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

Lab 6: Controlling Program Flow (10/3 or 10/4) (Due before the start of the next lab –10/17 or 10/18

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:

Previously, the program counter always moved to the next instruction (PC = PC + 4). The +4 came because each instruction is 4 bytes in size (32 bits). However, the key distinction in this lab assignment is that the program counter can now have its value changed to an arbitrary location in memory, thereby changing the control flow of the program to that location. There is one major restriction on this movement, however. The location that it moves to must be evenly-divisible by 4 (*i.e.*, the new location of the PC must be *word aligned*). The reason for the word-aligned requirement is that if the rule was not enforced, there could be the last byte(s) of one instruction smashed together with the first byte(s) of another instruction to form a garbage instruction that would just be a concatenation of 4 somewhat-arbitrary bytes. If the program counter were to be set to such a location, it would fetch that garbage instruction, attempt to decode it, and perhaps execute it. The reason that the last two steps are undetermined is because the garbage instruction could be undefined (the safest outcome) or it could represent an actual instruction. The latter is more dangerous because it is harder for the programmer to detect such a problem, thereby potentially having a more profound impact.

Key Point: Program Counter not just incremented anymore. Instead of moving the program counter sequentially, there now is the possibility of *branching* to a new location. The branch changes the program flow to the new location, and the pipeline must be refilled before an instruction at the new location can be executed. In other words, what previously existed in the pipeline to be decoded and executed next is no longer relevant and those values are *flushed*. The instructions at the new control-flow address must be fetched and decoded before being executed.

For this lab, we will now thoroughly read Chapter 4 of Smith textbook. I suggest doing the examples. However, the actual code implemented in the assignment is a continuation of the previous lab in various forms. In particular, the lab consists of the following steps:

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

1. Assuming you have already gone through Chapter 4 of Smith and examples, use the Code Given below. This is similar to that given for the Lab 5 assignment, however, this code allows an input string of up to 50 characters. Create a source file (e.g., "inputASCII.s"), compile it, and run it (./inputASCII) to ensure that it echos your input for up to 50 characters. If you have not done so already, create a debuggable makefile so that you can step through to debug any problems you have in the steps below.
2. Take your program that has successfully echoed a string of input characters to the terminal and add the following functionality: ensure that words are output with all capital letters. To do so:
    a. You will need to input your functionality between the stdin and stdout portions of code.
    b. There, you will need a loop that progresses through the array of characters that is now stored at the *msg* label.
    c. You will need at least one status variable (register that is not otherwise being used) to keep track of where you are in the string. You could do this by moving the base register through the string or keeping the base register at the same location and moving an offset register.
    d. In each iteration of the loop, you will need to:
        i. Check if the character is alphabetical. If so:
            1. Make sure it is capitalized.
            2. Otherwise, change the character to a capital letter and store it back at the same location.
        ii. Otherwise (if it is not an alphabetical character), leave any numeric characters or symbols alone.
        iii. Update the status variable to keep track of the number of characters before the '\n' (end of line) for the next iteration of the loop.
3. Describe your algorithm and include this in your report.
4. Include a screen capture of some test values and your program operating correctly. Input various upper and lower case mixtures of your first name, last name, and your student ID to show that your program is working correctly.

Code Given: Similar to that given for lab 5 but now allows a string of up to 50 characters to be input.

```
@
@ Assembler program to input ASCII characters via stdin
@ The program will take a string of up to 50 characters and modify it
@ such that all lower-cased letters will be made upper-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 characters and then call Linux to do it.
```

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

```
_start:         mov R0, #0          @ 0 = StdIn
                ldr R1, =msg        @address to receive input
                mov R2, #50         @receive at most 50 characters
                mov R7, #3          @read
                svc 0               @ Call linux to receive input
                mov r2, r0          @captures number of characters input

@ Insert your code here

@ Set up stdout to echo those characters back and then call Linux to do it.
                mov R0, #1          @ 1 = StdOut
                ldr R1, =msg        @ string to print
@ mov R2, #2 @ commented out because you will need to calculate the string length
                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:            .fill 51, 1, '\n'   @ Allocates enough memory for 50 characters
```

You have completed this lab when:

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