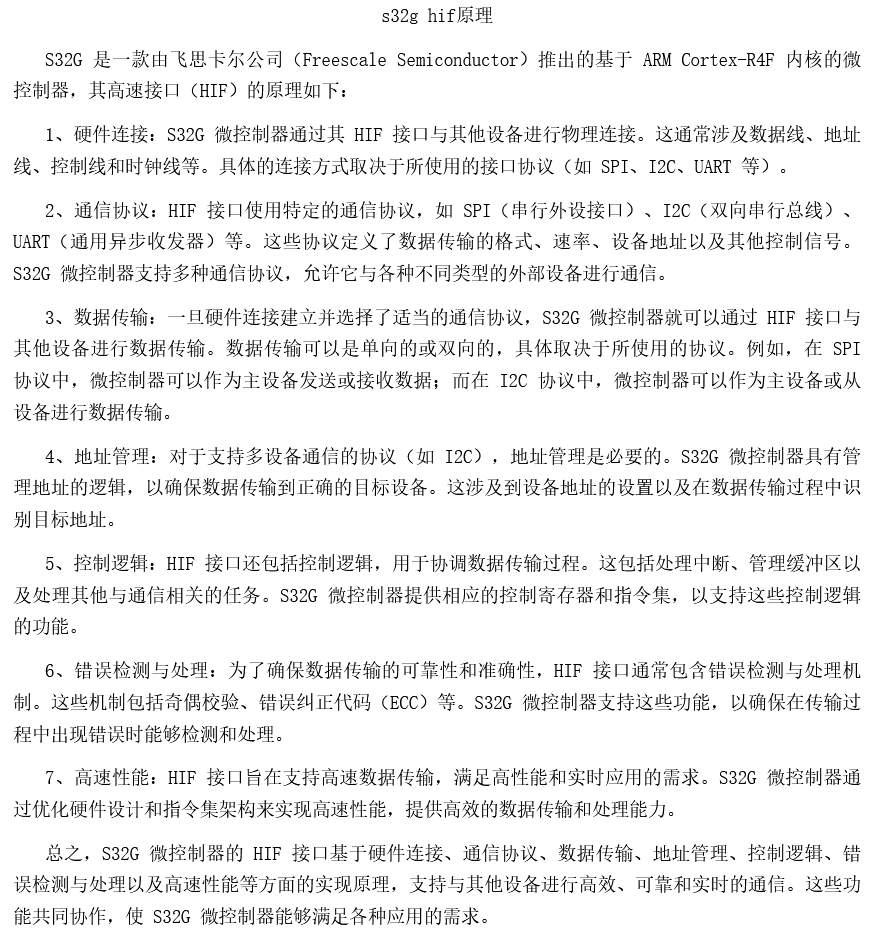
## 代码阅读笔记

C1\_Software\_Architecture\_Spec.docx 为顶层设计文档。

## 关于Cortex系列的总线



## HIF高速接口interface



## IPC进程间通信

综合介绍几个方式。



## MPECAN四个核心

Core0：发送及配置信息的处理，当然底层配置信息是写入到配置寄存器中的；

Core1：传输协议处理。

Core2：接收数据流的处理；

Core3：CAN帧格式转换器；

**章节8.4.1.1.**

其他：再host总线上具有多个FIFO、Share memory用于数据交互。

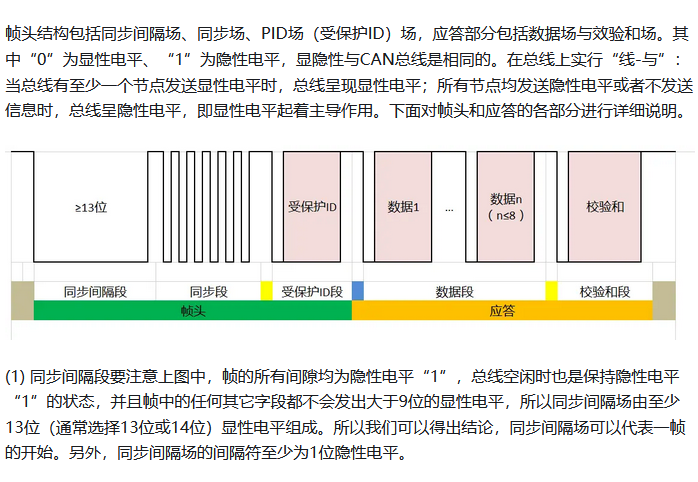
## 几个疑问

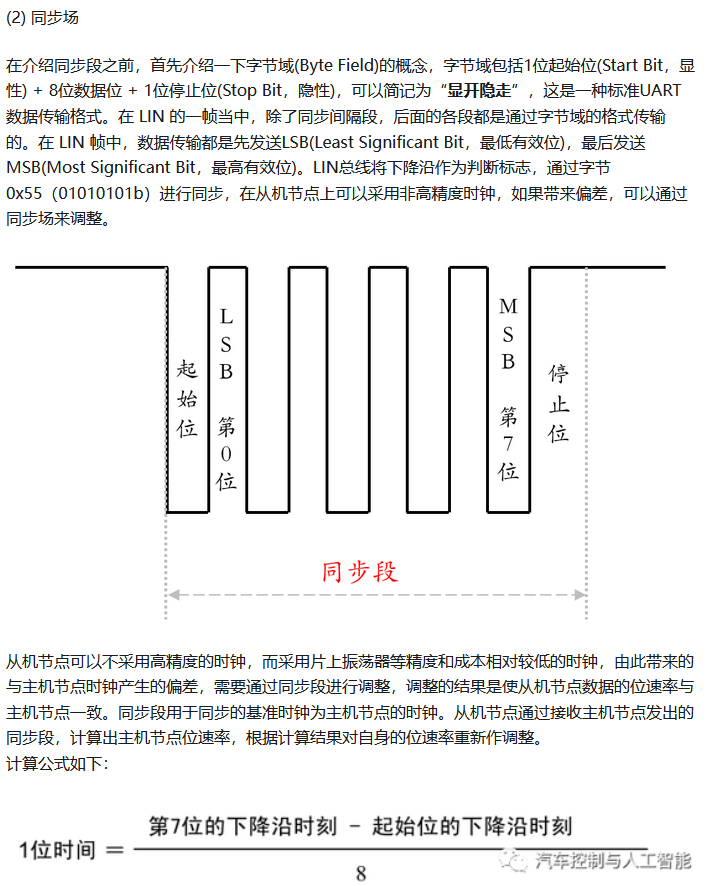
1. HOST和MPECAN之间交互接口有哪些？协议是什么？有哪些指令和内容？
2. MPECAN FW要实现的功能有哪些？

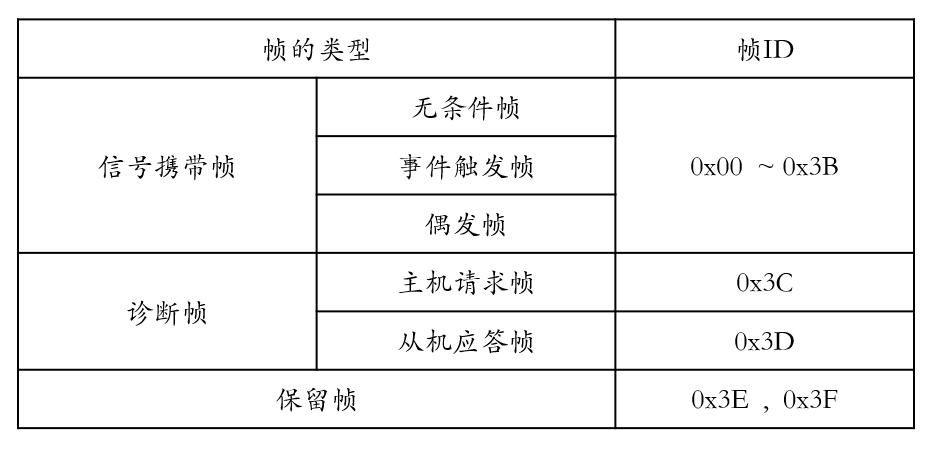
比较繁杂

## Lin总线

一主多从的总线结构，点名通信的方式。单总线节点个数限制16（1主15从），速率20Kbps。







一般而言，车内会选择统一字节数，最常用的是每帧传递8个字节。没有数据长度段。

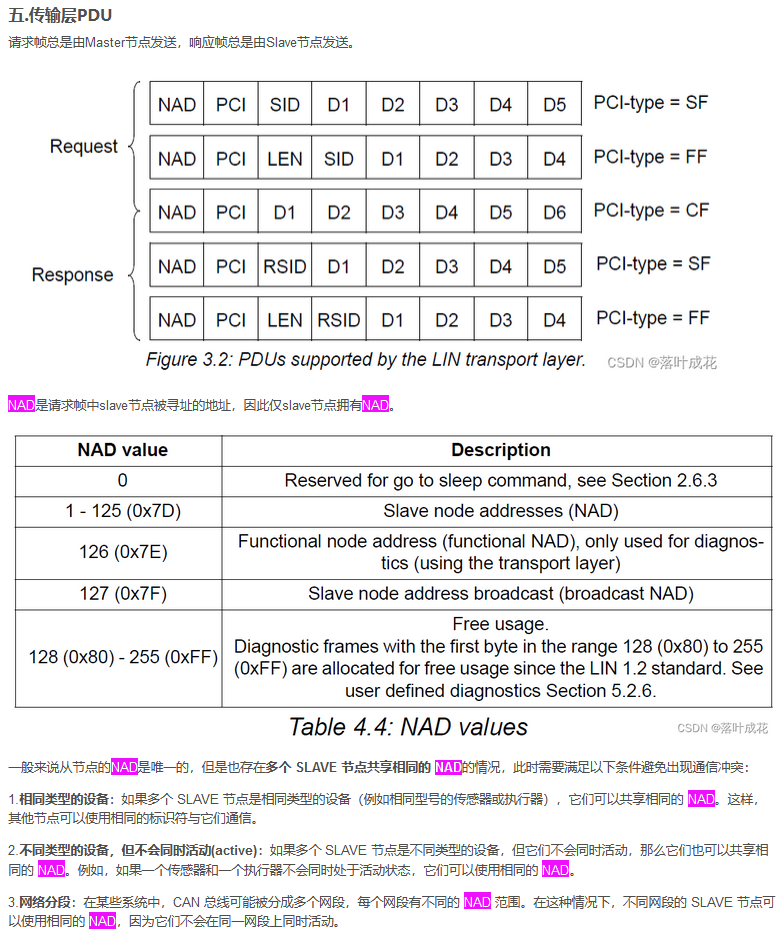
LIN帧类型

1. 无条件帧

这个帧类型类似于点名某个节点，把当前的帧空间留给该节点来进行报文的发送。此时，其他的节点可以接收来自该节点的报文信息（包括主节点）。

如何点名？

1. 事件触发帧
2. 偶发帧
3. 诊断帧 用于配置、识别和诊断。
4. 保留帧 后续扩展使用。



## LIN传输层知识

核心是了解 传输层完成地址绑定。

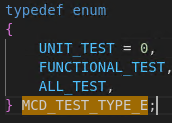
## MCU测试接口代码阅读

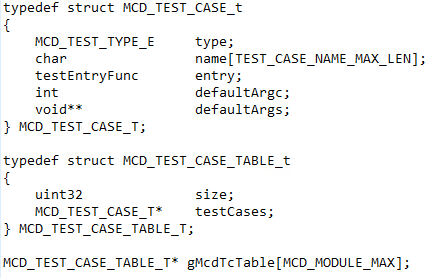
实现架构为：PC端运行脚本或者直接发送串口命令到目标机，目标机接收到指令后进行指令的解析处理，然后按照指令要求调用事先注册的函数，这些函数都是测试程序集。

程序实现逻辑：

1. 平台、外设等初始化；
2. 注册测试程序集；
3. 根据宏定义，按照命令执行对应的测试项，或者全部执行；执行测试项时候调用简单的log函数，可以根据配置为存储到ram或者通过uart存储。

//记录一些必要的数据结构

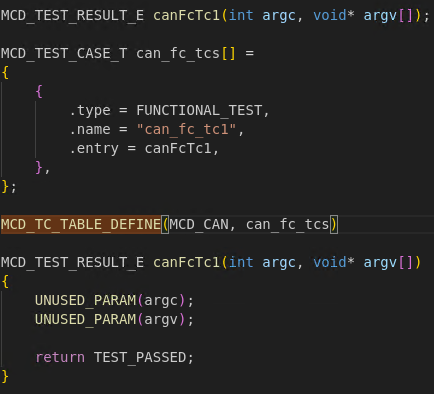






测试项的实现方式：

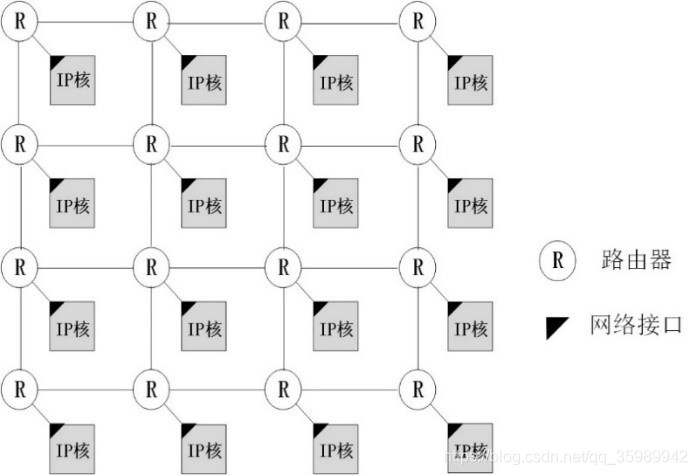
通过该函数进行调用，创建对应的全局结构体，作为某个外设的测试项列表结构体变量，



## NoC （Network-on-Chip）

片上网络，其结构组成如下所示：区别于

传统的总线型或交叉开关（crossbar）等互联结构有可扩展性差、带宽较低、延迟较大和功耗较高等缺点



# \*\*\*\*\*\*\*\*\*\*MPECAN 代码相关 \*\*\*\*\*\*\*\*\*

## Test

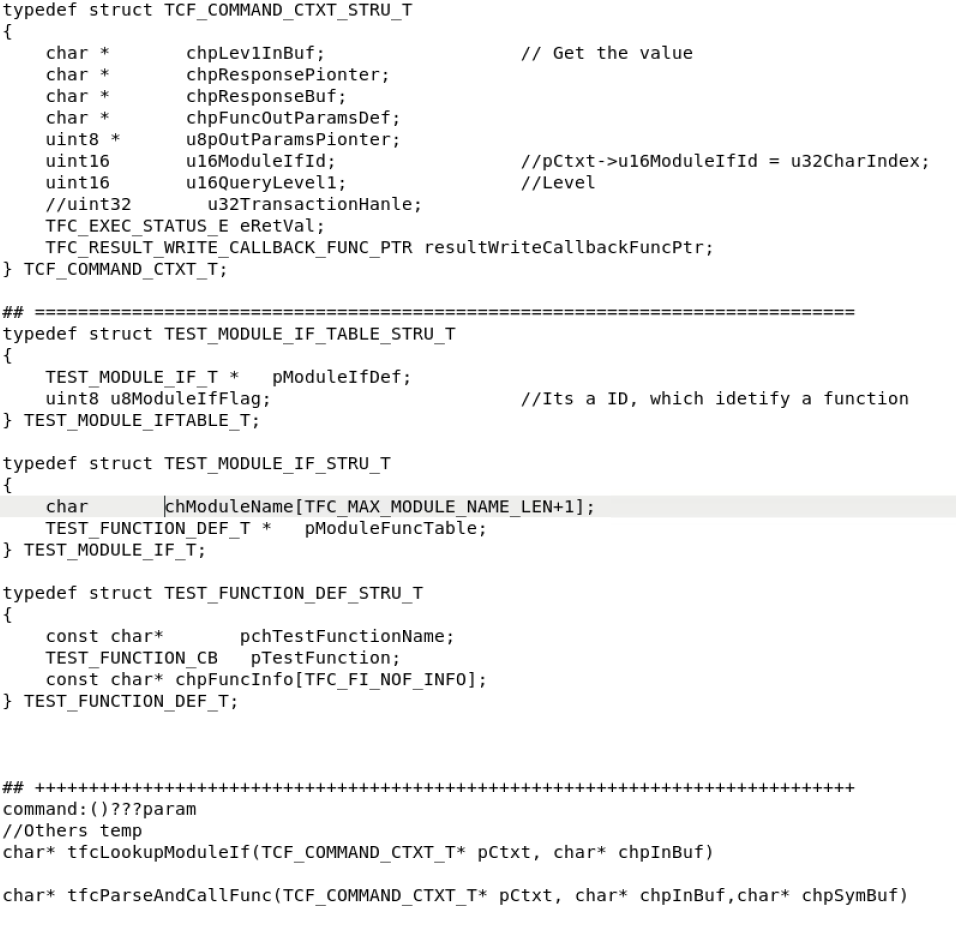
测试框架：

PC端运行python测试脚本，控制串口进行测试指令的发送；

下位机（C1）进行测试指令的接收、解析、处理过程；

下位机中，MCU完成：LIN的初始化配置，core\_dst完成串口接收，指令解析工作。

几个用到的核心数据结构体：（完善中）



该部分的核心是：

通过串口接收字符串，在接收中断中，check buffer，解析出测试指令、测试层级等信息，然后在全局测试程序集中调用对应的函数执行调用，中间涉及到name匹配等操作。

## Host and MPECAN之间的交互都是通过中断函数实现执行序列的

1. 通过指令FIFO中断执行命令中断服务函数；
2. 通过IN、OUT FIFO中断，执行对应的数据收发的中断服务函数；

## Host and MPECAN interface data structure

CMD FIFO用于传递CAN控制器通道ID。

Host和MPECAN交互使用FIFO和share memorys，其中，memory结构体名称为：



MPECAN\_CAN\_SHARE\_MEMORY\_TYPE\_T;

目前需要记得几个核心结构有：

全局数组gChangelMaxTxMBCount：保存不同CAN控制器对应的最大传输帧数，大概是每次发送的帧数，后续再详细确认。

**Host对MPECAN通信而定义的命令**：有命令号、命令参数（定义成不同的数据结构体，采用union方式存储），返回值（用于Host读取返回值）；

**接收消息缓冲区描述符RxMB：**filterID/搜索器搜索结果等

**发送消息缓冲区描述符TxMB：**

**CAN报文缓冲区**：主要是Index字段用于检索。

*关于具体配置、操作的寄存器等相关的细节，暂时不深究，后期遇到问题再钻研。*

## Host and MPECAN LIN模块之间接口

MPECAN CAN模块和LIN模块采用不同的指令和数据通讯方式，可能是方便进行通道控制等；

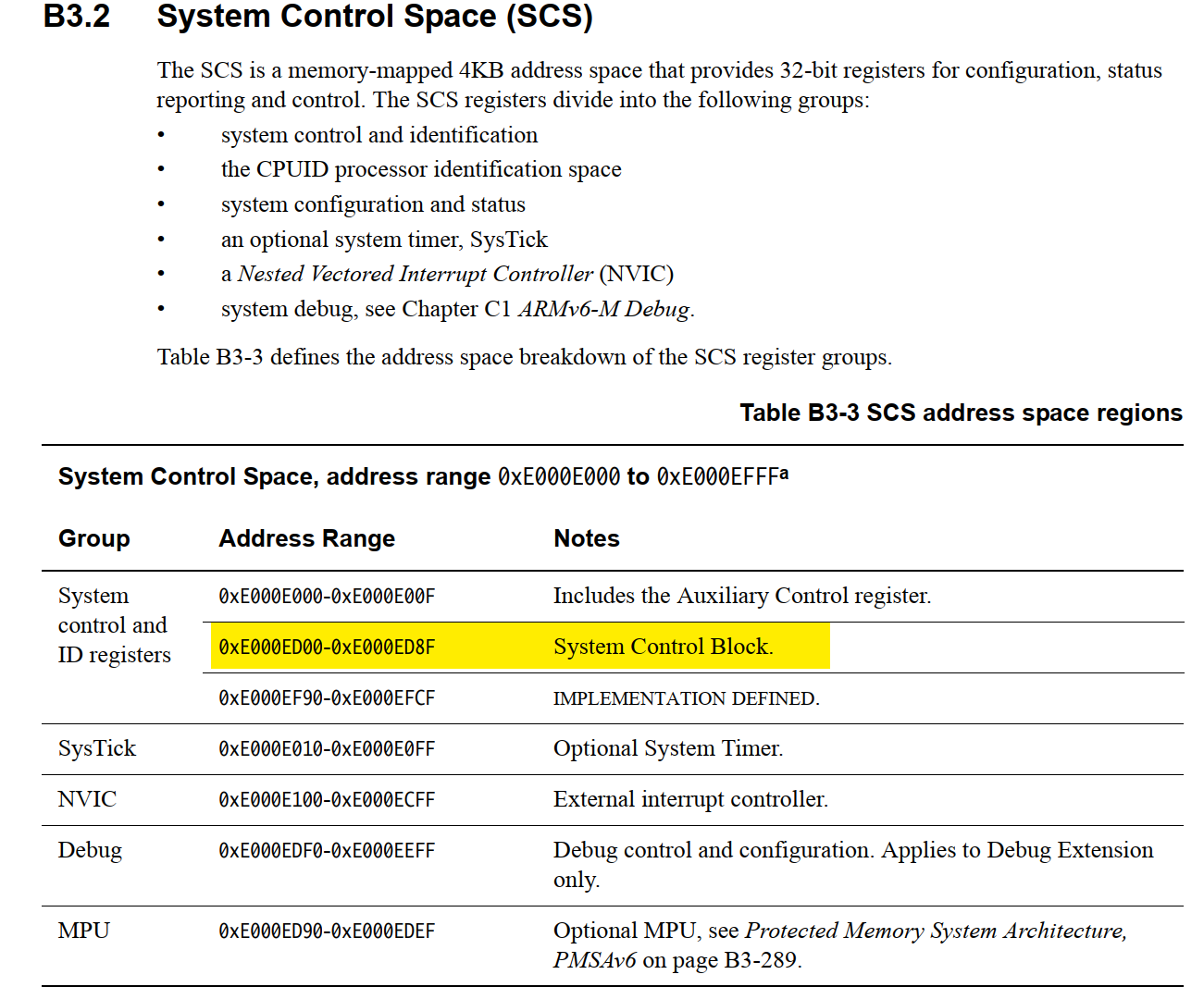
Lin总线与host之间采用core2core事件同步，数据和指令的传输采用share memory通讯。

其中core2core模块每个core之间有16个事假可以选择，目前的用法是标志为4个不同的lin通道。

## FIFO 101/102用作HOST and MPECAN commucation port

并且该fifo写入数据触发中断，中断由中断集中器转到core\_DTX.

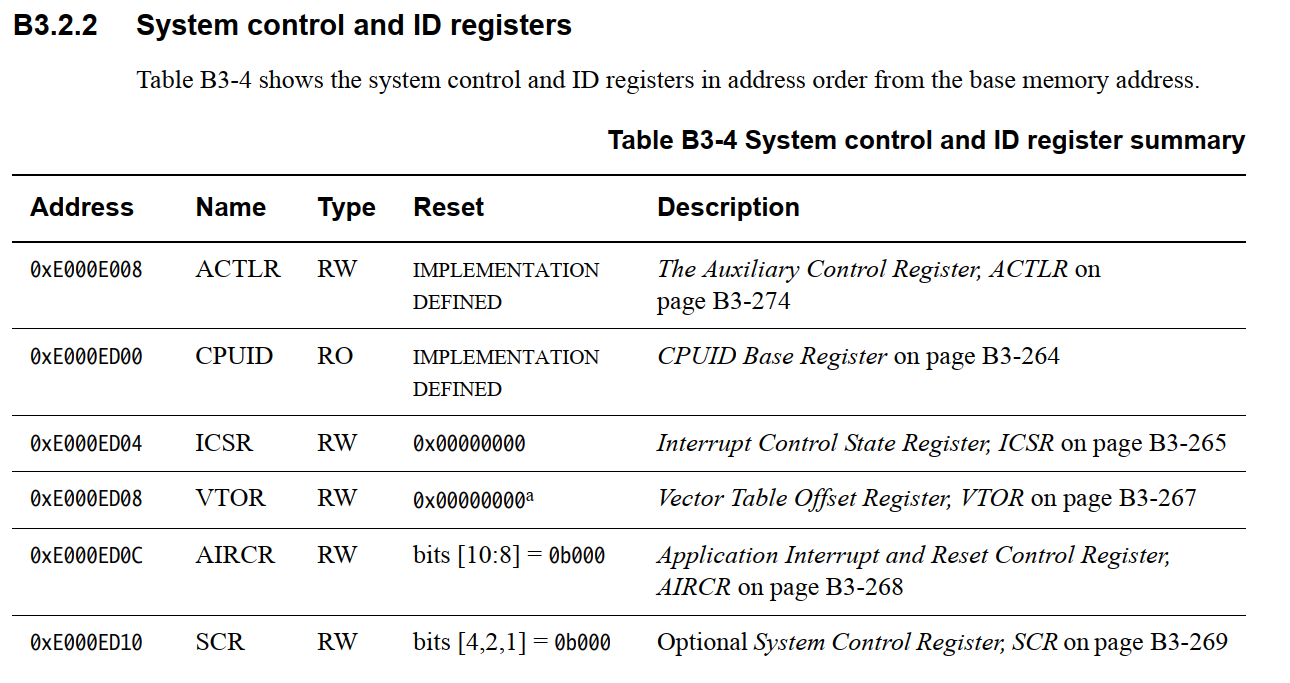
## cortexM0+(armV6m)System control space

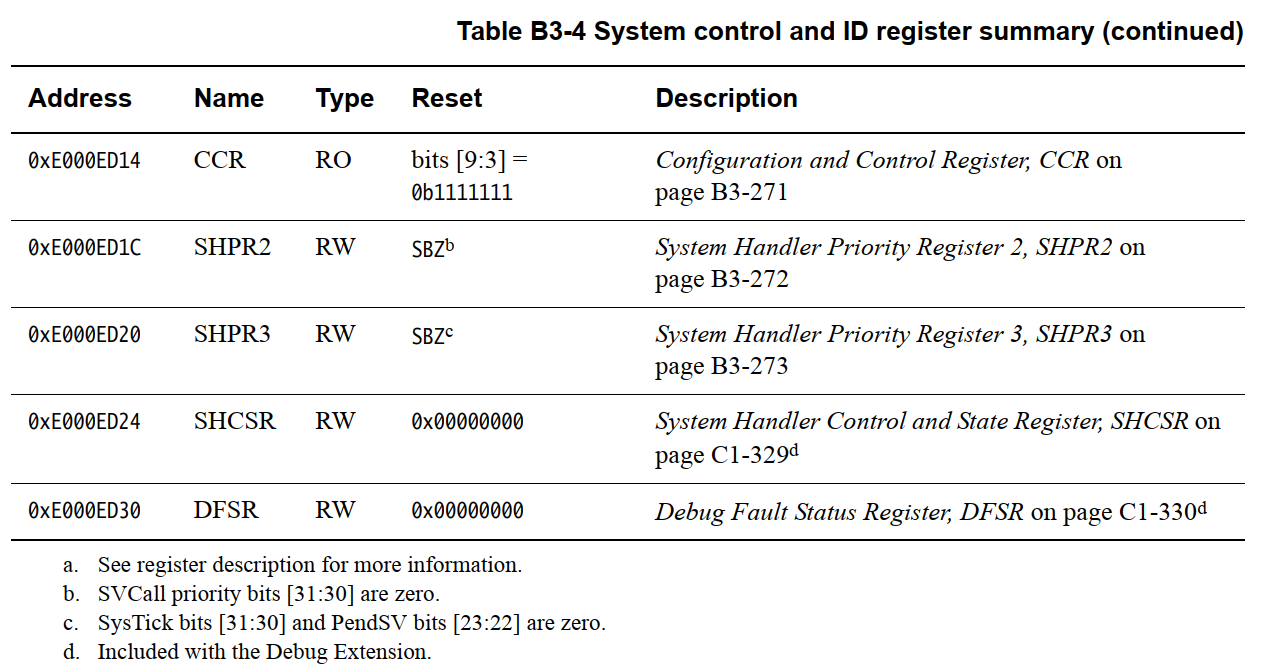


This excludes external interrupt handling. The NVIC handles all external interrupts.

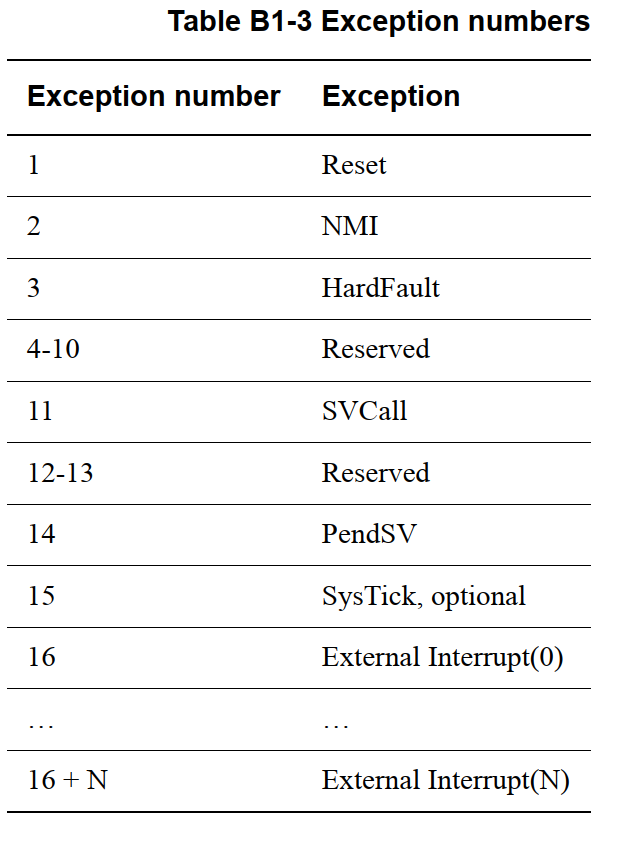
• The exception number of the currently executing code and of the highest priority pending exception

• Miscellaneous control and status features





## cortexM0异常号/中断号



Stm32F4的中断配置表格，可以参考英文手册的P374.

目前对于MPECAN的中断号定义，没有找到相关出处。

## 代码中关于中断的处理逻辑

* 1. 在启动代码定义的中断向量表中，各个中断的调用函数为统一的接口函数，即：cm0InterruptHandler()；
  2. 如本章节的3小节的说明，在ICSR寄存器中，可以读取当前异常号，异常 号和中断向量表是对应的；
  3. 代码中，代用全局数组的方式把异常号和中断服务函数关联在一起，在任意中断发生时候，进入到中断服务函数cm0InterruptHandler(),该函数的执行序列是这样的：首先读取ICSR寄存器，获取异常号，作为参数，获取对应的中断服务函数；
  4. 中断函数的定义和注册，使用接口INSTALL\_ISR()

## 检索中断号

搜索：

INSTALL\_ISR关键字。

# \*\*\*\*\*\*\*\*\*\* 嵌入式知识 \*\*\*\*\*\*\*\*\*

## LD链接脚本基础知识

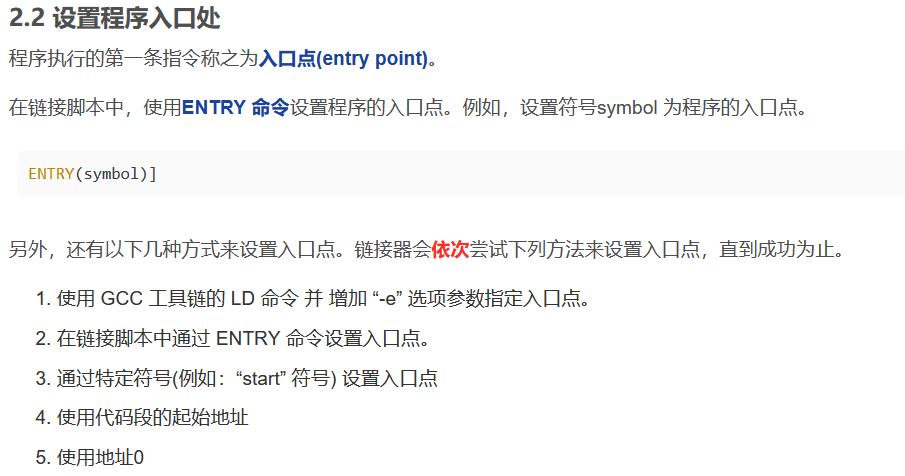
因为最初直接用的机器加载，所以习惯叫loader。

arm-linux-ld -Ttext 0 start.o sdram.o -o sdram.elf

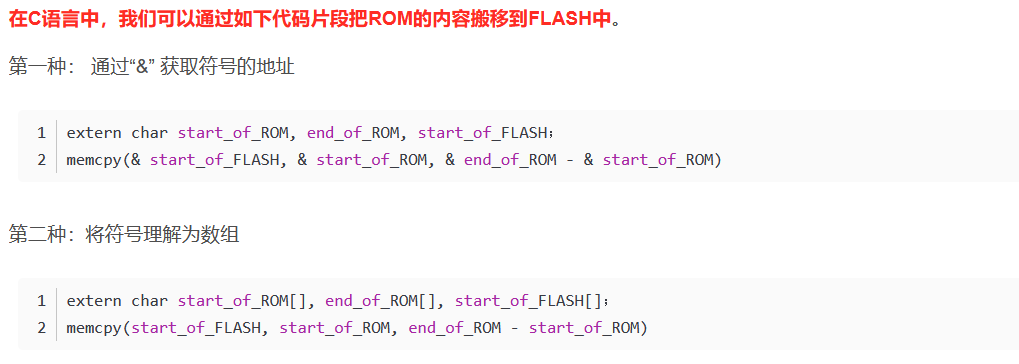


联想 程序员的自我修养 中关于elf中的段的相关知识，最终程序需要知道代码段、数据段、bss段等段所在的地址及地址符号表以及其他的符号表，基于这些目的，进行考虑。

**设置程序的入口地址，根据优先级，可有如下顺序：**



**符号引用，采用 “ = ”来赋值符号，并且该符号作为地址符号可以进行运算；并且符号作为符号表存在于elf文件中，可以在代码中引用，应用方法有如下两种：**



**输入段，一般为自定义或者生成的\*.o文件的各个段组成，输出段，一般为可执行文件或者库文件的总和段。**

**两个例子：**





Linker script中的常用命令：



新增：

INCLUDE命令，一如其他脚本的应用。

NOLOAD命令， 表示在程序加载时候当前段不被加载到内存中。

简单应用总结：

链接器用在链接阶段，用于把编译、汇编后生成的目标文件整合到一个统一的text段、data段、bss段，自定义段、调试相关的段等，也包括用户对内存的分段定义，地址定义，用于生成符号表，以在源码中应用之。

一般是先用MEMARY命令标识当前设备的内存地址等信息，然后执行SECTIONS命令，以分配各个段。并且，当加载地址和存储地址不同时候，可以用AT命令引用ram地址的方式来实现嵌入式的程序加载。

优点：

链接器可以在内存部分耗尽时自动检查，并以错误结束

我们不需要将点指针保持在正确的值

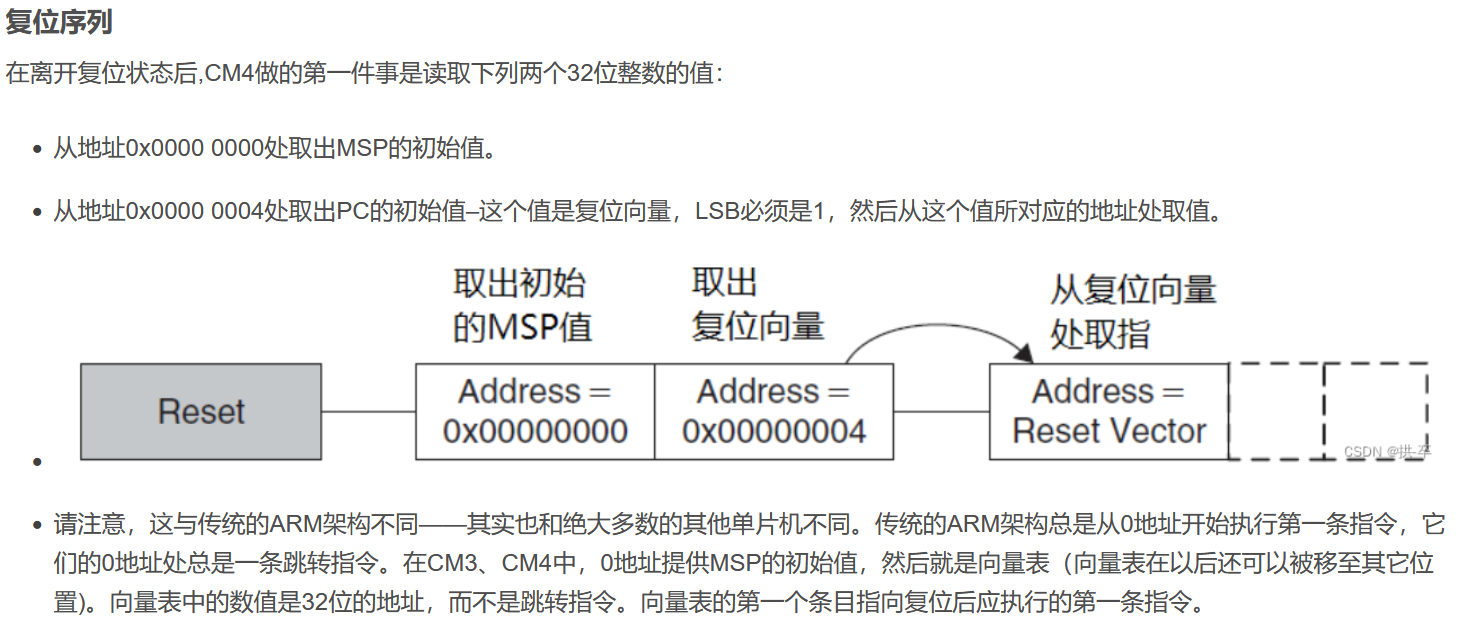
链接器脚本会自动解释代码的位置

新版本的 ld 可以以用户友好的方式显示所有内存区域的内存消耗

## ARM启动代码理解

1. **单片机上电、程序启动过程**

目前已知cortex-M3/M4/M0单片机的起始地址是以Stack Top开始的，存储的是栈顶地址，紧接着第二个地址存储的是reset\_handler的地址。该系列单片机的启动方式为：



在reset代码中，可以执行可执行文件的拷贝命令等操作，具体后续遇到再完善；另外，reset还执行未初始化变量的clear操作等。

通过reset函数，最终可以通过调用C库的\_start符号最终调用main函数，或者可以直接修改启动改吗中reset\_handler直接调用自定义的函数接口。（问题：*C库中的调用过程是什么？*）

（PS：在不应用C语言时候，也就是不使用栈顶指针，这时候可以不用初始化MSP。因为C语言的函数调用的本质就是基于堆栈空间的各种入栈、出栈操作；当然，这也要求汇编中不能使用堆栈相关的命令。）

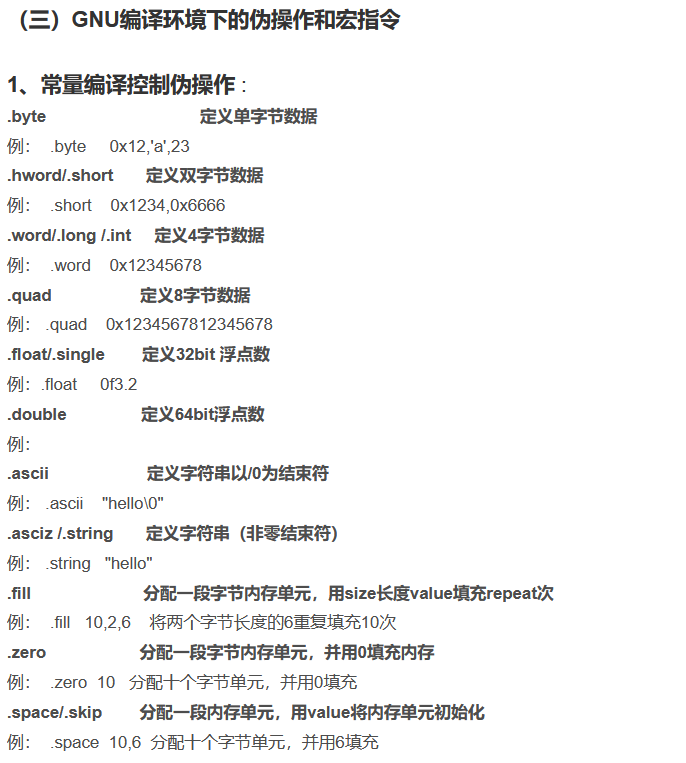
也就是在起始阶段写入了栈顶指针，该符号的值一般在启动代码中定义或者在链接脚本中定义。

那么堆空间呢？

堆空间是程序设计者通过函数动态引用的内存空间，所以只在运行时会用到，单片机硬件是不用感知堆空间的存在的。

1. **启动代码的作用及与链接脚本之间的关联**
2. 启动代码可以认为和其他生成的目标文件类似，具有各种段，最终通过链接脚本进行最终的链接，生成text、data、bss等段。根据cortexm3等系列单片机的启动过程，需要在链接脚本中更改vector段放在最开始的位置，以执行上电启动过程。
3. 代码中自定义的一些内存段符号表可以在链接脚本中定义，方便在源代码中引用，并在链接时候链接到对应的地址空间。
4. 其他：待查询完善；

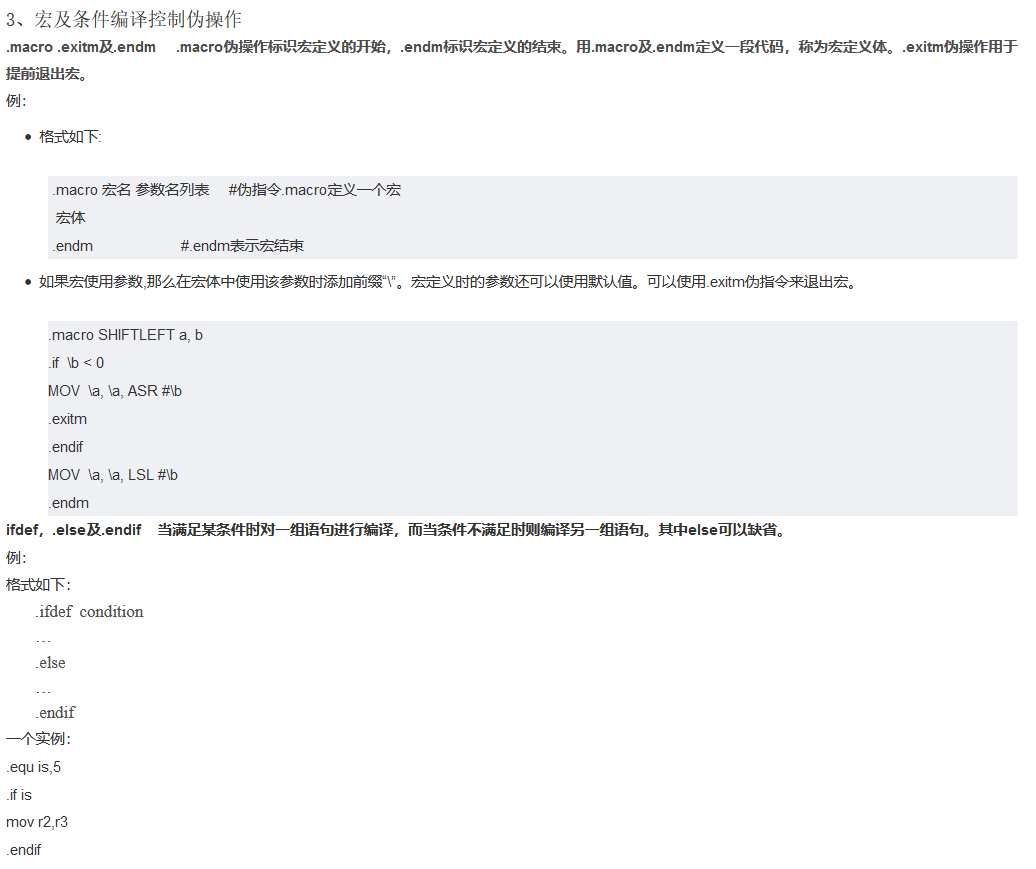
**Arm汇编一些用到的指令记录：**





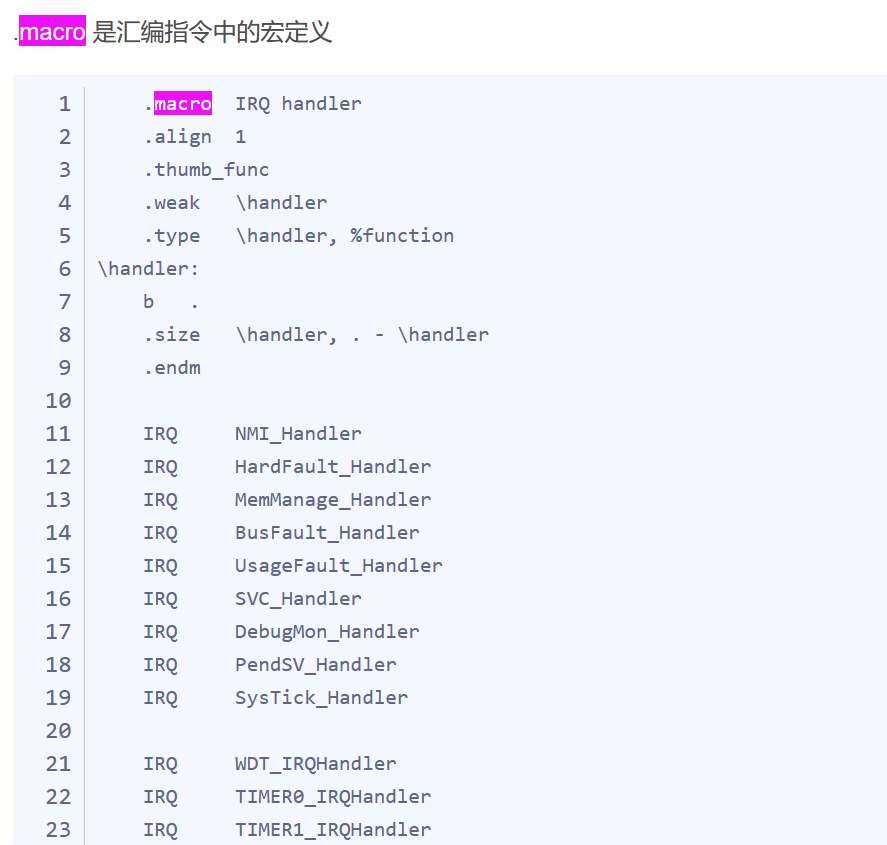
---------------------------------------------------------------------------------------------------------------------

\_start意义及在LD文件中的entry。



**基于该知识点的一个cortexM3中的使用实例：**

**其中，handler是IRQ宏名字的参数，宏的功能是对该参数进行一系列的操作。**





## ARM架构介绍

ARMv6-M与Cortex-M的关系：

ARMv6-M是ARMv6架构的一个子集或变种，专为微控制器设计。

Cortex-M0和Cortex-M0+是基于ARMv6-M架构的具体处理器实现。

Cortex-M0处理器采用了三级流水线结构（取值、译码、执行），并且支持Thumb指令集，包括16位和32位的Thumb-2指令集。

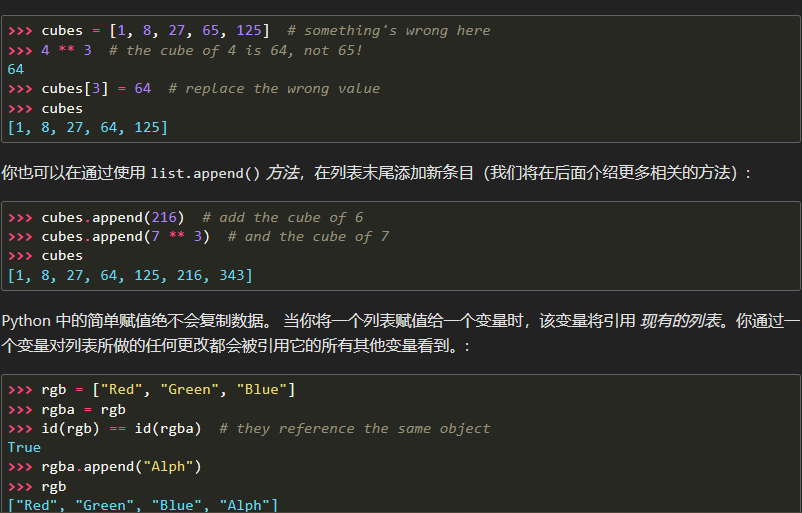
其中，cortexM0中NVIC部分作为系统控制器的一部分，其寄存器结构及地址是在armV6m中规定好的，在cortexM0手册中也是引用armV6m的信息。

## 打算

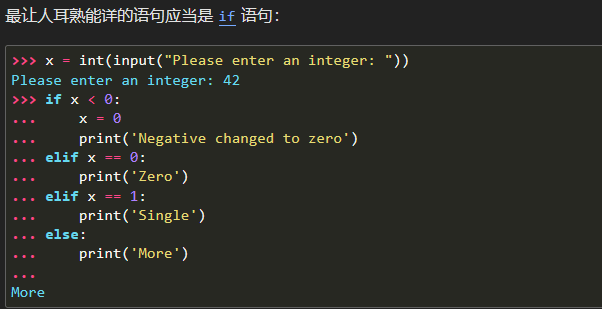
# \*\*\*\*\*\*\*\*\*\* Python \*\*\*\*\*\*\*\*\*

## List

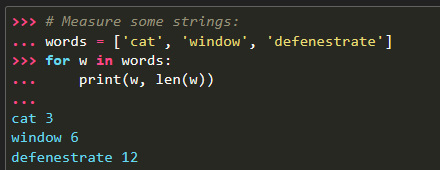
赋值操作为引用，没有拷贝list。



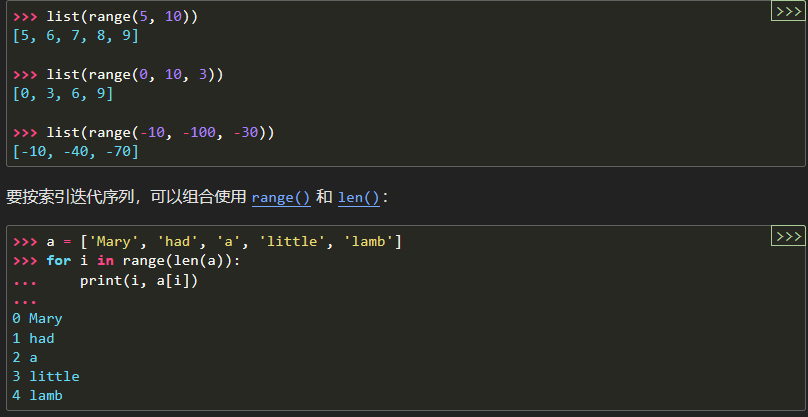
## If （注意缩进tab）



## For



## Range()函数



## Pass 不执行任何操作，语法需要



## Match语句（组合形式待查）

类似switch语句；注意case \_:匹配其他所有，类似default。



## 函数——默认值



# \*\*\*\*\*\*\*\*\*\* Tools \*\*\*\*\*\*\*\*\*

## Git 重设URL

