ECE 1181: Microcontrollers and Embedded Systems Lab
Fall 2024 – Professor Joe Camp

Lab 2: Tooling Up (9/5 or 9/6)
(Due before the start of the next lab – 9/12 or 9/13)

Reminder: Each lab report should have the following.
1. Name, SMU ID, and Lab Session Number
2. In your own words, the objective of the lab assignment and description of code you use and write. If writing an algorithm, provide pseudocode. (25%)
3. The actual code and in-line comments (25%)
4. In your own words, answers to any questions on the lab description, results, screenshots, and verification (50%)

Reports must be submitted before the start of the next lab.

Description:

Last lab, we could not see any values of the registers in the ARM7TDMI data path. While the textbook Raspberry Pi Assembly Language Programming (1st edition) by Stuart Smith begins working on different instructions (MOV, ADD) in Chapter 2, the ability to see registers via the debugger is not taught until Chapter 3. Hence, we are going to start with understanding the debugger in this lab so that we can see values being changed in registers per line of executed ARM assembly code. We will follow Chapter 3 (Tooling Up) very closely. In particular, the lab consists of the following steps:

1. We will first be learning about how to create a makefile so that we can assemble all of the assembly source code in the directory with one command (make). We will transition from building our Hello World Program from Chapter 1/Lab 1 from using a build file to using a makefile. Follow the steps on pages 53-56 to do so. Once you have the final version working that uses the OBJS variable, then modify your HelloWorld.s source code to have your output "Hello <first name> <last name>! (built from a makefile)''. Show that you actually build it using "make" and run it as you did before. Perform a print screen and include in your report.

2. Now, observe the code given below in the "Code Given" section. This code is the only relevant portion of Chapter 2 that is referred to in Chapter 3. We will more thoroughly go through the code in Chapter 2 in Lab 3 when we have the ability to use the debugger (learned in this lab). You can call this code movexamps.s to be consistent with the steps from Chapter 3 that follow.

   The code given below (now called movexamps.s) places the value of 0x4F006E into the R2 register. To do so, two different versions of move instructions are used to first fill the bottom 16 bits of R2 (MOV R2, #0x6E) and then the top 16 bits of R2 (MOVT R2, #0x4F). Note that the Smith textbook uses a MOVT instruction that is not mentioned or used in the Hohl

textbook. We will discuss other ways to fill a register with a value that does not immediately fit into an instruction. This is just one way.

Follow the instructions on pages 57-62. After you are comfortable with this process, open the source code for movexamps.s and modify the 32 bit value that is being placed into R2 with a hexadecimal version of the first four digits of your student ID. For example, if your student ID is "12345678", then you would use the value "0x120034" for the one that will eventually end up in r2. Show building it using "make" and run it as you did on pages 57-62 with a screen capture of the terminal once you complete the steps, to be included in your report.

3. Lastly, add the same line of code but using increasing register names. For example (where "0x120034" represents the first four digits of your student ID in hex form):
   MOV R2, #0x34
   MOVT R2, #0x12
   MOV R3, #0x34
   MOVT R3, #0x12
   MOV R4, #0x34
   MOVT R4, #0x12
   MOV R5, #0x34
   MOVT R5, #0x12
   Show that you can set a breakpoint to the other lines in addition to the ones that are modifying R2, run to those breakpoints, step over those lines where the new breakpoint is placed, and display the contents of both the registers and memory. Take a screen capture of the terminal when you are done and include in your report.

Code Given (WARNING: I strongly recommend typing this into your Red Pitaya directly – The reason is that there are hidden symbols that may cause a high degree of frustration from copy/paste that would be spared by just typing it in. Also, you get a better feel for the code in doing so.):

```
@
@ Examples of the MOV instruction.
@
.global _start @ Provide program starting address
@ Load R2 with 0x4F006E first using MOV and MOVT
_start: MOV R2, #0x6E
        MOVT R2, #0x4F
        MOV R0, #0          @ Use 0 return code
        MOV R7, #1          @ Service command code 1
        svc  0              @ Call Linux to terminate
```

You have completed this lab when:

Demonstrate to your lab instructor that you have completed the steps above.