1，如果程序以NAND启动，让bin文件超过4K, 需要把所有的执行程序拷贝到SDRAM中，这里就需要重定位

2，如果以nor启动，由于nor进行写操作需要发送特殊命令，因此全局/静态变量就无法随意更改，因此需要把全局/静态变量重定位到SDRAM中

3，程序组成

1）.txt 代码段

2）.data 数据段

3）.rodata 只读的数据，如用const修饰过的

4）.bss 未初始化和初始化为0的全局/局部变量 （不占bin的大小）

5）.comment 注释段（不占bin的大小）

举例：

SECTIONS {

.text 0 : { \*(.text) } #.text 代码段， \*(.text) 所有的代码

.rodata : { \*(.rodata) } #.rodata 只读段， \*(.rodata) 所有的只读段

.data 0x30000000 : AT(0xeff) #.data 数据段 AT(0xeff)bin文件中的代码段， 0x3000000重定位的位置

{

data\_load\_addr = LOADADDR(.data); #获取bin文件中代码段的位置，0xeff

data\_start = . ; #代码开始的位置 0x30000000

\*(.data) #所有的代码段

data\_end = . ; #代码的结束位置 = 代码开始位置+所有代码段的大小

}

.bss : { \*(.bss) \*(.COMMON) } #.bbs bbs段， \*(.bbs)所有的bbs段， \*(.COMMON)所有的注释段

}

4，相对跳转指针B/BL 是基于当前指令的地址，不是一个确定的数值（PC+Offset)

5，怎么写位置无关的程序：

使用位置无关码! 不使用绝对地址! 最根本的办法是看反汇编

a. 调用程序时使用B/BL相对跳转指令

b. 重定位之前, 不可使用绝对地址，比如：

不可访问全局变量/静态变量

不可访问有初始值的数组(因为初始值放在rodata里，使用绝对地址来访问)

c. 重定位之后, 使用绝对跳转命令跳到Runtime Addr，比如：

ldr pc, =main

6， 需要使用lds链接脚本中的常量，如\_\_copy\_code\_start, 则需要在C文件中extern 出来

{

extern int \_\_copy\_code\_start；

volatile unsigned int \*dst = (unsigned int \*)&\_\_copy\_code\_start;

}

这里使用&\_\_copy\_code\_start 不是取得\_\_copy\_code\_start常量的地址，而是获取它的值，因为在lds脚本里面，一个符号会分为name和value, 通过 & 来获取常量的值；跟C语言不太一样，C语言中& 是获取变量的地址，而lds脚本确是获取常量的值