EENG260 (Microcontrollers) Lab 0:

Introduction to the labs format, tools, and expectations.

Tasks for this lab

- Read and understand this Lab 0 document
- Decide on lab teams (as instructed during the lab)
- Set up a workspace for your Code Composer Studio projects
- Learn how to set up a new project in Code Composer Studio
- Familiarize yourself with the assembly instructions you will use to complete your first labs

Reference Materials

Non-hyperlinked items can be found in Canvas, in the "Lab_Files" folder of the Files area.

- https://www.ti.com/tool/EK-TM4C123GXL
- CortexM_InstructionSet.pdf
- Tiva TM4C123GH6PM Microcontroller spms376e.pdf

Overview

Welcome to the labs for your EENG260 course! Here, you will be putting what you have been taught in class to use, performing a variety of tasks using the Tiva C board and, eventually, additional hardware.

The first thing to note is the format of these lab writeups. Each will begin with a "Tasks for this lab" section, which details what I will be looking for when I grade each lab. If you are ever in doubt as to whether something I say changes those requirements, ask – as a rule, I don't change the required tasks mid-lab, but I will be sure to announce to the entire lab if that need does arise. Similarly, if something isn't listed in that section, it shouldn't be done – in particular, activating unnecessary hardware is a good way to lose points.

The next section of each lab will be a "Reference Materials" section, containing links to files and web pages that you will either need or find helpful. For example, the first item listed on this page is a link to the Texas Instruments (TI) website for the Tiva C board we will be using for this class. You can get this board through other sources, such as Digi-Key, but make sure you get the right board. Other boards may have similar capabilities, but different assembly languages and register/memory layouts.

After those first two sections, there will be one or more sections with more detailed instructions and helpful hints. Then, the final section, "To turn in", details what needs turned in for grading, and in what format. Generally, that will be files containing your lab writeup and whatever source code you wrote. Be sure to answer the questions posed there for your writeup, and check your files for completeness and correctness before submitting.

Now, let's get set up to do these labs.

Code Composer Studio

We will be using Code Composer Studio (CCS) for all the programming work in our labs, whether in assembly or in C. The first thing you need to do is to set up a workspace folder to hold your projects. Your best option is to use the Google Drive application on your computer to connect to your Google Drive, then create a folder there, named something like 'EENG260 Workspace', that you can easily find later. This will allow you to have access to the same workspace in the lab, at home, or wherever you have access to the Internet. The only downside is, you will want to be sure to check, both before you start and after you finish using CCS, that any changes have been synced with Google, or you could lose your work. In Windows, you can click on the Google Drive status icon in the Windows toolbar, and look for the line at the bottom that says "Everything is up to date".

Next, you will need a copy of CCS. If you are using one of the lab computers, they all come with CCS pre-installed, and you can proceed to the next paragraph. Otherwise, you can download CCS through the TI website link for our Tiva C board. You should choose the "on-demand installer", which will let you download just what you need for our microcontroller (TM4C123GH6PM) and not all the support files for all of TI's other microcontrollers. Once you have installed CCS, you should be ready to proceed.

The first time you start CCS, you should be asked where to find your workspace folder – let CCS know where to find the folder you set up on Google Drive, and you should be good to go. If, for whatever reason, you don't get asked where to find your workspace folder, you can tell CCS to switch to your workspace folder. That option is under the File menu, labeled "Switch Workspace".

With CCS running and your workspace folder selected, you are ready to create a project. One way to do that is to right-click in the "Project Explorer" panel on the left-hand side of the CCS window, then select New > CCS Project. On the window that comes up, you need to enter four pieces of information:

- At the top of the window is a pair of dropdowns for Target. In the left dropdown, select "Tiva C Series", then select "Tiva TM4C123GH6PM" in the right dropdown. This tells CCS the specific microcontroller we're using for this project.
- Directly below the Target dropdowns is a dropdown labeled Connection, which you want to make read "Stellaris In-Circuit Debug Interface". This tells CCS how to talk to (mainly, program and debug) our microcontroller over the provided USB cable.
- A bit further down is the Project Name field. Name your project what you like (Lab1 is a likely candidate), just be sure your project name starts with a letter – some peculiar and hard-to-troubleshoot issues may arise otherwise.
- Finally, in the "Project templates and examples" section, choose "Empty Project" (more specifically, the one that says just "Empty Project" under the "Empty Projects" category), then hit the Finish button at the bottom of the window.

That all done, your new project should show up in the Project Explorer panel. Clicking the arrow next to the project name expands the project to show its files and folders, several of which have already been set up for you. As-is, these files are configured to start up a "main" program defined by you, whether that's the "void main (void)" you remember from C, or a memory location defined by a "main:" label in assembly. We could change that to whatever we like, but main works, so we'll just stick with that, when we start our first lab.

Assembly instructions

Your first labs will be written in assembly language, a collection of mnemonic codes that closely mirrors the actual opcodes that drive the behavior of your microcontroller. Compared to a high-level language like C, it seems very limited – for example, where in C you can type something like "x++;", that same command would take at least three separate assembly instructions to complete. However, high-level languages are converted into assembly as part of the compilation or interpretation process.

You will be taught the basic structure of an assembly program in class, but there won't be time to cover every possible instruction in detail. Even the small subset of instructions that you will need for these labs would take too long, so you will need to look up some instructions on your own. At this point, the main thing you need to worry about is what, in general, each instruction does, without getting bogged down by all of the different options available for each instruction. For example, the ADD instruction takes the contents of a register, adds something to it (either the contents of another register or a constant), then stores the results in a register. With that level of knowledge, you will be able to plan out how you want to use the instructions available to you to perform the task you have been given, and work out the exact details by what the CCS assembler will let you do.

So, without further ado, here is a list of instructions that will be useful for completing your labs. If you have the textbook, details on each instruction are in Appendix 4, with more user-friendly descriptions available in chapter 3. Otherwise, the "CortexM_InstructionSet.pdf" document from the Reference Materials has details on these instructions in sections 2, 3, and 9.

ADD AND B BIC BL BX CMP EOR LDR LSL LSR MOV MVN ORR STR SUB

There are many, many more instructions available, and others may be covered in class, but these should be enough to cover every task you will meet in these labs. Of particular note, LDR and STR are the *only* instructions in this list that will let you interact with memory or memory-mapped registers, and B is the *only* instruction for doing loops and if/else statements that we will cover (other methods add complexity that we neither want nor need for this course). BL and BX, we will use exclusively for calling and returning from functions. The rest simply work on the contents of one or two of the "core" registers (R0-R12), mostly producing a result that will need to also be stored in a core register.

To turn in

No submissions are required for this lab, beyond notifying me of the members of your lab team, if appropriate. Each lab builds on the previous lab, so you still want to be sure to complete this lab before we begin Lab 1.