本文由 Jacky 原创，来自 http://blog.chinaunix.net/u1/58780/showart.php?id=462971
对于.lds 文件，它定义了整个程序编译之后的连接过程，决定了一个可执行程序的各个段的存储位置。虽然现在我还没怎么用它，但感觉还是挺重要的，有必要了解一下。
先看一下 GNU 官方网站上对.lds 文件形式的完整描述：

```
SECTIONS {
...
secname start BLOCK(align) (NOLOAD) : AT ( ldadr )
  { contents } >region :phdr =fill
...
}
```

secname 和 contents 是必须的，其他的都是可选的。下面挑几个常用的看看： 1、secname：段名
2、contents：决定哪些内容放在本段，可以是整个目标文件，也可以是目标文件中的某段（代码段、数据段等）
3、start：本段连接（运行）的地址，如果没有使用 AT（ldadr），本段存储的地址也是 start。GNU 网站上说 start 可以用任意一种描述地址的符号来描述。
4、AT（ldadr）：定义本段存储（加载）的地址。
看一个简单的例子：（摘自《2410完全开发》）

```
/* nand.lds */
SECTIONS {
 firtst 0x00000000 : { head.o init.o }
 second 0x30000000 : AT(4096) { main.o }
 }
```

以上，head.o 放在0x00000000地址开始处， init.o 放在 head.o 后面，他们的运行地址也是0x00000000，即连接和存储地址相同（没有 AT 指定）；main.o 放在4096（0x1000，是 AT 指定的，存储地址）开始处，但是它的运行地址在0x30000000，运行之前需要从0x1000（加载处）复制到0x30000000（运行处），此过程也就用到了读取 Nand flash。
这就是存储地址和连接（运行）地址的不同，称为加载时域和运行时域，可以在.lds 连接脚本文件中分别指定。
编写好的.lds 文件，在用 arm-linux-ld 连接命令时带-Tfilename 来调用执行，如
arm-linux-ld–Tnand.lds x.o y.o–o xy.o。也用-Ttext 参数直接指定连接地址，如
arm-linux-ld–Ttext 0x30000000 x.o y.o–o xy.o。

既然程序有了两种地址，就涉及到一些跳转指令的区别，这里正好写下来，以后万一忘记了也可查看，以前不少东西没记下来现在忘得差不多了。。。
ARM 汇编中，常有两种跳转方法：b 跳转指令、ldr 指令向 PC 赋值。
我自己经过归纳如下：

（1）

b step1：b 跳转指令是相对跳转，依赖当前 PC 的值，偏移量是通过该指令本身的 bit[23:0]算出来的，这使得使用 b 指令的程序不依赖于要跳到的代码的位置，只看指令本身。

（2）

ldr pc,=step1：该指令是从内存中的某个位置（step1）读出数据并赋给 PC，同样依赖当前 PC 的值，但是偏移量是那个位置（step1）的连接地址（运行时的地址），所以可以用它实现从 Flash 到 RAM 的程序跳转。

（3）

此外，有必要回味一下 adr 伪指令，U-boot 中那段 relocate 代码就是通过 adr 实现当前程序是在 RAM 中还是 flash 中。仍然用我当时的注释：

```
relocate: /* 把 U-Boot 重新定位到 RAM */
    adr r0, _start /* r0是代码的当前位置 */
/* adr 伪指令，汇编器自动通过当前 PC 的值算出 如果执行到_start 时 PC 的值，放到 r0中;

当 此段在 flash 中执行时 r0 = _start = 0；当此段在 RAM 中执行时_start =_TEXT_BASE(在 board/smdk2410/config.mk 中指定的值为0x33F80000，即 u-boot 在把代码拷贝到 RAM 中 去执行的代码段的开始) */
    ldr r1, _TEXT_BASE /* 测试判断是从 Flash 启动，还是 RAM */
 /* 此句执行的结果 r1始终是0x33FF80000，因为此值是又编译器指定的(ads 中设置，或-D 设置编译器参数) */
    cmp r0, r1 /* 比较 r0和 r1，调试的时候不要执行重定位 */
```

下面，结合 u-boot.lds 看看一个正式的连接脚本文件。这个文件的基本功能还能看明白，虽然上面分析了好多，但其中那些 GNU 风格的符号还是着实让我感到迷惑，好菜啊，怪不得连被 3家公司鄙视

```
OUTPUT_FORMAT("elf32-littlearm", "elf32-littlearm", "elf32-littlearm")
  ;指定输出可执行文件是 elf 格式,32位 ARM 指令,小端
OUTPUT_ARCH(arm)
  ;指定输出可执行文件的平台为 ARM
ENTRY(_start)
  ;指定输出可执行文件的起始代码段为_start.
SECTIONS
{
        . = 0x00000000 ; 从0x0位置开始
        . = ALIGN(4) ; 代码以4字节对齐
        .text : ;指定代码段
        {
          cpu/arm920t/start.o (.text) ; 代码的第一个代码部分
          *(.text) ;其它代码部分
        }
        . = ALIGN(4)
         .rodata : { *(.rodata) } ;指定只读数据段
        . = ALIGN(4);
```

```
        .data : { *(.data) } ;指定读/写数据段
        . = ALIGN(4);
        .got : { *(.got) } ;指定 got 段, got 段式是 uboot 自定义的一个段, 非标准段
        __u_boot_cmd_start = . ;把__u_boot_cmd_start 赋值为当前位置, 即起始位置
        .u_boot_cmd : { *(.u_boot_cmd) } ;指定 u_boot_cmd 段, uboot 把所有的 uboot
命令放在该段.
        __u_boot_cmd_end = .;把__u_boot_cmd_end 赋值为当前位置,即结束位置
        . = ALIGN(4);
        __bss_start = .; 把__bss_start 赋值为当前位置,即 bss 段的开始位置
        .bss : { *(.bss) }; 指定 bss 段
        _end = .; 把_end 赋值为当前位置,即 bss 段的结束位置
}
```