UNIVERSITY OF CAMBRIDGE

Computer Laboratory

Lesson 2 OK02

The OK02 lesson builds on OK01, by causing the 'OK' LED to turn on and off repeatedly. It is assumed you have the code for the Lesson 1: OK01 operating system as a basis.

Contents

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$:
sub r2,#1
cmp r2,#0
bne wait1$
```

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 $3F0000_{16}$ 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]

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.

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.

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. Be careful to make sure all of your labels are unique. When you write `wait1$:` you cannot label another line `wait1$`.

On my Raspbery 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 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]^ 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 vareity of other comparison instructions exist, but cmp is the most intuitive.