Lab 1: Boot Loader

Computer Architecture

Scott Overholser

WESTMINSTER

SALT LAKE CITY • UTAH

This lab was written for the Westminster College Computer Architecture course Spring 2015 semester.

Last Modified: Tue Jan 20 11:37:48 2015 -0700

Table of Contents

1 Lab Objective

The primary objective of this lab exercise is to understand the sequence by which the TM4C begins program execution from a power on reset.

Secondary objectives include the following:

- Introduce the Stack Pointer, Program Counter, and the IPSR.
- Get hands-on experience working with a microprocessor at the lowest level.
- Practice looking up technical information in the product data sheet.
- Introduce the IAR Embedded Workbench for writing and debugging assembly language programs.
- Introduce the ARM Cortex M4F assembly language.

2 Notes

You will see memory locations written in the form `0x0000.0000` in this lab exercise because it is easy to read accurately. But that is not a valid representation of a memory location. In the code you will see this represented as `0x00000000U` with the `U` meaning *unsigned*. Specifying unsigned in your code is not always necessary but it is a good habit.

2.1 Reference Material

Tiva TM4C123GH6PM Microcontroller Data Sheet will be referenced often in this and future labs. It is in the Canvas Files area filed under Tiva C Series Microcontroller. The filename is `tm4c123gh6pm.pdf`.

Refer to the following on the TM4C data sheet:

- *2.3.4 Register Descriptions* on page 76.
- *Register 14: Stack Pointer (SP)* on page 78.
- *Register 16: Program Counter (PC)* on page 80.
- *Register 17: Program Status Register (PSR)* on page 82.
- *2.5.2 Exception Types* on page 102.
- *5.2.2.2 Power-On Reset (POR)* on page 214.
- *8.2.2 ROM* on page 526.
- *8.2.2.1 Boot Loader Overview* on page 526.
- *Register 38: Boot Configuration (BOOTCFG), offset 0x1D0* on page 581.

3 Experiments

3.1 Create A New Project

Begin by selecting `Project->Create New Project...` Expand `asm`, select the second `asm` for *Empty pure assembler project.* Navigate to where you want the project to be saved. Create a folder as needed. Perhaps you should name the folder `Lab_1`. Then name the project. I suggest that you name the project `project.ewp`. Similarly, when it comes time to save your workspace, I suggest you use the name `workspace.eww`.

If you are using a VirtualBox virtual machine, consider using shared folders to select a location on your *host* operating system so that you do not lose your work if your virtual machine has problems.

3.2 An Infinite Loop

Begin this lab exercise by transcribing the following assembly language program into the IAR Embedded Workbench.

```
        PUBLIC  __iar_program_start

        SECTION .intvec:CODE (2)
        DATA
        DC32    0x20008000U
        DC32    __iar_program_start

        SECTION LAB1:CODE (2)
        THUMB

__iar_program_start
        B        main

main    NOP
        B        main

        END
```

Select `Project->Rebuild All` to compile the program. Did it compile without errors or warnings? If not, fix your program.

Select `Project->Download and Debug`. The IAR Embedded Workbench will at this time flash your compiled executable to the flash memory on the Stellaris LaunchPad and then begin debugging at the first breakpoint.

Select `View->Register` and `View->Memory`. These options are available only when debugging. When you save your workspace, these selections will be remembered.

3.3 Program Counter

The initial *program counter* is known as the reset vector. This is the 4-byte value located at memory location `0x0000.0004`. This tells the processor where to begin executing code. The value found at that location is loaded into the `PC` register.

In our first program, we used `__iar_program_start`. This symbol is replaced by the IAR Embedded Workbench linker with the location of our code in memory.

3.3.1 Modify Initial PC 1

As an experiment, replace the reset vector location `__iar_program_start` with the location of the ROM boot loader `0x0100.0000`. In other words, replace the first line below with the second in your program.

```
DC32    __iar_program_start
DC32    0x01000000U
```

Rebuild the program and then download and debug.

The IAR Embedded Workbench will display a warning message about the `__vector_table` symbol table. That is okay. Click through. You will find that your program is running. Click stop and examine register `PC`. What do you see there? What color is the memory view and the register view?

Expand `IPSR`. What is the value of `IPSR`? What is the value of `Exception Number` under `IPSR`?

3.3.2 Modify Initial PC 2

Now try replacing `__iar_program_start` with `0xFFFF.FFFF`. Rebuild the program. Then download and debug. Your program should be stopped at the correct breakpoint on `main` now.

3.3.3 Relocate Your Program

Return now to the first program. Rebuild, then download and debug. Locate your program in the Memory View. It should be very small. Locate a suitable section of RAM and copy your program there. Should it start on an even word boundary? Once you copy your program, enter the new memory location into the `PC` register. Single step through your program in it's new location.

Are the instructions executed as before? Describe what happens in the `PC` register.

3.4 Minimum Viable Executable

We can strip down the previous program as follows:

```
        PUBLIC  __iar_program_start

        SECTION LAB1:CODE (2)
        THUMB

__iar_program_start
        B       .
```

```
    END
```

Replace your previous code with the code listing above. Compile and begin debugging. You will again see the `__vector_table` warning. Just click through. What is your program doing?

4 Lab Report

Write a report of your experience with this lab exercise. Include meaningful observations that you have made. As you do so, keep in mind that the focus of this course is *Computer Architecture*.

The primary objective of this lab exercise is to demonstrate the Stellaris LaunchPad boot loading process. Explain the process.

In your reports, consider each of the experiments. What was the experiment intended to illustrate? How did it work for you? Did you have any trouble? How large was each program in bytes?

As you write your report, channel Joel Spolsky and explain your ideas in terms a non-CS student can understand. Write your report so that it flows naturally. Do not simply enumerate a list of answers to the questions below. Address some of your own observations that do not appear in the list.

Questions that you should be answered in your report:

1. What is a register?
2. What is the Program Counter and what is the PC register?
3. What is the Stack Pointer and what is the SP register?
4. What is an IPSR? What is the IPSR register?
5. What is a fault?
6. What is the correct approach to handling a fault condition?
7. Can you make the processor stop executing instructions? Explain?
8. What happened when the executable instructions were located at `0x0000.0000` where the Stack Pointer is expected to be found.
9. When the flash memory at `0x0000.0004` contained `0xFFFF.FFFF`, the TM4C did not behave as the data sheet describes. Speculate on why.
10. Relate your experience with this exercise to the subject of Computer Architecture.