|  |  |  |
| --- | --- | --- |
| Lesson 2 OK02  The OK02 lesson builds on OK01, by causing the 'OK' or 'ACT' LED to turn on and off repeatedly. It is assumed you have the code for the [Lesson 1: OK01](http://www.cl.cam.ac.uk/projects/raspberrypi/tutorials/os/ok01.html) operating system as a basis.   |  | | --- | | **Contents**   * [1 Waiting](http://www.cl.cam.ac.uk/projects/raspberrypi/tutorials/os/ok02.html#waiting) * [2 The All Together](http://www.cl.cam.ac.uk/projects/raspberrypi/tutorials/os/ok02.html#together) |   1 Waiting  Waiting is a surprisingly useful part of Operating System development. Often Operating Systems find themselves with nothing to do, and must delay. In this example, we wish to do so in order to allow the LED flashing off and on to be visible. If you just turned it off and on, it would not be visible, as the computer would be able to turn it off and on many thousands of times per second. In later lessons we will look at accurate waiting, but for now it is sufficient to simply waste time.  mov r2,#0x3F0000 wait1$: sub r2,#1 cmp r2,#0 bne wait1$  **sub reg,#val** subtracts the number **val** from the value in **reg**.  **cmp reg,#val** compares the value in **reg** with the number **val**.  Suffix **ne** causes the command to be executed only if the last comparison determined that the numbers were not equal.  The code above is a generic piece of code that creates a delay, which thanks to every Raspberry Pi being basically the same, is roughly the same time. How it does this is using a **mov** command to put the value 3F000016 into **r2**, and then subtracting 1 from this value until it is 0. The new commands here are **sub**, **cmp**, and **bne**.  **sub** is the subtract command, and simply subtracts the second argument from the first.  **cmp** is a more interesting command. It compares the first argument with the second, and remembers the result of the comparison in a special register called the current processor status register. You don't really need to worry about this, suffice to say it remembers, among other things, which of the two numbers was bigger or smaller, or if they were equal.[[1]](http://www.cl.cam.ac.uk/projects/raspberrypi/tutorials/os/ok02.html" \l "note1" \o "Note 1)  **bne** is actually just a branch command in disguise. In the ARM assembly language family, any instruction can be executed conditionally. This means that the instruction is only run if the last comparison had a certain result. We will use this extensively later for interesting tricks, but in this case we use the **ne** suffix on the **b** command to mean 'only branch if the last comparison's result was that the values were not equal'. The **ne** suffix can be used on any command, as can several other (16 in all) conditions such as **eq** for equal and **lt** for less than.  2 The All Together  I mentioned briefly last time that the status LED can be turned off again by writing to an offset of 28 from the GPIO controller instead of 40 (i.e. **str r1,[r0,#28]**). Thus, you need to modify the code from OK01 to turn the LED on, run the wait code, turn it off, run the wait code again, and then include a branch back to the beginning. Note, it is not necessary to re-enable the output to GPIO 16, we need only do that once. If you're being efficient, which I strongly encourage, you should be able to reuse the value of **r1**. As with all lessons, a full solution to this can be found on the [download page](http://www.cl.cam.ac.uk/projects/raspberrypi/tutorials/os/downloads.html). Be careful to make sure all of your labels are unique. When you write **wait1$:** you cannot label another line **wait1$**.  On my Raspberry Pi it flashes about twice a second. this could easily be altered by changing the value we set **r2** to. However, unfortunately we can't precisely predict the speed this runs at. If you didn't manage to get this working see our trouble shooting page, otherwise, congratulations.  In this lesson we've learnt two more assembly commands, **sub** and **cmp**, as well as learning about conditional execution in ARM.  In the next lesson, [Lesson 3: OK03](http://www.cl.cam.ac.uk/projects/raspberrypi/tutorials/os/ok03.html) we will evaluate how we're coding, and establish some standards so that we can reuse code, and if necessary, work with C or C++ code.   1. [1][^](http://www.cl.cam.ac.uk/projects/raspberrypi/tutorials/os/ok02.html#note1a) I suppose if you've followed the link you really do want to know about it. The CPSR is a 32 bit register consisting of many individual bit fields. It has bit fields for positive, zero and negative. When a cmp instruction is issued, it subtracts the second argument from the first, and notes down whether it is positive, zero or negative with these fields. Zero means the numbers were equal (a-b=0 implies a=b), positive means a is bigger than b (a-b>0 implies a>b) and negative less than. A variety of other comparison instructions exist, but cmp is the most intuitive.   Spot a mistake? You can help improve this tutorial on [GitHub](https://github.com/chadderz121/bakingpi-www).  [Creative Commons Licence](http://creativecommons.org/licenses/by-sa/3.0/deed.en_GB) Baking Pi: Operating Systems Development by Alex Chadwick is licensed under a [Creative Commons Attribution-ShareAlike 3.0 Unported License](http://creativecommons.org/licenses/by-sa/3.0/deed.en_GB).  Based on contributions at <https://github.com/chadderz121/bakingpi-www>. | 第2课 OK02  OK02的课程是基于OK01课程的，所以，很有必要搞懂OK01课。这一课，我们的任务是让“OK”或者“ACT”LED灯周期性亮灭——闪烁。我们假设你实践了OK01课的操作系统代码了。  目录   1. 等待 2. 组装 3. 等待   等待是操作系统开发中一个非常有用的部分，很神奇吧。操作系统经常会发现自己没有事情可做，所以必须等待。在这个例子中，我们想要做的就是让LED灯一亮一灭到人眼可以观察的程度。如果当你刚接通LED灯时就把它关闭掉，我们是无法观察到它的闪烁的。而且计算机可以在一秒内进行开关操作上千次，这样我们就更加难以观察了。在后续课程中，我们会学到精确等待的技术，不过，我们现在的用“干等着”技术就足够了。  （代码）  指令sub reg, #val意思是把寄存器reg中存储的数值减去数值val后所得结果仍在reg寄存器中。  指令cmp reg, #val意思是寄存器reg中的数值和数值val进行比较。  指令后缀ne一般附着在其他指令后面，意思是根据最近一次比较指令产生的结果是否为不相同来决定其所附着的指令的执行。那么，指令bne wait1$意思就是其上一条指令所比较的结果如果是不相等，则跳转（branch）到标签wait1$处下一行执行；如果相等，就什么也不执行。  上面的代码会产生一个延迟。由于每一块树莓派都一样，所以，这个产生的延迟长短基本上可以说是一样的。它是怎么做到延迟的呢？首先使用mov指令，把0x3F0000十六进制数值存入寄存器r2中，然后把r2中的数值减去1，存入r2中，然后再把r2中的数值减去1，周而复始，直到r2中的数值变为0为止。因为每一步都需要耗费一定的时间，这样累加起来的时间消耗就是产生的延迟的时间长短。这里用到的新指令包括sub，cmp和bne。  指令sub是减法指令，sub是subtract的简写。指令cmp更加有意思。它把第一个参数和第二个参数进行比较，并且把比较结果保存在一个特殊的寄存器中。这个特殊的寄存器名叫当前处理器状态寄存器。别太担心这个寄存器的古怪名字。因为两个数值进行比较，结果只能是大、小或者相等。刚才说的那个特殊寄存器里就用某种方式来存储这个大、小或者相等的信息。具体如何是如何表示的，后面后详细讲到。  指令bne实际上就是一个跳转（branch）指令。在ARM这种汇编语言格式中，任何指令都可以条件地进行执行。这个意思是说仅当最近的比较结果是确定的，当前指令才执行，否则就不执行。以后，我们会用到这个技巧来扩展我们的代码，现在就用bne吧。当前的情况就是当比较的两个数值不相等时，才去执行跳转指令（branch）。后缀ne可以用在任何指令中。除了ne后缀还有其他的后缀，比如eq表示相等，lt表示小于后缀。   1. 组装   我曾经简短地提及到：给GPIO控制器的偏移28个地址单元处进行写入操作（比如：str r1, [r0, #28]），就可以关闭LED灯。我们现在就可以在OK01课程代码的基础上进行修改，加上延迟代码，再加上关闭LED代码，再转到延迟代码进行执行。所以，我们这里需要一个跳转指令以便让执行流程转到代码的开始部分。这里要谨记的是，并不需要重新让GPIO的16号引脚可以输出，我们只需要做一次这样的动作。  如果你经验足够丰富了，那么我强烈建议你重用r1寄存器中的数值。当所有的课程都结束时，本门课程的全解会再下载页里提供。请仔细对待代码中的每一个标签，确保它们时独一无二的。比如当你在某处写下wait1$，那么在其他任何地方都应该再出现wait1$了。  我树莓派板子上的LED灯每1秒闪烁2次。如果想改变闪烁频率（既每1秒闪烁的次数），修改寄存器r2中的数值就可以了。尽管可以修改数值来改变闪烁频率，但是想要准确测算其频率值是不可能的。如果你无法让你的LED灯闪烁，请移步问题解决页。否则，恭喜你了。  这一个课时中，我们学习了两个汇编指令：sub和cmp。还学习了ARM汇编语言中的条件执行。  下一个课时是OK03。那时我们将学习如何评估我们的代码，并建立一些标准来重用我们的代码。如果有必要，我们将在代码中插入C或者C++代码。心动了吗？   1. 我假设你真的明白你想要做什么。CPSR寄存器是一个32位的寄存器，其由许多独立的位域组成。当然，里面有表示数值位正、负或零的位域。当执行cmp指令时，指令会把第1个参数减去第2个参数，并把结果是正、负或零用位域标记出来。零表示两个数值是相同的。因为a-b=0隐含着a=b。正表示a大于b，a-b>0隐含着a>b。负表示a小于b，a-b<0隐含着a<b。还有其他一些比较指令，但cmp指令最常用。 |