ECE253 – Laboratory Exercise 7 and 8

Using the ARM A9 Processor

Preparation and In-Lab Marks

Preparation marks: For Lab 7, do Parts I (tutorials) and II of the exercise. Print out your assembly language for Part II and provide it to your TA at the beginning of the lab. For Lab 8, do the remaining parts of the lab and provide printouts of your assembly language for Parts IV and V to your TA at the beginning of the lab.

In-lab marks: For Lab 7, you are required to demonstrate to a TA Part II. For Lab 8, you are required to demonstrate to a TA Parts IV and V.

This exercise provides an introduction to the ARM A9 processor, its assembly language, and development tools.

To prepare for this exercise you need to be familiar with the ARM A9 processor architecture and its assembly language. This processor is described in the tutorial called *Introduction to the ARM Processor*, which is available on the Piazza site. In Section 5.1, it is *highly recommended* you ignore the pre-indexed and post-indexed addressing modes for now. You should also ignore Section 6.1.1 on loading multiple registers. Likewise, for now, ignore Section 6.3.1 and Section 6.10 on conditional instructions. The ARM processor is also described in Appendix D of the textbook for ECE253.

For this exercise, you will use an ARM A9 processor that is part of a computer system called the *DE1-SoC Computer*. This computer system is implemented inside the Altera Cyclone V SoC chip on the Altera DE1-SoC lab board.

A simulator for the ARM processor on the DE1-SoC chip has been created by one of our recent Ph.D. graduates, Dr. Henry Wong. The simulator is available here: `https://cpulator.01xz.net/`. On the landing page, as the system, select "ARMv7 DE1-SoC". You will then enter an environment where you can type assembly code and execute it. Use the simulator to develop and test the solutions for labs 7-10 of ECE253.

Part I

For the in-lab portion of this exercise, you will need to use the Intel Monitor Program for ARM, which will allow you to execute your assembly programs on the *actual* ARM processor on the Cyclone V chip. To prepare for this, read the tutorial for the monitor program, available on the Piazza website. Read Sections 1 and 3 of the tutorial *before* coming to the lab. It is *not* mandatory that you install the monitor program on your own computer (of course, you can if you want to).

Part II

In this part, you will write your own ARM assembly language program that adds together a list of positive numbers and deposits their sum into register R7, as well as counting the number of positive numbers and depositing the count into register R8. The list of numbers is terminated by a -1; the size of the list is not known in advance. Skeleton code is given below. The list of numbers is at a memory location with label TEST_NUM. In this case, the value 21 (0x15) should appear in R7 when the program complete; the value 5 (0x5) should appear in R8 when the program completes. The TA may change the list to test your program.

1. Create a new, empty, folder to hold your Monitor Program project for this part. Create a file called *sum-nums.s*.
2. Write your assembly language code.
3. Compile and load the program. Fix any errors that you encounter. Once the program is loaded into memory in the DE1-SoC Computer, single step through it to see how the program works.

```
                .text
                .global     _start
_start:
...
... Write your code here
...

END:            B           END

TEST_NUM:       .word       1,2,3,5,0xA,-1

                .end
```

Part III

Perform the following:

1. Create a new, empty, folder to hold your Monitor Program project for this part. Create a file called *count1s.s*, and type the assembly language code shown in Figure 1 into this file. This code uses an algorithm to count the longest string of 1s in a word of data. The algorithm uses shift and AND operations to find the required result—make sure that you understand how this works.
2. Make a new Monitor Program project in the folder where you stored the *count1s.s* file. Use the DE1-SoC Computer for this project, and specify the assembly language file that you created (rather than a sample program as was done in the *Altera Monitor Program Tutorial for ARM*).
3. Compile and load the program. Fix any errors that you encounter (if you mistyped some of the code). Once the program is loaded into memory in the DE1-SoC Computer, single step through it to see how the program works.

```
/* Program that counts consecutive 1's */
            .text
            .global  _start
_start:
            LDR      R1, TEST_NUM   // load the data word into R1

            MOV      R0, #0         // R0 will hold the result
LOOP:       CMP      R1, #0         // loop until the data contains no more 1's
            BEQ      END
            LSR      R2, R1, #1     // perform SHIFT, followed by AND
            AND      R1, R1, R2
            ADD      R0, #1         // count the string lengths so far
            B        LOOP

END:        B        END

TEST_NUM:   .word    0x103fe00f

            .end
```

Figure 1. A program that counts consecutive 1s.

Part IV

Perform the following.

1. Make a new folder and make a copy of the file *count1s.s* in that new folder.
2. In the new copy of the *count1s.s* file, take the code which calculates the number of consecutive 1s and make it into a subroutine called ONES. Have the subroutine use register R1 to receive the input data and register R0 for returning the result.
3. Add more words in memory starting from the label TEST_NUM. You can add as many words as you like—but include at least 10 words.
4. In your main program, call the newly-created subroutine in a loop for every word of data that you placed in memory. Keep track of the longest string of 1s in any of the words, and have this result in register R5 when your program completes execution.
5. Make sure to use the single-step capability of the Monitor Program to step through your code and observe what happens when your subroutine call and return instructions are executed.

Part V

One might be interested in the number of 1s in the binary representation of a number, as well as the number of *leading* and *trailing* 0s in the binary representation of a number. For example, the 16-bit binary number 0010110101000110 contains seven ones; it has two leading 0s; it has one trailing 0. The number of leading 0s is the number of 0s on the left side of the number (before the first 1 appears). The number of trailing 0s is the number of 0s on the right side of the number.

Write a program that determines the following:

- The number of 1s in a word of data—write the result to register R5
- The number of leading 0s in a word of data—write the result to register R6
- The number of trailing 0s in a word of data–write the result to register R7

Make each calculation in a separate subroutine (called ONES, LEADING, and TRAILING). Call each of these subroutines in the loop that you wrote in Part IV, and keep track of the largest result for each calculation, from your list of data. Store the largest results for ONES, LEADING and TRAILING in registers R8, R9, and R10, respectively.