EENG 260 (Microprocessor/Microcontrollers

HW0

Assembly, Pseudo Code, Instructions Set and Libraries

**Problem 1:** Assembly language

Assembly language is a low-level programming language compared to writing code in C, Python, C++, Fortran, Pascal etc. Writing code in any of the high level programming languages is appealing as you do not need to know the machine specifics i.e. CPU registers, addressing memory in a specific and precise manner. Why we persist programming in Assembly?

**Assembly allows the greatest degree of control over the device being programmed. Some embedded devices don’t have C compilers. Understanding assembly is a good foundation for understanding the performance of high-level code.**

**Problem 2:** Programming Models

We have direct register access model and software driver model. What is the difference between the two programming models? Which would you select for this course?

**Direct register access: total control over the hardware. Program with a datasheet on your lap for register descriptions and addresses. I actually quite like programming this way and have done it for my own projects quite a few times.**

**Software driver model: Give up total control, use an existing library to control the hardware.**

**It sounds like in this class we are using direct register access for the assembly labs, and software driver model (with the ti library) for the C labs. I would go all-direct-register for the greater learning possibilities.**

**Problem 3:** Significance of flow chart or pseudo code

Why is it necessary to produce pseudo-code or a program flow chart before writing code in whatever language?

**A flow chart or pseudo code can be either a communication tool (Discussing the planned operation of a piece of software or hardware), or a learning tool for those new to programming. It can be very helpful for a new programmer to produce flowcharts and pseudo-code to help them learn to approach programming in an organized manner. In practice, how often do software engineers write pseudo-code or produce flowcharts before actual code? In my opinion flow charts should only be created for top-level business logic, and then only when communicating is absolutely necessary. Otherwise, a flow chart will VERY quickly become outdated – who is going to update a power- point related to a function when they make changes to the function? PERSONALLY, I usually start by writing all my function prototypes with docstrings that describe what the function does at a high-level (not pseodocode level) in a sentence or two. The docstrings let me organize my thoughts about how the program will behave, have lasting value, and are quick to update. Then creating the program is a matter of grabbing the function prototypes one at a time and doing the thing described in the docstring.**

**Problem 4:** Instruction Set

Of what value/use is the instruction set architecture of an MCU?

**The instruction set architecture (ISA) describes the function of the processor core on the MCU – which registers there are, how memory is addressed and accessed, what the endianess of the system is, and what instructions are available on the system. IE, Most every system has “ADD” available, but many embedded devices do not have “MUL” available. What assembly instructions are available is defined by the ISA. I’ve read the RISC-V ISA document pretty intently – I wrote a partially functional (only some of the instructions) RISC-V cpu core emulator. The document contains all the information needed to write that emulator. The cortex m4 ISA document is here: https://developer.arm.com/documentation/ddi0439/b/CHDDIGAC**

**Problem 5:** The Tiva C Series Development Board

Why do we have two TM4C123GH6PM microcontroller ICs on a single development board?

**We never discussed this is class, but I’m fairly certain that one of them handles talking to your computer, acting as a debugger/programmer. The other one is the actual MCU that we are programming. That’s why there are two USB ports.**