2024

EEE3095/6S: PRACTICAL 4

1. OVERVIEW

An Assembly program is made up of a sequence of instructions, each of which gets executed by the CPU. But why bother learning Assembly when we have C for embedded programming?

The main reason nowadays is that Assembly code is usually used as a tiny piece of a (larger) C program to do something very fast and specialised. And in this course, it also helps to give you an understanding of low-level operations and register manipulation. And after writing your own Assembly code, you should have a greater appreciation for all the simplicity inherent in C programming!

2. OUTCOMES AND KNOWLEDGE AREAS

In this practical, you will be writing Assembly code to interface with your microcontroller board to perform basic operations. After this practical, you should know how to manipulate the LEDs on your STM board and use the pushbuttons to change their sequencing — all of which is done in Assembly!

You will learn about the following aspects:

- Interfacing with the STM board using Assembly
- Using (and varying) time delay loops in Assembly
- Displaying LED patterns using Assembly
- Reading from pushbuttons using Assembly

3. DELIVERABLES

For this practical, you must:

- Develop the code required to meet all objectives specified in the Tasks section
- Push your completed code to a shared repository on GitHub
- Demonstrate your working implementation to a tutor in the lab (explicitly showing the use of Assembly code instead of C). You will be allowed to conduct your demo during any lab session before the practical submission deadline.
- Write a short report documenting your assembly.s code, GitHub repo link, and a brief description of the implementation of your solutions. This must be in PDF format and submitted on Amathuba/Gradescope.

Note: Your code (and GitHub link) should be copy-pasted into your short report so that the text is fully highlightable and searchable; do **NOT** submit screenshots of your code (or repository link) or you will be penalised.

- Your practical mark will be based both on your demo to the tutor (i.e., completing the below tasks correctly) as well as your short report. Both you and your partner will receive the same mark.

4. GETTING STARTED

As before:

1. 1. Clone or download the Git repository:
2. $ git clone https://github.com/eee3096s/2024
3. 2. The project folder that you will be using for this practical is /
4. eee3096s/2024/Prac4
5. Open STMCubeIDE, then go to File --> Import --> Existing Code as Makefile Project -->> Next. Then Browse to the project folder above, select it, and select "MCU ARM GCC" as the Toolchain --> Finish.
   Note: This IDE provides a GUI to set up clocks and peripherals (GPIO, UART, SPI, etc.) and then automatically generates the code required to enable them in the main.c file. The setup for this is stored in an *.ioc* file, which we have provided in the project folder if you would ever like to see how the pins are configured. However, it is crucial that you do **NOT** make/save any changes to this *.ioc* file as it would re-generate the code in your main.c file and may **delete** code that you have added.
6. In the IDE, navigate and open the assembly.s file under the Core/src folder, and then complete the Tasks below. You will not be editing main.c in this practical.
   Note: All code that you need to write/add in the assembly.s file is marked with a "TODO" comment; do not edit any part of the other code that is provided.

5. TASKS

Complete the following tasks using the assembly.s file in STM32CubeIDE, and then demonstrate the working execution of each task to a tutor:

1. By default, the LEDs should increment by 1 every 0.7 seconds (with the count starting from 0).
2. While SW0 is being held down, the LEDs should change to increment by **2** every 0.7 seconds.
3. While SW1 is being held down, the increment timing should change to every **0.3** seconds.

4. While SW2 is being held down, the LED pattern should be set to **0xAA**. Naturally, the pattern should stay at 0xAA until SW2 is released, at which point it will continue counting normally from there.
5. While SW3 is being held down, the pattern should **freeze**, and then resume counting only when SW3 is released.
   Notes: Only *one* of SW2 or SW3 will be held down at one time, but SW0 *and* SW1 may be held at the *same* time.

6. HELPFUL ASSEMBLY TIPS!

- Spending a few minutes beforehand and planning your code path (with a flowchart or some such) can make your coding a lot more efficient.
- Check the Course Notes for how to implement a basic **delay**.
- Use **labels**, **conditions**, and **branches**; you could always cut down on them later when optimising your code once you have a working solution.
- Do not edit the provided code base — especially the constants at the end of the code and frontmatter. These have been marked "DO NOT EDIT" for good reason!
- You only need to edit assembly.*s* in this practical; the *.ioc* file and *.c* files can be ignored and left as they are.