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

Lab 5: Literal Pools and Logic (2/22 or 2/23)
(Due before the start of the next lab – 2/29 or 3/1)

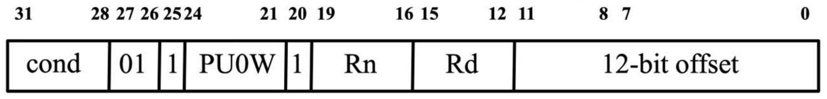
Reminder: Each lab report should have the following.
1. Name and SMU ID
2. Lab Session number & objective of the lab assignment
3. Description of the program and if writing new code, algorithm in pseudocode (25%)
4. The actual code used comments (25%)
5. Answers to any questions on the lab description, results, screenshots, and verification (50%)

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

Description:

Last lab, we skipped over the section in Chapter 5 of the Smith textbook on PC relative addressing (pages 92-95) because it was mentioning concepts dealing with literal pools that we had not yet dealt with in class.

Key Point (Literal Pools): Distance from LDR pseudo-instruction to constant cannot be longer than 4096. The reason for this is that the offset from the PC in the load instruction is 12 bits. Hence, 2^12 is 4096, which is the maximum number of bytes from which maximum value that could be added to the PC to get to the memory location of the constant. Thus, if there is a large program or data structure in between the LDR pseudo-instruction and the location of the constant, there would be an error when the assembler tries to compile the program that the literal pool needs to be closer (*i.e.*, the constant cannot be reached).



We then tiptoe our way into Chapter 4 of the Smith textbook, focusing first on the logic portion of the chapter. We focus only on logic portion of Chapter 4 on the use of AND, EOR, ORR, and BIC (pages 75-77) or CMN, TEQ, and TST (pages 84-85). It will be up to you which instructions make the most sense in your program. You will be taking in a single ASCII alphabetical character and switching the case (*e.g.*, upper to lower case or lower to upper case). Your program must do both but you can assume that the character being input to your program is either one or the other (you can check the boundary between these ASCII values to know how to convert it). There are many ASCII tables online or one in Appendix C of the Hohl textbook.

Unlike prior lab assignments, after reading the relevant portions of the Smith text closely, we will only loosely follow the steps in the textbook. In particular, the lab consists of the following steps:

1. Read through pages 92-95 on PC relative addressing.
2. To understand this more deeply, take your original Code Given in Listing 1-1 of the first lab assignment for the "Hello World!" program. Create a

makefile that allows debugging and disassembly so that you can find the memory address of where the "Hello World!" string is located in memory as a result of the literal pool created by the *LDR r1, =helloworld* line of code. Show a screen capture of the memory where the string is located and discuss in your report the memory range that the string consumes.

3. Now, create some separation in memory between your code that ends with the last *svc 0* statement and the .data section. To do so, you may use assembler directives or additional ARM instructions. However, you should know the number of bytes of memory that you are adding to the original size of the program.
4. Remake the program and inspect where the "Hello World!" string is now located in memory. Does this line up with your expectation for the amount of space that you put between the end of the svc 0 statement and the .data section? Please explain in your report.
5. Next, add enough bytes in between the last svc 0 statement and the .data section that the assembler will not successfully compile and report "Error: invalid literal constant: pool needs to be closer". Include a screen shot of this occurring in your report and describe how you made it occur.
6. Now, you will move to a second program. In the Code Given section, there is code that takes in exactly 1 character and then echos that character back to the user to ensure that the program "heard" the user correctly. You will notice that the string to stdout is 2 characters to include the end of line and print nicely on the terminal. Take this code and create a source file for it such as inputASCII.s.
7. To ensure that your source code is running correctly, all you really need to do is build it as we did with the first program in the first lab assignment with a build file. However, so that you can debug the latter steps, go ahead and create a makefile as you have done for the other lab assignments. Then, make the program and run it without the debugger so that it can see how the stdin and stdout are operating. For example, just run "./inputASCII" at the command line.
8. Once you have a feel for this, know that you could also step through with the debugger to the "svc 0" and when this line is stepped over, it would ask for input in the debugger.
9. From here, are given some freedom in how to implement this, but ultimately, you will need to take any capital letter ('A' to 'Z') and convert it to lower case or lower case letter ('a' to 'z') and convert it to upper case. To make it easier on yourself, you do not need to check if it is an alphabetical character being input. You can assume it is a letter being input (alphabetical ASCII value). However, you do not know beforehand if it is a capital letter or lower-case letter being input.
    a. You will need to input your functionality between the stdin and stdout portions of code.
    b. Hint: Notice that the range of ASCII values for 'A' to 'Z' is 0x41 to 0x5A and the range for 'a' to 'z' is 0x61 to 0x7A.

i. First – think about the mathematical operation of converting from one to another.
ii. Second – I would recommend that you pick 'Z' or 'a' and, since you are guaranteed that the input is a letter, you know that below this range is upper case and above this range is lower case.

c. Use conditional execution so that you do not need any branches.

10. Describe your algorithm and include this in your report.
11. Test specifically the initial of your first name in lower case, then upper case. Then, test the initial of your last name in lower case and then upper case. Include a screen capture of these test values and your program operating correctly.

Code Given: Listing 1-1 from Chapter 1 of Smith text (original "Hello World!" program) for steps 1-5 above. The following code will be for steps 6+:

```
@
@ Assembler program to input ASCII characters via stdin
@ The program will transform a lower-cased character to upper case
@ or an upper case character to lower case
@ R0-R2 - parameters to linux function services
@ R7 - linux function number
.global _start @ Provide program starting @ address to linker
@ Set up the parameters to retrieve the character and then call Linux to do it.
_start:         mov R0, #0          @ 0 = StdIn
                ldr R1, =msg        @address to receive input
                mov R2, #1          @only receive one character
                mov R7, #3          @read
                svc 0               @ Call linux to receive input
@ Set up stdout to echo that character back and then call Linux to do it.
                mov R0, #1          @ 1 = StdOut
                ldr R1, =msg        @ string to print
                mov R2, #2          @ length of string (include '\n' to print nicely)
                mov R7, #4          @ linux write system call
                svc 0               @ Call linux to print
                mov R0, #0          @ Use 0 return code
                mov R7, #1          @ Service command code 1
@ terminates this program
                svc 0               @ Call linux to terminate
.data
msg:            .ascii "0\n"
```

You have completed this lab when:

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