Lab 1

Kyle Swanson

February 2, 2015

To begin programming with our new TI Stellaris Launchpads, a logical starting point is the boot loader and the most simple aspects of the processor. This lab covered some basic assembly, and using assembly to load information into the different registers. Just as important, we got some experience with the development environment, IAR Embedded Workbench.

Since it had been well over a year since I had any exposure to registers, I needed to give myself a quick refresher on how they work and what their purpose is. Put simply, a register is a very small piece of memory, where computer systems usually put their data for doing operations on it. Some registers contain special information, the Stack Pointer, Program Counter, and Interrupt Program Status Register are all examples from this lab. Note figure 1 for examples of different registers.

These experiments nicely illustrated how registers work. The program we used simply loaded itself, then did an infinite loop. By clicking step by

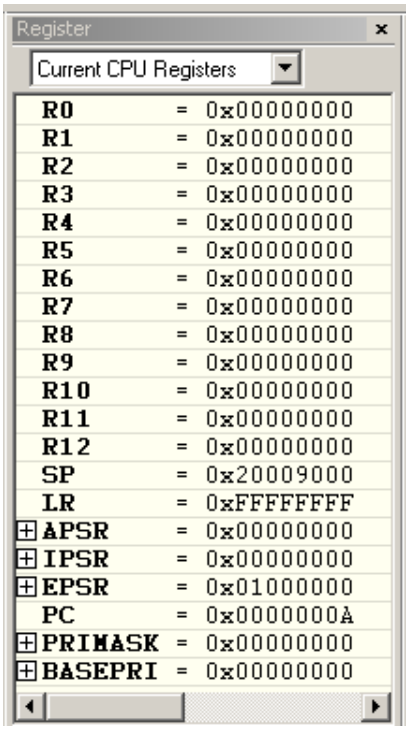

Figure 1: Some of the registers on the TM4C123G

step through the simple program we could watch as it loaded itself into the program counter register, which was one of the most interesting aspects of this lab. When the program loaded, you could watch as the program counter stepped through different addresses, *0x0000000A*, then *0x0000000C*, and so on. This register tells the processor which address to read instructions from.

At one point, we copied the program from its initial location in memory, and copied it to a different location in memory. Then, you update the program counter register to the new location of the application. This was a difficult task to do, you needed to ensure that the program was allowed ac-

cess to the portion of memory you were copying it to, and you had to find the program in memory to copy. These problems made me appreciate the complexity of this system. A part that continually confused me across all the experiments was the stack counter. While we learned the absolute basics about the stack pointer in earlier classes, in practice I found it difficult to relate the assembly commands to their actions.

A major aspect of this lab was how computers handle faults, and in the case of this program, how they don't. At its most basic level, a software fault is when a program tries to go outside of its restricted memory. We encountered problems with this while we were trying to copy data between different parts of the program. At times we were completely denied, through the semester, I will have to learn more about memory allocation and how the system gives memory to applications. One register that is closely related to faults is the IPSR or Interrupt Program Status Register. According to the ARM documentation, this register *"...contains the exception type number of the current Interrupt Service Routine (ISR)."*. Which then relates a interrupt to an fault or other status of the system. This is a register I will need to watch more closely in future labs, it was not until later that I realized the full meaning of this register. The following is a few of the status codes that may be loaded into the IPSR.

- 1 = Reserved
- 2 = NMI

- 3 = HardFault

I found that generally, when instructions were set away from the defaults that the system pushes you towards, you got strange and unexpected results.

These experiments were a good way to get a very basic understanding of ARM processors and the TI platform and work environment. Having more questions in my head than when I started will help me watch and learn in more specific ways in future labs, than simply being lost. Which happened a few times in this initial lab.