ECE271: Microcomputer Architecture and Applications — University of Maine

Lab #4: GPIOs in Assembly Language

Week of 18 February 2019

Goals

1. Learn Thumb-2 Assembly Language

Pre-lab

1. Complete the pre-lab before attending lab. The pre-lab is in a separate pdf file, found on the website.

Lab Procedure

Using your Lab#1 as a reference, implement a Thumb-2 assembly language program that causes the red LED to light up when the joystick is pressed up, and the green LED to light up when the joystick is pressed down. Do this lab in three parts, as described below.

NOTE: be sure to disconnect your keypad before starting this lab, as the external pullups will confuse the GPIOA joystick inputs.

Part A – Lighting up the LEDs

1. As you should recall from Lab#1, there are two user-controllable LEDs (light-emitting diodes) on the STM32L4 discovery board. The red one is connected to GPIO-PB2 (GPIO Port B Pin 2) and the green one is connected to GPIO-PE8 (GPIO Port E Pin 8).

2. To review, to turn on the LEDs you will do the following five steps:

   (a) Enable the clock for the corresponding GPIO ports (by default they are disabled).

   (b) Set the mode of the GPIO pins to be output (by default they are analog).

   (c) Set the push-pull/open-drain setting for the GPIO pins to push-pull.

   (d) Set the pull-up/pull-down setting for the GPIO pins to no pull-up/pull-down.

   (e) Finally, set the output of the GPIO pin to have a value of 1 (corresponding to 3.3V)

3. For this lab we will program in Thumb-2 assembly.

   (a) First download the Lab4 template `ece271_lab4.zip`. This will be on the course website. (If you wish to use Linux/OSX then there is a separate template, similar to previous labs. Also see the end of this document for additional Linux instructions).

   (b) Modify the `main.s` file to enable the registers you need to turn on the LEDs.

The values will be the same as those calculated for Lab #1.

Please use pre-defined constants (rather than raw hex values) wherever possible.

The pre-lab should walk you though how to set a register in assembly language.

i. Enable the GPIOA, GPIOB, and GPIOE blocks via the AHB2ENR register.
ii. Now set the GPIOB Pin 2 MODER output register to be a digital output.
iii. Do the same for the push-pull setting in OTYPER.
iv. Set the no-pull-up no pull-down setting in GPIOB->PUPDR
v. Finally to actually enable/disable the LED we will need to set the proper bit in the output-data register ODR.
vi. Repeat the above steps, but do them for GPIOE pin 8 for the green LED.
vii. At the end of your code put an infinite loop to keep the code from executing off the end of your program.

```
BRANCHFOREVER        b BRANCHFOREVER
```

(c) While doing this, be sure to comment your code appropriately!

Part B – Joystick Buttons

1. To review: to read the state of the joystick you will have to do the following:
   (a) Enable the clock for the corresponding GPIO ports (by default they are disabled).
   (b) Set the mode of the GPIO pins to be input (by default they are analog).
   (c) Set the pull-up/pull-down setting for the GPIO pins to have a pull-down.
   (d) Finally, read the status of the corresponding GPIO pin.
2. You will do this in Thumb-2 assembly language.
   (a) Initialize the GPIOA settings as you did in Lab#1.
      i. You should have already enabled GPIO bank A in Part A of the lab. If you didn't, make sure your code properly sets the enable bit.
      ii. Now enable the joystick GPIO pins so that the proper pins in MODER are set to be inputs.
      iii. Set each joystick GPIO to be pull-down in the PUPDR register.
      iv. Finally to read the value read the corresponding bit in the IDR (input-data) register. To read the status on GPIO pin#X just check if bit#X is 1 or 0. You can use a bitwise and instruction with a proper mask to check this.
   (b) Modify your code to have an infinite loop.

   Just add a label at the start, and a `b` branch always instruction up to it.

(c) Each time through the loop read the status of the UP and DOWN GPIOs. If UP is high, then light the red LED, otherwise turn it off. If DOWN is high, then light the green LED. Otherwise turn it off.

Do this by doing an if/then/else type construct in assembly.

Test the proper IDR register bit, and if it is set, enable the LED, if not set, skip ahead to code that disables the LED.

(d) Once everything goes well, the two LEDs should be lit when the proper button is pressed. If it doesn't work you will have to debug your code to find out what is wrong.

Part C – Something Cool

Do something cool! You can come up with something on your own, but here is a list of ideas you can use.

1. Re-do your something cool from Lab#1 but in assembly.
2. Optimize your code for size! See how small you can make your code.
3. Look into using the GPIO BSRR register to enable/disable things rather than using ODR.

Lab Demo

Student Name:_____ TA:_____ Date:_____

1. Submit your code
    - Complete a README with the post-lab (next page) answers.
    - Make sure the code is properly commented.
    - Submit your code. Push it via git to gitlab. Directions on how to do this will be posted to the course webpage.
2. Demo your implementation to your lab TA.

Post-Lab

- Place your answers to the question in a file Readme.md
- Submit with your code via gitlab

1. You can use the BIC instruction to bitwise-clear out bits. Is this instruction necessary? How could you do the same thing if BIC did not exist?
2. How does the CMP compare instruction differ from the SUB subtract instruction?

Linux / GNU Assembler Notes

While the actual assembly language is the same between Keil and Linux, the assembler directives are a bit different. Here is a summary of things that might affect this lab.

1. Code Comments: You can use C and C++ style comments. You *cannot* use the semi-colon for an end-of-line comment, however you can use the '@' sign instead.

2. Labels: Labels are followed by a colon

```
loop:
    b loop
```

3. Directives

   (a) EQU – instead use `.equ`

```
.equ    FINAL_ANSWER, 42
```

   (b) The other directives are different too but I'm not sure any are used in this lab.