Lab3: ARM Cortex-A Processor Assembly Programming

In this lab, you will use the *Intel Monitor Program* to develop, compile, load, and run ARM Cortex-A assembly programs. Refer to the *Introduction to the ARM Processor* document (available in the Bb Major References folder) when necessary.

Part I

This part will explore some features of the Intel Monitor Program by using a simple ARM assembly program shown in Figure 1 on next page. Note that some sample data is included in this program. The word (4 bytes) at the label *RESULT* is reserved for storing the result of the program. The next word, *N*, specifies the number of entries in the list. The words that follow give the actual numbers in the list.

1. Create a new folder for this part with a name such as Lab3Part1. Use a text editor (e.g. notepad or wordpad) to type in the program shown in Figure 1 and save it as part1.s. Study the program instruction by instruction, and add a comment to each instruction (See the top of the program for the format). Then answer the following question:
Question #1: What's the purpose/function of this program? In other words, what does it try to accomplish given a list of values? Please write your answer as comments on the top of the program part1.s. under your name.

2. Perform the following:

1) Use the Monitor Program to create a new project in Lab3Part1; When you reach the *Specify a program type* window, choose Assembly Program and add part1.s to the project. Compile and load the program.

2) The Monitor Program will display a disassembled view of the machine code loaded in the memory, as indicated in Figure 2. The first column shows the addresses of the instructions and data. Note that the pseudo-instruction LDR R4, =RESULT from the source code has been implemented by using the instruction, LDR R4, [PC, #88], which loads the 32-bit address of the label RESULT into register R4. After this instruction has been executed, the content of register R4 will be 0x0000003C, because this is the address in the memory of the label RESULT.
Single step through the program by clicking on the icon . Watch how each instruction changes the data in the processor's registers.
Question #2: After the program is over, which register contains the result? Please write your answer as comments on the top of the program part1.s. Go to the memory tab, find the address of RESULT. **Take a picture of the memory tab and circle the RESULT contents. Include this picture in the zip folder when you submit the project.**

3) Change the number (*N*) of the entries to 10, and the seven given values to the following 10 numbers: -9, 1, 10, -5, 4, 12, -4, 8, 32, 5. Compile and load the program. After the program is ready to run, set a breakpoint at address 0x00000030 (by clicking on the gray

bar to the left of this address), so that the program will automatically stop executing whenever the branch instruction at this location is about to be executed.

Then run the program. Observe the contents of register R0 each time the breakpoint is reached. Does it work correctly? Note: Clicking on the restart icon  will make the program to go back to the beginning. Watch the register's value when the negative values are loaded. **Question #3:** How is the decimal number -5 represented in the computer? Describe how to manually convert the decimal number -5 to the binary value shown in the register.

```
            /* Program for Part 1.
             You may use // or /* & */ for comments.
             Labels must start at column 1 (leftmost)
             Programs must end with .end */

            .text           //code follows
            .global   _start

 _start:
            LDR       R4, =RESULT
            LDR       R2, [R4, #4]
            ADD       R3, R4, #8
            LDR       R0, [R3]

LOOP:       SUB       R2, R2, #1
            CMP       R2, #0
            BEQ       DONE
            ADD       R3, R3, #4
            LDR       R1, [R3]
            CMP       R0, R1
            BGE       LOOP
            MOV       R0, R1
            B         LOOP

DONE:        STR      R0, [R4]

END:        B         END

RESULT:     .word     0
N:          .word     7
NUMBERS:    .word     4,5,3,6,1,8,2
            .end
```

Figure 1: Assembly language program for Part I.

Part II

Modifying your part1 program so that it uses a subroutine for the function. The subroutine starts at SUB1. The main program passes the number of entries (i.e. value of N) and the address of

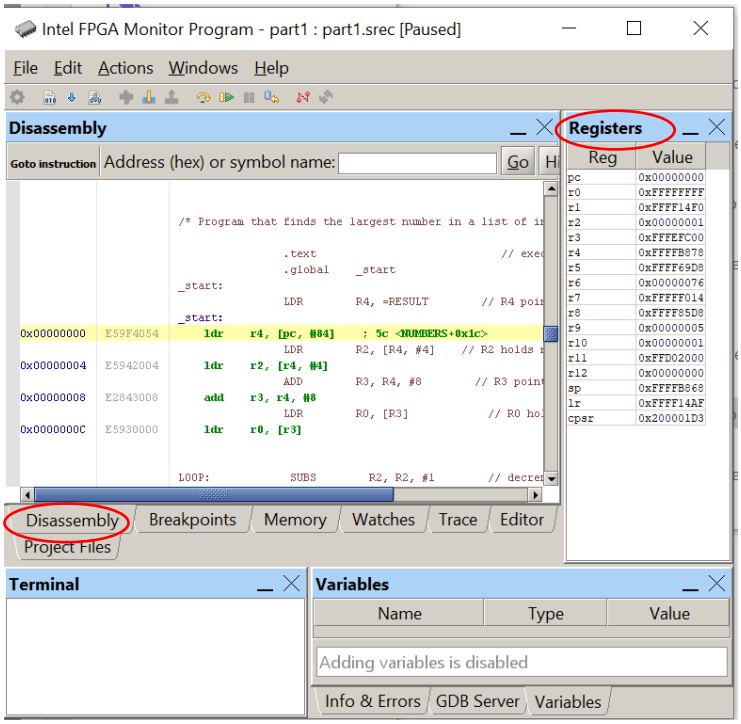

Figure 2: The disassembled view of the program in Figure 1.

the start of the list (i.e. address of NUMBERS) as parameters to the subroutine via registers R0 and R1. **The subroutine SUB1 returns the result to the calling program via register R0**. Test your part2 program on the board to verify if the memory location contains the correct value. A suitable main program is given below.

```
            .text
            .global _start
_start:     LDR R4, =RESULT
            LDR R0, [R4, #4]
            ADD R1, R4, #8
            BL SUB1             //call subroutine at SUB1
            STR R0, [R4]
END:        B END               //stop here

            // Your subroutine starts below
SUB1:
                                //instructions for SUB1 go here

            BX LR          // End of the subroutine; goes back
                                // to the instruction immediately
                                // after BL
                                    //
RESULT:     .word 0
N:          .word 7             // number of entries in the list
NUMBERS:    .word 4, 5, 3, 6, 1, 8, 2   // the data
            .end
```

Part III

Write an assembly program to display a decimal digit on the 7-segment display HEX0. The other seven-segment displays HEX5-HEX1 should be blank. Pressing a KEY has the following behavior:

- KEY 0. sets the display to 0
- KEY 1. increments the displayed number
- KEY 2. decrements the displayed number
- KEY 3. sets the display to be "blank" (You can assign the value -1)

You should also check the range of the number to display on the HEX to make sure it is between 0 and 9. When it is larger than 10 or less than 0, reset it to 9 or 0, respectively.

A template with a HEX subroutine is given in Lab3Part3.s (in the Lab Files folder) for you to start with. The subroutine assumes the number to show is passed in R0 by the main program.

The assembly code getting_started.s for the getting started example of Lab1 is also provided for your reference. For example, how to check if a KEY is pressed.

Pre-lab reading for Lab 4:

- Overview of the Generic Interrupt Controller (GIC) of ARM Cortex-A processors

What to submit:

Zip all the three respective project folders for the three parts and submit them in BB. Please make sure that your part1.s includes your answers to the questions, and that the zip file includes the picture showing the memory location of RESULT.