Faculty of Science, Engineering and Technology

Computer Systems

Week 8

Overview

In this laboratory you will begin programming the Raspberry Pi in ARM Assembly

Purpose: Learn to implement loops and comaprisons, and timers in ARM assembly with your Raspberry Pi.

Task:

Time: This lab is due by the start of your week 9 lab.

Assessment: This lab is worth 1% of your assessment for this unit, and only if demonstrated to your lab demonstrator in the week it is due.

Bring to Lab

- your Raspberry Pi (either model 2B,3B,3B+, 4B)
- an SD card and SD card reader
- 2 lead wires, one LED, one resistors and and breadboard.

Resources:

- Lectures slides - week 7 and 8
- RPiGPIO.xls (from Week 7 Lab resources)
- Video tutorials:
  - The GPIO registers and header pins
  - Connecting up a simple LED circuit

Submission Details

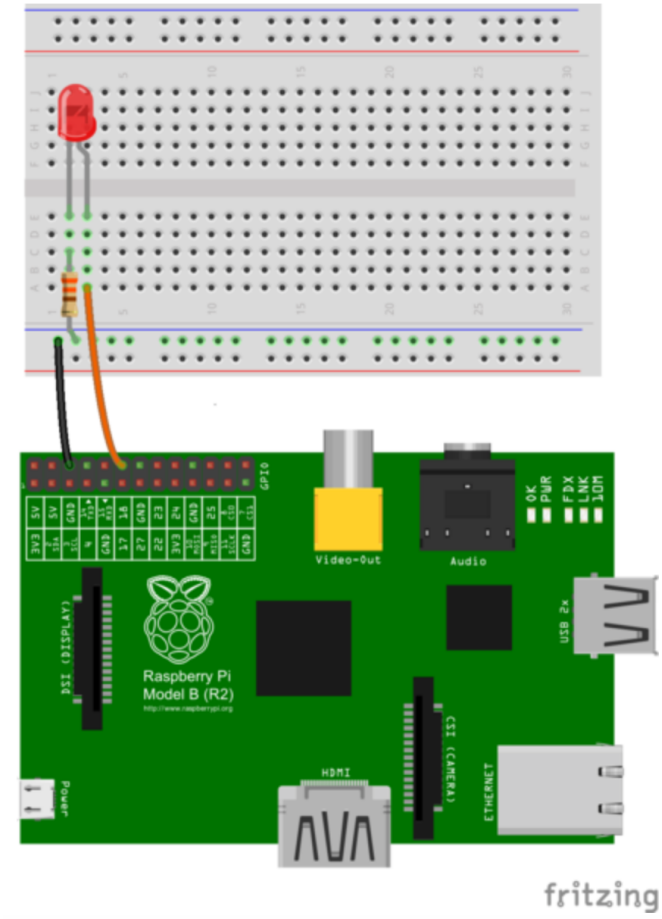
You must submit the following files to Canvas:

- A document containing all required work as described below.

Instructions

Timers

1. Wire up a circuit exactly as shown below (same as last week). Ask your lab demonstrator for help if you need.



2. Start up FASMARM.exe and open up OK2.ASM from the task resources on Canvas.
3. Take some time to step through the code and understand what it is doing
4. Locate the code specifying the number of loop operations (Look for mov r2,$0F0000)
5. Change this to the first six digits of your student number (preceded with #) e.g. my number is 342112 so I would enter #342112 (no letters btw - just numbers)
6. Try compiling and see the error message that comes back.

Insert the error message into your submission document

7. Convert your student number to Hex, and ***enter it in your submission document.***

8. In lectures we discussed some limitations of the mov command when loading numbers into registers. In particular, mov requires the number it is moving to have at least two pairs of 0 nibbles (i.e., Four 0 Hex digits in pairs). Review the relevant lecture slides and provide an answer to the following:
   8.1. Why does MOV only work with numbers with 24 bits set to 0 ?
   8.2. How can MOV still be used for numbers that do not satisfy this ?
   8.3. Identify the three bytes (as hex digits) needed to construct your student number, and write the code to load the entire number into a register.

Discuss your answers with your lab demonstrator and include them in your submission document.

9. Replace the mov r2,$0F0000 lines in the OK2 file with your code from 8.3.
10. Try to compile, and ask your demonstrator if you need help.
11. Once compiled, follow the usual procedures to load it onto your SD card, and test on your Pi. You should see the LED flashing, using your student number in the count. Spend some time enjoying it!

Some Patterned LED Flashing

12. Save your work and now open up OK4.ASM from the task resources.
13. Take a look at the code for OK4.ASM and see if you can interpret what is going on (without looking ahead).
14. Currently the code simply turns the LED on and off, this time using a timer. OK4 implements the following loop structure to achieve this:

```
Start Outer loop (infinite loop)
    turn on LED
    Set timer
            start timer loop (timed loop)
            end timer loop
    Turn off LED
            start timer loop 2 (timed loop)
            end timer loop 2
end Outer loop
```

15. We are going to modify this code so that it implements 3 flashes followed by a pause. This will require 2 extra loops . Think about how you would do this, using comments or hand written notes to plan what modifications.

Include your notes or comments in your submission document

16. Now implement it, setting the time interval to 3 seconds (0x2DC6C0)
17. If you're not sure, then do it like this:
    17.1. Add a counter to the outer loop (initialise a currently unused register - like r2).
    17.2. Subtract one from the counter (r2) each iteration of the outer loop using the sub instruction.
    17.3. Modify the b loop$ instruction so it now only branches back to loop$ when r2 is NOT zero (for this you will need to use the cmp instruction to check for 0, and then the bne instruction to perform the conditional branch). Ask your lab demonstrator if you need.
    17.4. To implement the pause between sets of three flashes, you will need a second timed loop after the end of the outer loop. Use the timed loop between flashes as a guide, however remember to use a different label to branch to (they must be unique). For this timed loop, set the time interval to 3 seconds (0x2DC6C0). You can use register r4 to set this. You will need to load the r4 register using one mov and two ours just before the loop starts.
    17.5. To ensure the pattern continues forever, you will need to wrap a new loop around the now modified outer loop + timed 3 second loop. This is straight forward. Just add a new label at the top and a branch ("b") to that label at the bottom.
18. Compile, test and debug your program. Some debugging tips:
    - Read the error messages provided by the assembler. They are not too bad, and they indicate the filename and line number of syntax errors.
    - Use the web. A Google search on ARM ASM bne will get you relevant examples on using bne. Go to the infocenter.arm.com result to get the "official" instructions.
    - Get the mov operations right. # means decimal, $ means hex.

Demonstrate your program to your lab demonstrator and copy your code into your submission document

When complete:

- Submit your answers (screen shots, etc) in a single document using **Canvas**
- Show your lab demonstrator your working circuits in class (you must do this to get the 1%). Your lab demonstrator may request you to resubmit if issues exist.