Faculty of Science, Engineering and Technology

Computer Systems

Week 10

Overview

In this laboratory you will be implementing functions on your Raspberry Pi in ARM Assembly

Purpose: Learn how GPIO inputs can be read, and how pixels can be drawn to screen using the frame buffer

Task:

Time: The screen writing task is optional for assessment - you may either do this or work on your Assignment 2. If you do not do this lab, submit evidence of your assignment 2 planning / design

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+ or 4)
- an SD card and SD card reader
- a HDMI to DVI adapter (for lab screens)

Resources:

- Lectures- 10.1, 10.2 and 10.3
- 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

Screen Writing

1. Download and unzip the Task Resources from Canvas
2. In your task resources folder, go into the folder DrawPixel-example
3. Open and examine the file kernel7.asm in FASMARM. See if you can follow the instructions. Do the same for DrawPixel.
4. Identify the parameters required when calling drawPixel, and identify what happens to them in the drawPixel function.

Provide your answer to this in your submission document

5. Now identify the code which controls the calling of drawPixel, particularly the counter incrementing code and the test to see the loop should continue. What is this code doing ?

Provide your answer to this in your submission document

6. Assemble it, and copy the kernel7.img file to your SD card so you can verify that it works. Don't forget to plug your screen into your Pi (best do this before you power up).

7. Time for some fun. Edit kernel7.asm to draw geometric shapes on the screen. You will probably want to include a new loop inside the outer loop for this.
8. Start with a box to keep things simple (say 8 pixels wide x 16 pixels high). Try other shapes like a triangle.
9. Assemble and test your code regularly, and keep copies so you can back track and try different things.

Add your kernel7.asm code of your shape drawer to the submission document

Some debugging advice:

- Read the error messages provided by the assembler (they at least tell you useful things like which line is causing trouble, and possibly more)
- Use Google to search on instructions. For example, search terms like ARM ASM bne will get you relevant examples on using bne. Goto the infocenter.arm.com to get the official documentation.

- Get the mov and ldr operations right. ldd uses registers, mov uses # for a number. Mov will generate an error for numbers not divisible by 256 (so you may need to use orr commands to build up the number).
- Function does not return: check that the pops match the push operations in reverse order (unless using braces "{ , }"). Don't pop {lr} - just use **bl** and **bx lr** to return from function.

Combining input handling with screen text writing

10. Folder OK04-input has example code for handling input (using header pin 19 connected to GPIO10), with LEDs connected to show output (connected to pins 12 and 14 (GPIO18 and GPIO23)
11. Folder CHAR4-DIY_KBD has example code for handling input (again using pin 19 connected to GPIO10), this time in combination with screen output.
12. Have a look at the code for these, wire up your pi appropriately and assemble the code…. and be inspired.

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.