Laboratory Exercise 7 and 8

Using the ARM A9 Processor

This exericse 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 in the Computer Organization section of Altera's University Program website. 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 detailed description of the DE1-SoC Computer can be found in the document entitled *DE1-SoC Computer with ARM*, which can be found in the Computer Systems secton of Altera's University Program website.

Software to Install for the Computer Organization Lab Exercises

The Computer Organization lab exercises require additional software other than Quartus II. To use the ARM processor on your own computer, you need to have installed the software package called the *Altera Monitor Program*, as described below.

Part I

In this part of the exercise you will become familiar with the *Altera Monitor Program*. You will use this tool to develop software programs that run on the ARM processor. You will be able to compile your code, download the compilation results into the DE1-SoC Computer on the DE1-SoC board, and then execute and debug your code on the ARM processor.

Before you can use the Altera Monitor Program, it has to be installed on your computer. Instructions for installing this program are given in the document *Altera Monitor Program Tutorial for ARM*. Obtain this tutorial from the Computer Organization section of Altera's University Program web site.

After successfully installing the Monitor Program, perform the steps found in Section 3 of the tutorial to learn about the features of the tool. In these steps you will create a Monitor Program project that targets the ARM processor in the DE1-SoC Computer. You will use a sample of an assembly language program that is provided in the Monitor Program tool, and you will learn about compiling code, downloading compiled programs onto the DE1-SoC board, and executing and debugging programs.

You do not need to perform the steps in the tutorial after Section 3; those steps deal with advanced topics such as working with interrupts (we will learn about interrupts later in this course) and working with C programs.

Part II

In this part you will create your own assembly language program and execute it on the ARM processor in the DE1-SoC Computer.

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 III

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 IV

One might be interested in the longest string of 0s, or even the longest string of alternating 1s and 0s. For example, the binary number 101101010001 has a string of 6 alternating 1s and 0s. The number 1010 has a string of 4 consecutive 1s and 0s.

Write a program that determines the following:

- Longest string of 1s in a word of data—write the result to register R5
- Longest string of 0s in a word of data—write the result to register R6
- Longest string of alternating 1s and 0s in a word of data—write the result to register R7 (Hint: What happens when an n-bit number is XORed with an n-bit string of alternating 0s and 1s?)

Make each calculation in a separate subroutine (called ONES, ZEROS, and ALTERNATE). Call each of these subroutines in the loop that you wrote in Part III, and keep track of the largest result for each calculation, from your list of data.

Preparation and In-Lab Marks

Preparation marks: read the suggested tutorials, and write the required assembly language code for each part of the exercise.
In-lab marks: You are required to demonstrate to a TA Parts III and IV.