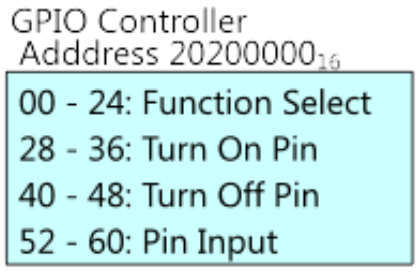

A diagram showing key parts of the GPIO controller.

阅读了手册可以得知，我们需要给 GPIO 控制器发送两个消息。我们必须用它的语言告诉它，如果我们这样做了，它将非常乐意实现我们的意图，去打开 OK 的 LED 指示灯。幸运的是，它是一个非常简单的芯片，为了让它能够理解我们要做什么，只需要给它设定几个数字即可。

mov r1,#1

lsl r1,#18

str r1,[r0,#4]

mov reg,#val 将数字 val 放到名为 reg 的寄存器中。

lsl reg,#val 将寄存器 reg 中的二进制操作数左移 val 位。

str reg,[dest,#val] 将寄存器 reg 中的数字保存到地址 dest + val 上。

这些命令的作用是在 GPIO 的第 16 号插针上启用输出。首先我们在寄存器 r1 中获取一个必需的值，接着将这个值发送到 GPIO 控制器。因此，前两个命令是尝试取值到寄存器 r1 中，我们可以像前面一样使用另一个命令 ldr 来实现，但 lsl 命令对我们后面能够设置任何给定的 GPIO 针比较有用，因此从一个公式中推导出值要比直接写入来好一些。表示 OK 的 LED 灯是直接连线到 GPIO 的第 16 号针脚上的，因此我们需要发送一个命令去启用第 16 号针脚。

寄存器 r1 中的值是启用 LED 针所需要的。第一行命令将数字 1_{10} 放到 r1 中。在这个操作中 mov 命令要比 ldr 命令快很多，因为它不需要与内存交互，而 ldr 命令是将需要的值从内存中加载到寄存器中。尽管如此，mov 命令仅能用于加载某些值。[4] 在 ARM 汇编代码中，基本上每个指令都使用一个三字母代码表示。它们被称为助记词，用于表示操作的用途。mov 是 “move” 的简写，而 ldr 是 “load register” 的简写。mov 是将第二个参数 #1 移动到前面的 r1 寄存器中。一般情况下，# 肯定是表示一个数字，但我们已经看到了不符合这种情况的一个反例。

第二个指令是 lsl（逻辑左移）。它的意思是将第一个参数的二进制操作数向左移第二个参数所表示的位数。在这个案例中，将 1_{10}（即 1_2）向左移 18 位（将它变成 $100000000000000000_2=262144_{10}$）。

再强调一次，我们只有去阅读手册才能知道我们所需要的值。手册上说，GPIO 控制器中有一个 24 字节的集合，由它来决定 GPIO 针脚的设置。第一个 4 字节与前 10 个 GPIO 针脚有关，第二个 4 字节与接下来的 10 个针脚有关，依此类推。总共有 54 个 GPIO 针脚，因此，我们需要 6 个 4 字节的一个集合，总共是 24 个字节。在每个 4 字节中，每 3 个比特与一个特定的 GPIO 针脚有关。我们想去启用的是第 16 号 GPIO 针脚，因此我们需要去设置第二组 4 字节，因为第二组的 4 字节用于处理 GPIO 针脚的第 10-19 号，而我们需要第 6 组 3 比特，它在上面的代码中的编号是 18（6×3）。

最后的 str（“store register”）命令去保存第一个参数中的值，将寄存器 r1 中的值保存到后面的表达式计算出来的地址上。这个表达式可以是一个寄存器，在上面的例子中是 r0，我们知道 r0 中保存了 GPIO 控制器的地址，而另一个值是加到它上面的，在这个例子中是 #4。它的意思是将 GPIO 控制器地址加上 4 得到一个新的地址，并将寄存器 r1 中的值写到那个地址上。那个地址就是我们前面提到的第二组 4 字节的位置，因此，我们发送我们的第一个消息到 GPIO 控制器上，告诉它准备启用 GPIO 第 16 号针脚的输出。

5、生命的信号

现在，LED 已经做好了打开准备，我们还需要实际去打开它。意味着需要给 GPIO 控制器发送一个消息去关闭 16 号针脚。是的，你没有看错，就是要发送一个关闭的消息。芯片制造商认为，在 GPIO 针脚关闭时打开 LED 更有意义。[5]

硬件工程师经常做这种反常理的决策，似乎是为了让操作系统开发者保持警觉。可以认为是给自己的一个警告。

```
mov r1,#1

lsl r1,#16

str r1,[r0,#40]
```

希望你能够认识上面全部的命令，先不要管它的值。第一个命令和前面一样，是将值 1 推入到寄存器 r1 中。第二个命令是将二进制的 1 左移 16
位。由于我们是希望关闭 GPIO 的 16 号针脚，我们需要在下一个消息中将第 16 比特设置为
1（想设置其它针脚只需要改变相应的比特位即可）。最后，我们写这个值到 GPIO 控制器地址加上 40_{10} 的地址上，这将使那个针脚关闭（加上 28 将打开针脚）。

6、永远幸福快乐

似乎我们现在就可以结束了，但不幸的是，处理器并不知道我们做了什么。事实上，处理器只要通电，它就永不停止地运转。因此，我们需要给它一个任务，让它一直运转下去，否则，树莓派将进入休眠（本示例中不会，LED 灯会一直亮着）。

```
loop$:

b loop$
```

name: 下一行的名字。

b label 下一行将去标签 label 处运行。

第一行不是一个命令，而是一个标签。它给下一行命名为 loop$，这意味着我们能够通过名字来指向到该行。这就称为一个标签。当代码被转换成二进制后，标签将被丢弃，但这对我们通过名字而不是数字（地址）找到行比较有用。按惯例，我们使用一个 $ 表示这个标签只对这个代码块中的代码起作用，让其它人知道，它不对整个程序起作用。b（ “branch” ）命令将去运行指定的标签中的命令，而不是去运行它后面的下一个命令。因此，下一行将再次去运行这个 b 命令，这将导致永远循环下去。因此处理器将进入一个无限循环中，直到它安全关闭为止。

代码块结尾的一个空行是有意这样写的。GNU 工具链要求所有的汇编代码文件都是以空行结束的，因此，这就可以你确实是要结束了，并且文件没有被截断。如果你不这样处理，在汇编器运行时，你将收到烦人的警告。

7、树莓派上场

由于我们已经写完了代码，现在，我们可以将它上传到树莓派中了。在你的计算机上打开一个终端，改变当前工作目录为 source 文件夹的父级目录。输入 make 然后回车。如果报错，请参考排错章节。如果没有报错，你将生成三个文件。kernel.img 是你的编译后的操作系统镜像。kernel.list 是你写的汇编代码的一个清单，它实际上是生成的。这在将来检查程序是否正确时非常有用。kernel.map 文件包含所有标签结束位置的一个映射，这对于跟踪值非常有用。

为安装你的操作系统，需要先有一个已经安装了树莓派操作系统的 SD 卡。如果你浏览 SD 卡中的文件，你应该能看到一个名为 kernel.img 的文件。将这个文件重命名为其它名字，比如 kernel_linux.img。然后，复制你编译的 kernel.img 文件到 SD 卡中原来的位置，这将用你的操作系统镜像文件替换现在的树莓派操作系统镜像。想切换回来时，只需要简单地删除你自己的 kernel.img 文件，然后将前面重命名的文件改回 kernel.img 即可。我发现，保留一个原始的树莓派操作系统的备份是非常有用的，万一你要用到它呢。

将这个 SD 卡插入到树莓派，并打开它的电源。这个 OK 的 LED 灯将亮起来。如果不是这样，请查看故障排除页面。如果一切如愿，恭喜你，你已经写出了你的第一个操作系统。 课程 2 OK02 将指导你让 LED 灯闪烁和关闭闪烁。

---

1. 是的，我说错了，它告诉的是链接器，它是另一个程序，用于将汇编器转换过的几个代码文件链接到一起。直接说是汇编器也没有大问题。 □
2. 其实它们对你很重要。由于 GNU 工具链主要用于开发操作系统，它要求入口点必须是名为 _start 的地方。由于我们是开发一个操作系统，无论什么时候，它总是从 _start 开时的，而我们可以使用 .section .init 命令去设置它。因此，如果我们没有告诉它入口点在哪里，就会使工具链困惑而产生警告消息。所以，我们先定义一个名为 _start 的符号，它是所有人可见的（全局的），紧接着在下一行生成符号 _start 的地址。我们很快就讲到这个地址了。 □
3. 本教程的设计减少了你阅读树莓派开发手册的难度，但是，如果你必须要阅读它，你可以在这里 SoC-Peripherals.pdf 找到它。由于添加了混淆，手册中 GPIO

使用了不同的地址系统。我们的操作系统中的地址 0x20200000 对应到手册中是 0x7E200000。

4. mov 能够加载的值只有前 8 位是 1 的二进制表示的值。换句话说就是一个 0 后面紧跟着 8 个 1 或 0。
5. 一个很友好的硬件工程师是这样向我解释这个问题的：

   原因是现在的芯片都是用一种称为 CMOS 的技术来制成的，它是互补金属氧化物半导体的简称。互补的意思是每个信号都连接到两个晶体管上，一个是使用 N 型半导体的材料制成，它用于将电压拉低，而另一个使用 P 型半导体材料制成，它用于将电压升高。在任何时刻，仅有一个半导体是打开的，否则将会短路。P 型材料的导电性能不如 N 型材料。这意味着三倍大的 P 型半导体材料才能提供与 N 型半导体材料相同的电流。这就是为什么 LED 总是通过降低为低电压来打开它，因为 N 型半导体拉低电压比 P 型半导体拉高电压的性能更强。

   还有一个原因。早在上世纪七十年代，芯片完全是由 N 型材料制成的（NMOS），P 型材料部分使用了一个电阻来代替。这意味着当信号为低电压时，即便它什么事都没有做，芯片仍然在消耗能量（并发热）。你的电话装在口袋里什么事都不做，它仍然会发热并消耗你的电池电量，这不是好的设计。因此，信号设计成 “活动时低” ，而不活动时为高电压，这样就不会消耗能源了。虽然我们现在已经不使用 NMOS 了，但由于 N 型材料的低电压信号比 P 型材料的高电压信号要快，所以仍然使用了这种设计。通常在一个 “活动时低” 信号名字上方会有一个条型标记，或者写作 SIGNAL_n 或 /SIGNAL。但是即便这样，仍然很让人困惑，那怕是硬件工程师，也不可避免这种困惑！

---

via: https://www.cl.cam.ac.uk/projects/raspberrypi/tutorials/os/ok01.html

作者： Robert Mullins 选题： lujun9972 译者： qhwdw 校对： wxy

本文由 LCTT 原创编译， Linux中国 荣誉推出

点击“了解更多”可访问文内链接