ECE 243S - Computer Organization
January 2023
Laboratory Exercise 1

Introduction to Assembly Language Programming
Using an ARM Cortex A9 System

The goal of this lab is to introduce you to the basic concepts of Assembly Language programming of a processor, which gives you insight into what a computer is actually doing when it executes a program. We will make use of the ARM® Cortex® A9 processor inside the *DE1-SoC Computer*. This computer system is designed for the DE1-SoC board and includes the processor, memory for holding programs and data, and various I/O ports. The ARM processor accesses each component in the system using memory-mapped I/O; the address ranges assigned to each of the various devices in the system are specified in the document called *DE1-SoC Computer System with ARM Cortex A9*, which is provided on Quercus along with Lab 1.

In this introductory exercise you will begin learning how to develop programs written in the ARM assembly language, which can be executed on the *DE1-SoC Computer*. You will need to be familiar with the ARM processor architecture and its assembly language. An overview of the ARM Cortex A9 processor can be found in the tutorial *Introduction to the ARM Processor*, which is provided with Lab 1.

To develop and "execute" programs for the ARM processor you will use two different software tools, described below: the *CPUlator* and the *Monitor Program*.

CPUlator: The *CPUlator* is a web-based software tool. It uses software techniques to *simulate* the functional behavior of the components in the *DE1-SoC Computer*, including the ARM processor, memory, and a number of I/O devices. You will find that the CPUlator is an excellent tool for developing and debugging ARM programs on your home computer—it is easy to use and does not require a DE1-SoC board. The CPUlator is introduced in Part I, below.

Monitor Program: The *Monitor Program* is an application software tool that allows you to develop and debug ARM programs that *execute in hardware* on a DE1-SoC board. You will need to use the Monitor Program to execute your ARM programs in the lab rooms at the University, where you will attach a DE1-SoC board to the computer at your workstation. You can also use the Monitor Program on your home computer, but only if you purchase a DE1-SoC board for use at home. Purchasing a board for home use is entirely optional, especially since the CPUlator provides a convenient way for you to develop ARM programs without access to a physical DE1-SoC board. The Monitor Program is introduced in Part V.

Part I

In this part of the lab exercise you are to watch a short video that provides an introduction to the *CPUlator* tool. As you will see in the video, the *CPUlator* is a full-featured software development environment that allows you to compile (assemble) and debug software code for the ARM processor. As mentioned in the video, we should all feel a huge debt-of-gratitude to the author of this tool, Dr. Henry Wong, who developed, and maintains, *CPUlator* as a volunteer effort on his own personal time. Thank you Henry!!

Use the following URL to access the video from Microsoft Streams, using your *UTOR* credentials:

https://web.microsoftstream.com/video/15948679-3895-4c1b-91fe-07f0c34c299b

Part II

Now, we will explore some features of the *CPUlator* by working with a simple ARM assembly language program. Consider the program given in Figure 1, which finds the largest number in a list of 32-bit integers that is stored in the memory.

```
/* Program that finds the largest number in a list of integers   */

        .text                   // executable code follows
        .global  _start
_start:
        MOV     R4, #RESULT   // R4 points to result location
        LDR     R2, [R4, #4]  // R2 holds number of elements in the list
        MOV     R3, #NUMBERS  // R3 points to the list of integers
        LDR     R0, [R3]      // R0 holds the largest number so far

LOOP:   SUBS    R2, #1        // decrement the loop counter
        BEQ     DONE          // if result is equal to 0, branch
        ADD     R3, #4
        LDR     R1, [R3]      // get the next number
        CMP     R0, R1        // check if larger number found
        BGE     LOOP
        MOV     R0, R1        // update the largest number
        B       LOOP
DONE:   STR     R0, [R4]      // store largest number into result location

END:    B       END

RESULT: .word   0
N:      .word   7             // number of entries in the list
NUMBERS: .word  4, 5, 3, 6    // the data
        .word   1, 8, 2

        .end
```

Figure 1: Assembly-language program that finds the largest number.

Note that some sample data is included in this program. The word (4 bytes) at the label *RESULT* is reserved for storing the result, which will be the largest number found. The next word, N, specifies the number of entries in the list. The words that follow give the actual numbers in the list.

Make sure that you understand the program in Figure 1 and the meaning of each instruction in it. Note the extensive use of comments in the program. You should always include meaningful comments in programs that you will write!

Perform the following:

1. Create an assembly-language source-code file for the program in Figure 1, and name the file *part2.s* (this file is included in the *design files* for this lab exercise). Then, open the *CPUlator* web-site and set its system parameters to choose the ARMv7 processor and DE1-SoC board (the corresponding URL for the *CPUlator* web page should be https://cpulator.01xz.net/?sys=arm-de1soc). As indicated in Figure 2, click on the File command near the top of the *CPUlator* window and then select Open.... Next, in the dialogue depicted in Figure 3, browse in your computer's file-system to choose the *part2.s* file and then click Open.

The assembly program should be displayed in the `Editor` pane of the *CPUlator* window, as illustrated in Figure 4. You can make changes to your code in this `Editor` pane if needed by simply selecting text with your mouse and making edits using your keyboard.
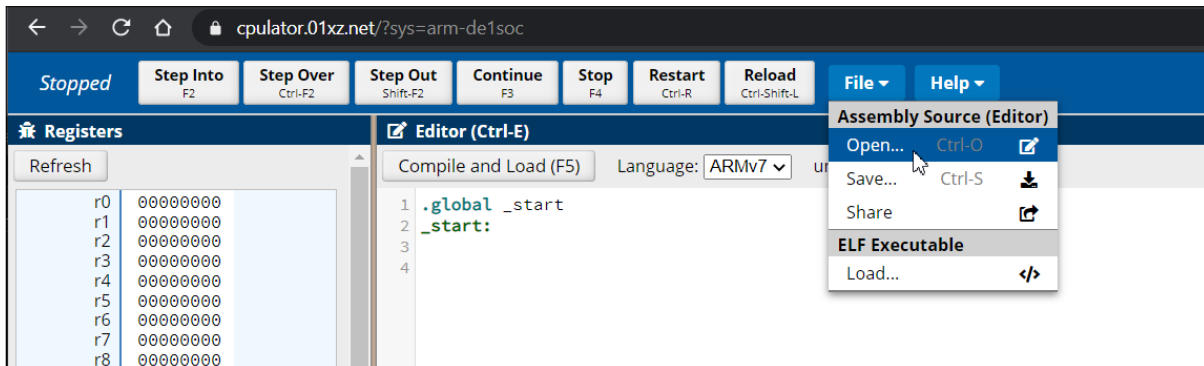

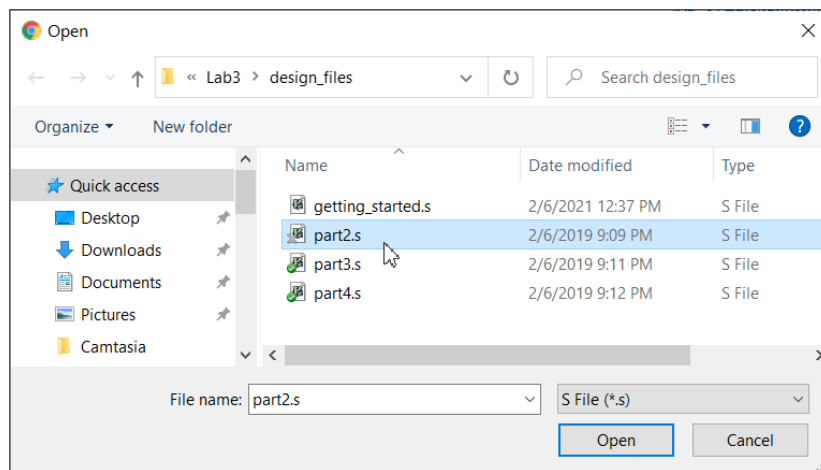Figure 2: The `File` menu in *CPUlator*.
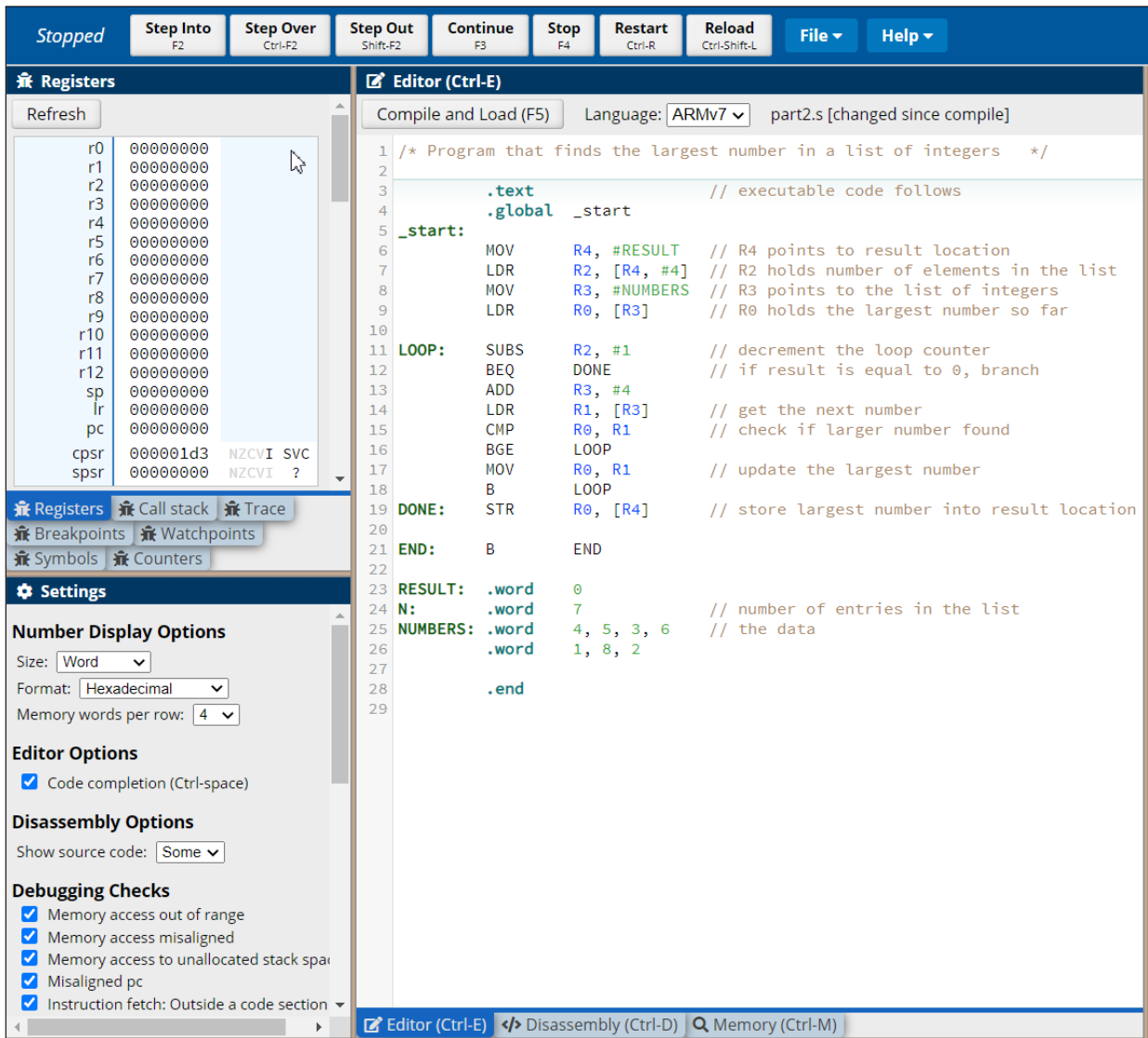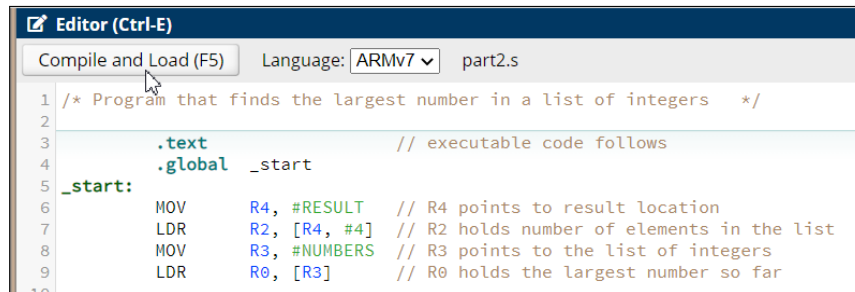

Figure 3: Selecting the *part2.s* file.

Figure 4: The assembly-language program in the *CPUlator* `Editor` pane.

2. As indicated in Figure 5, click on the `Compile and Load` command to *assemble* your program and *load* it into the *memory* that is part of the computer system being simulated within the *CPUlator* tool. You should see the message displayed in Figure 6, in the `Messages` pane, which reports a successful compilation result. If not, then you may have inadvertently introduced an error in the program code; fix any such errors and recompile.

Once the compilation is successful, the *CPUlator* window automatically displays the `Disassembly` pane, shown in Figure 7. This pane lets you see the machine code for the program and gives the address in the *memory* of each machine-code word. The `Disassembly` pane shows each instruction in the program twice: once using the original source code and a second time using the actual instruction found by *disassembling* the machine code. This is done because the *implementation* of an instruction may differ, in some cases, from the *specification* of that instruction in the source code (examples where such differences happen will be shown in class). Note: you can change the way that code is displayed in the CPUlator by

using its `Settings` menu, which is on the left hand side of the CPUlator window (for example, changing `Disassembly Options` from *Some* to *None* will ensure that each instruction is displayed only once).
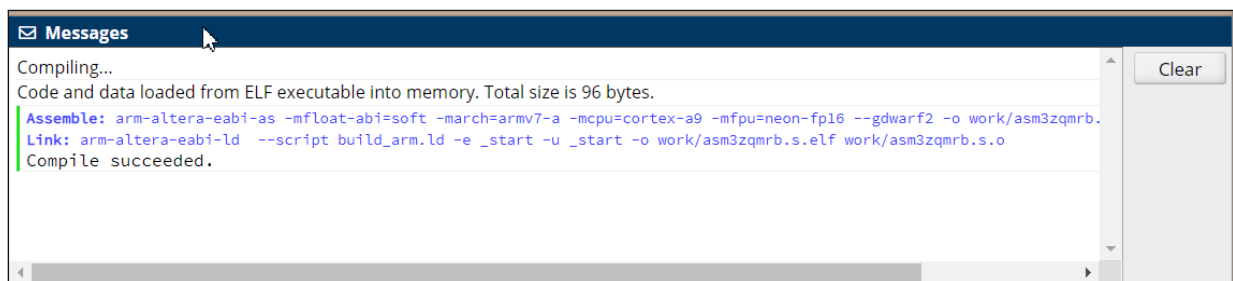

Figure 5: Compiling and loading the program.

Figure 6: The `Messages` pane.

3. Select the `Continue` command near the top of the *CPUlator* window. This command "executes" the program on the ARM processor that is part of the computer system being simulated within the *CPUlator* tool. As illustrated in Figure 7, the program runs to the line of code labeled `END`, at memory address `0x34`, where it remains in an endless loop. Select the `Stop` command to halt the program's execution. Note that the largest number found in the sample list is 8 as indicated by the contents of register `r0`. This result is also stored in memory at the label RESULT.

Figure 7: The result of executing the program.

The address of the label RESULT for this program is `0x00000038`, which can be seen near the bottom of Figure 7. Also, you may notice that the `Disassembly` pane attempts to figure out the machine code at this location, assuming that it represents a processor instruction; it does not, and so the resulting instruction displayed (`andeq`) is not meaningful.

Use the *CPUlator's* `Memory` pane, as illustrated in Figure 8, to verify that the resulting value 8 is stored in the correct location.
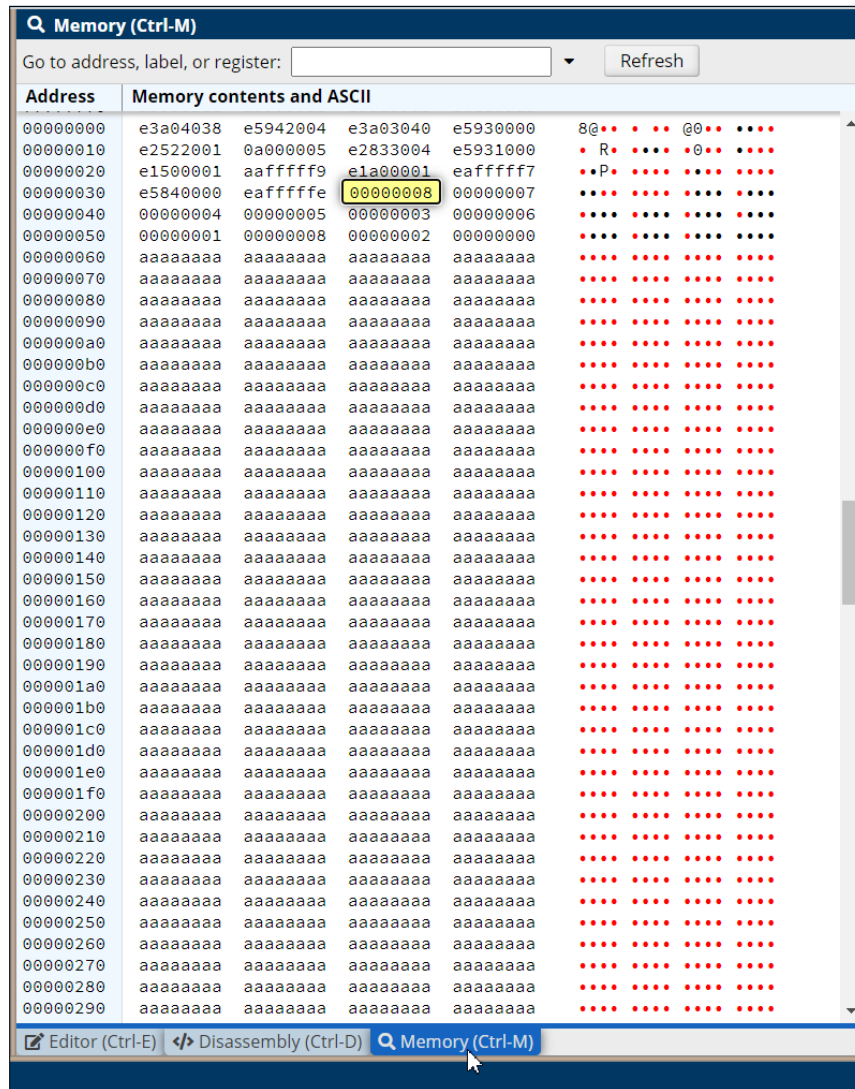
Figure 8: The `Memory` pane.

4. You can return control of the program to the start by clicking on the `Restart` command in *CPUlator*. Do this and then *single-step* through the program by (repeatedly) selecting the `Step Into` command. Observe how each instruction that is executed affects the contents of the ARM processor's registers.

5. Double-click on the `pc` register in the *CPUlator* and then change the value of the program counter to 0. This action has the same effect as selecting the `Restart` command.

6. Now set a breakpoint at address `0x0000002C` by clicking on the gray bar to the left of this address, as illustrated in Figure 9. Select the `Continue` command to run the program again and observe the contents of register `r0` when the instruction at the breakpoint, which is `B LOOP`, is reached. Use `Continue` to repeatedly run the program, which will stop at the breakpoint in each iteration of the loop, and monitor the contents of register `r0` as the program searches for the largest number in the list.
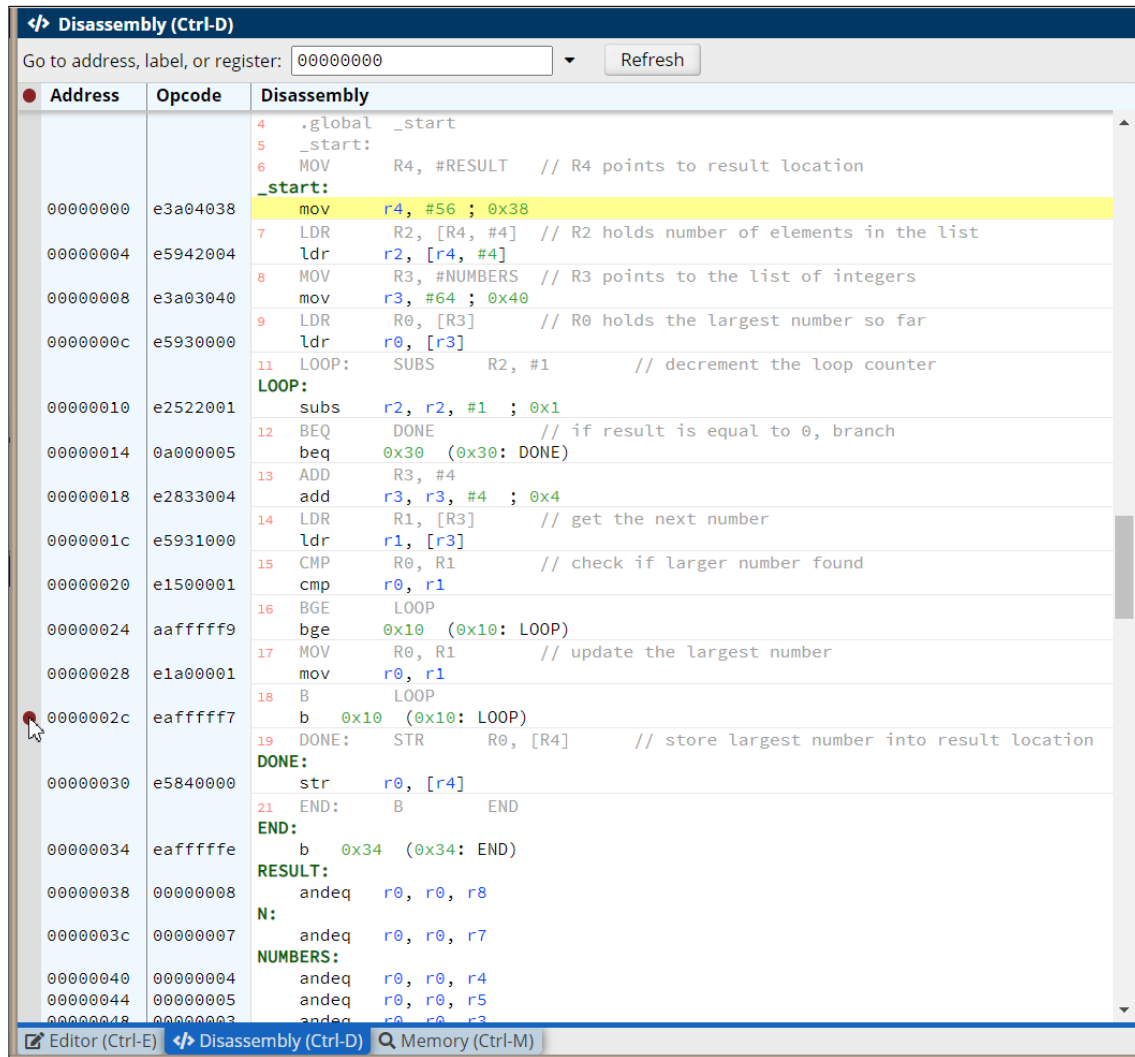
Figure 9: Setting a breakpoint.

Part III

Implement the task in Part II by modifying the program in Figure 1 so that it uses a subroutine. The subroutine, LARGE, has to find the largest number in a list. The main program passes the number of entries and the address of the start of the list as parameters to the subroutine via registers `r0` and `r1`. The subroutine returns the value of the largest number to the calling program via register `r0`. A suitable main program is given in Figure 10.

Use the *CPUlator* tool to assemble, load, execute, and debug (as needed!) your program.

```
/* Program that finds the largest number in a list of integers */

            .text                       // executable code follows
            .global _start
_start:
            MOV     R4, #RESULT     // R4 points to result location
            LDR     R0, [R4, #4]    // R0 holds the number of elements in the list
            MOV     R1, #NUMBERS    // R1 points to the start of the list
            BL      LARGE
            STR     R0, [R4]        // R0 holds the subroutine return value

END:        B       END

/* Subroutine to find the largest integer in a list
 * Parameters: R0 has the number of elements in the list
 *             R1 has the address of the start of the list
 * Returns: R0 returns the largest item in the list */
LARGE:      . . .
            . . .

RESULT:     .word   0
N:          .word   7               // number of entries in the list
NUMBERS:    .word   4, 5, 3, 6  // the data
            .word   1, 8, 2

            .end
```

Figure 10: Main program for Part III.

Part IV

The program shown in Figure 11 converts a binary number into two decimal digits. The binary number is loaded from memory at the location N, and the two decimal digits that are extracted from N are stored into memory in two bytes starting at the location *Digits*. For the value $N = 76$ (0x4c) shown in the figure, the code sets *Digits* to 00000706.

Make sure that you understand how the code in Figure 11 works. Then, extend the code so that it converts the binary number to four decimal digits, supporting decimal values up to 9999. You should modify the DIVIDE subroutine so that it can use any divisor, rather than only a divisor of 10. Pass the divisor to the subroutine in register r1.

If you run your code with the value $N = 9876$ (0x2694), then *Digits* should be set to 09080706. Use the *CPUlator* tool to develop your program, and to demonstrate that it works correctly.

```
/* Program that converts a binary number to decimal */

            .text                   // executable code follows
            .global _start
_start:
            MOV     R4, #N
            MOV     R5, #Digits     // R5 points to the decimal digits storage location
            LDR     R4, [R4]        // R4 holds N
            MOV     R0, R4          // parameter for DIVIDE goes in R0
            BL      DIVIDE
            STRB    R1, [R5, #1]    // Tens digit is now in R1
            STRB    R0, [R5]        // Ones digit is in R0
END:        B       END

/* Subroutine to perform the integer division R0 / 10.
 * Returns: quotient in R1, and remainder in R0 */
DIVIDE:     MOV     R2, #0
CONT:       CMP     R0, #10
            BLT     DIV_END
            SUB     R0, #10
            ADD     R2, #1
            B       CONT
DIV_END:    MOV     R1, R2          // quotient in R1 (remainder in R0)
            MOV     PC, LR

N:          .word   76              // the decimal number to be converted
Digits:     .space  4               // storage space for the decimal digits

            .end
```

Figure 11: A program that converts a binary number into two decimal digits.

Part V: Introduction to the Monitor Program

This part of the exercise introduces the *Monitor Program* application software, which allows you to develop and debug code for the ARM processor that is being executed on a *real* DE1-SoC board. As stated earlier in this exercise, you cannot use the CPUlator to control a real hardware board, because the CPUlator only *simulates* the features of the DE1-SoC Computer and does support the actual hardware.

To use the Monitor Program software, your computer must be connected by a USB cable to a DE1-SoC board (the cable has to be plugged into the *USB Blaster* port on the board). In a similar manner as described earlier in this exercise (when using the CPUlator), you will develop software code that runs on the DE1-SoC Computer, which includes the ARM processor, memory, and various I/O devices. However, in this case you are not using a *simulation* of the DE1-SoC Computer. Instead, the *DE1-SoC Computer* is implemented as a circuit which is downloaded by the Monitor Program tool into the SoC FPGA device on the DE1-SoC board. You use the Monitor Program to control the ARM processor on the board.

The remaining parts of this exercise require the use of a DE1-SoC board. Assuming that you do not have a board at home, you will need to perform these steps during your practical session in the University lab room, using your assigned workstation and a DE1-SoC board.

The first step in using the Monitor Program is to set up an ARM software development project. Perform the following:

1. Make sure that the power is turned on for the DE1-SoC board.

2. Open the *Monitor Program* software, which leads to the window in Figure 12.

   To develop ARM software code using the Monitor Program it is necessary to create a new project. Select File > New Project to reach the window in Figure 13. Give the project a name and indicate the folder for the project; we have chosen the project name *part1* in the folder *Exercise1\Part1*, as indicated in the figure. Use the drop-down menu shown in Figure 13 to set the target architecture to the ARM Cortex-A9 processor. Click Next, to get the window in Figure 14.

3. Now, you can select your own custom computer system (if you have one) or a pre-designed (by Intel) system. As shown in Figure 14 select the *DE1-SoC Computer*. Once you have selected the computer system the display in the window will now show where files that implement the chosen system are located. Click Next.
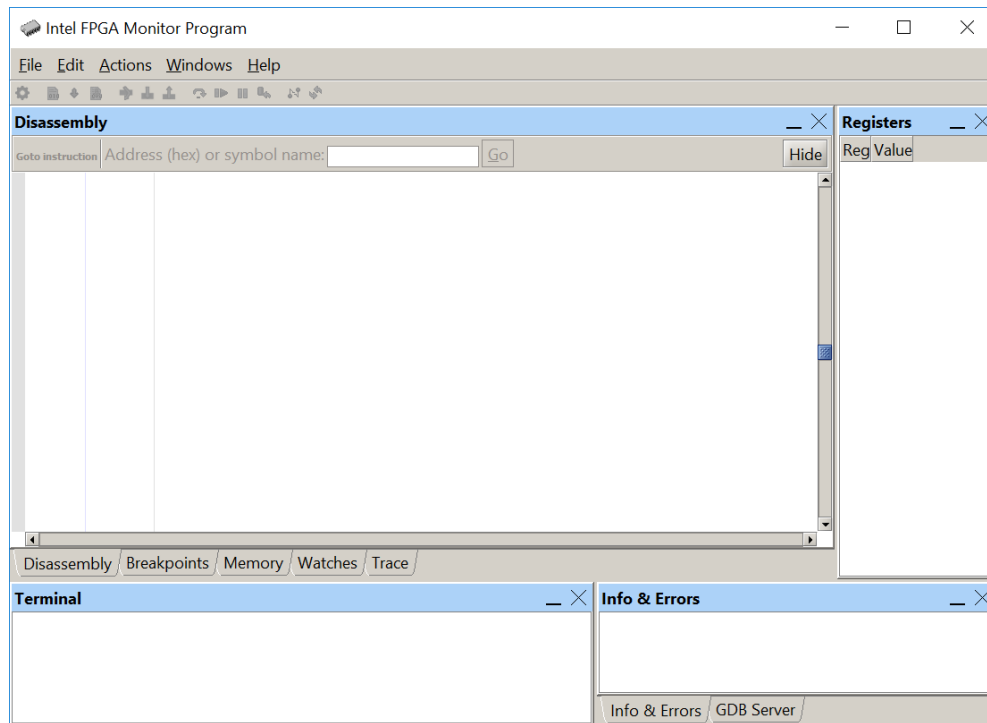
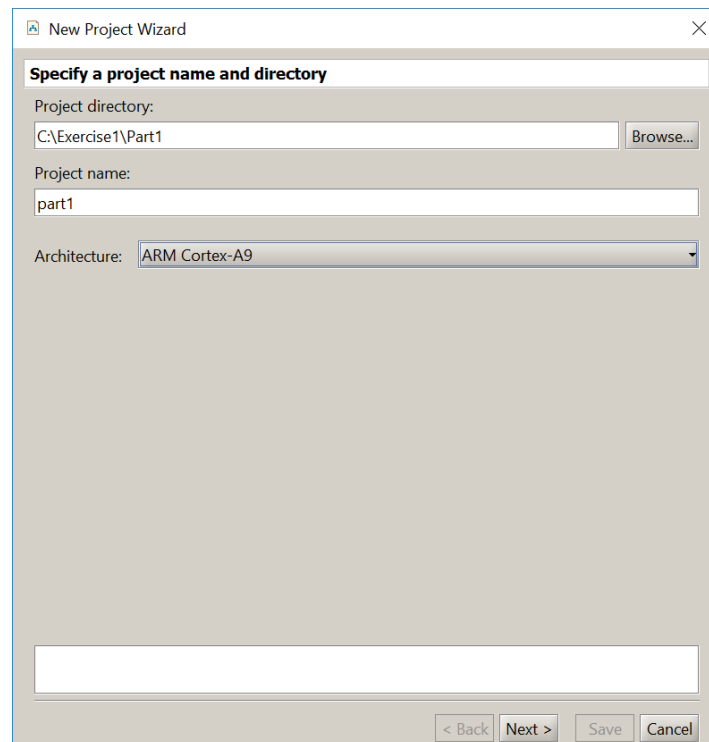Figure 12: The *Monitor Program* window.

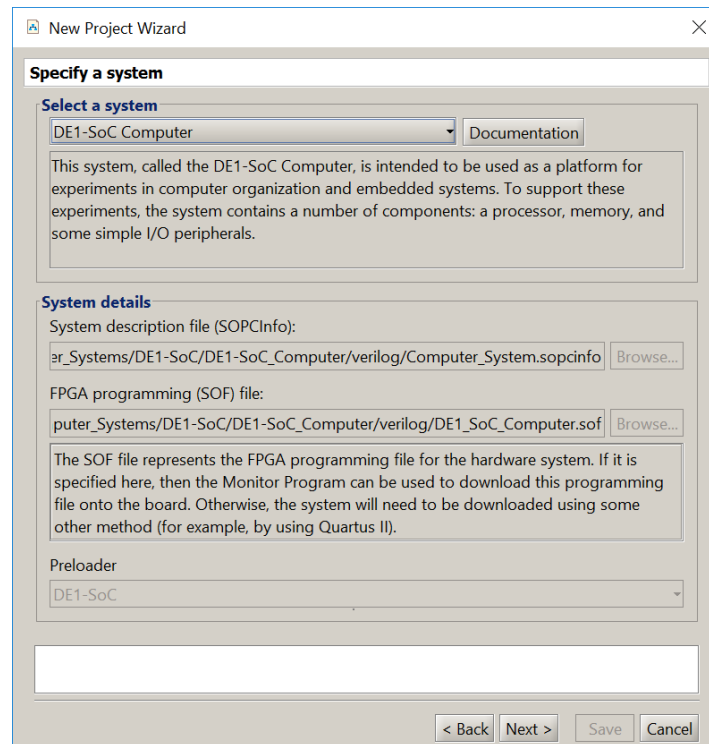Figure 13: Specify the folder and the name of the project.

Figure 14: Specification of the system.

4. In the window in Figure 15 you can specify the type of application programs that you wish to run. They can be written in either assembly language or the C programming language. Specify that an assembly language program will be used. The *Monitor Program* package contains several sample programs. Select the box Include a sample program with the project. Then, choose the Getting Started program, as indicated in the figure, and click Next.

5. The window in Figure 16 is used to specify the source file(s) that contain the application program(s). Since we have selected the *Getting Started* program, the window indicates the source code file for this program. This window also allows the user to specify the starting point in the selected application program. The default symbol is *_start*, which is used in the selected sample program. Click Next.

6. The window in Figure 17 indicates some system parameters. Note that the figure indicates that the *DE-SoC [USB-1]* cable is selected to provide the connection between the DE-series board and the host computer. This is the name assigned to the Intel USB Blaster connection between the computer and the board. Click Next.
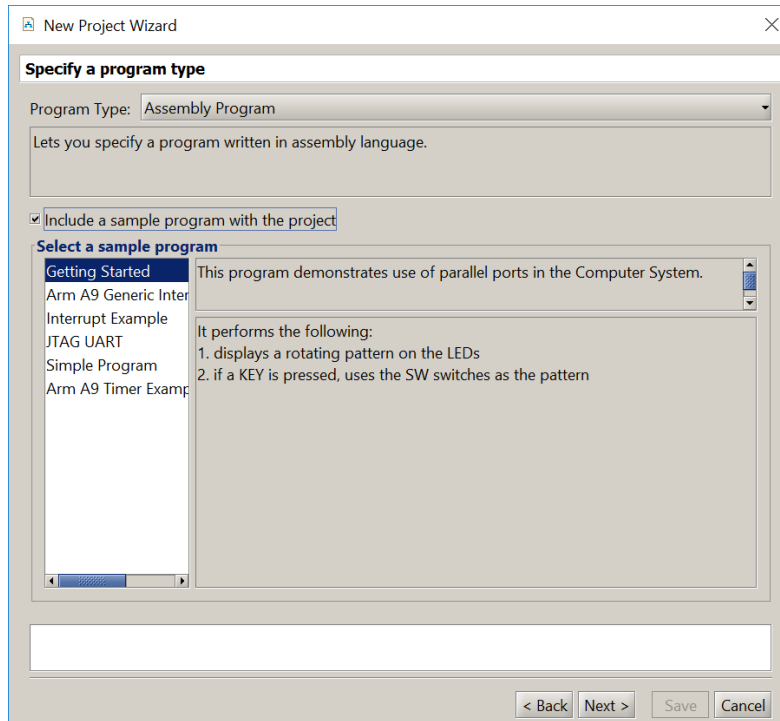
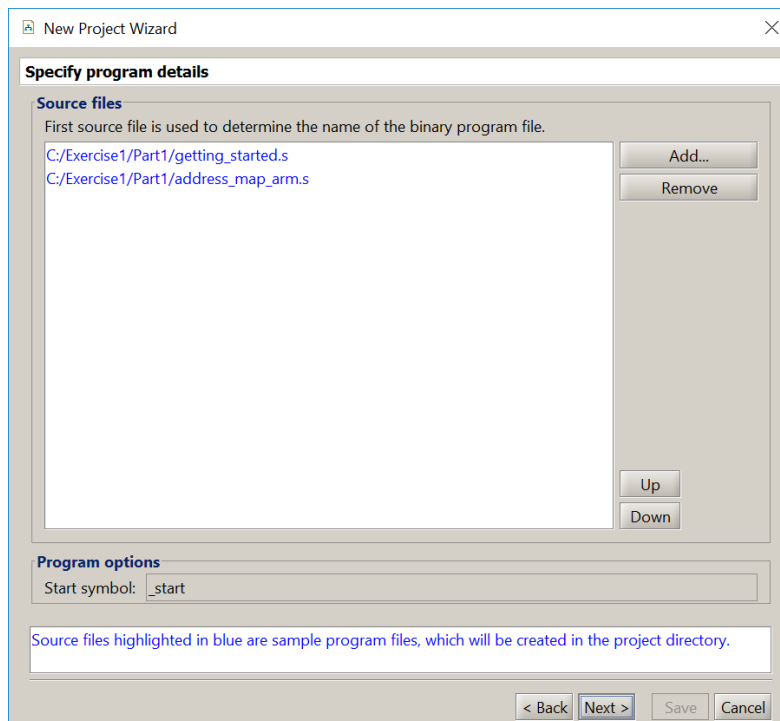Figure 15: Selection of an application program.

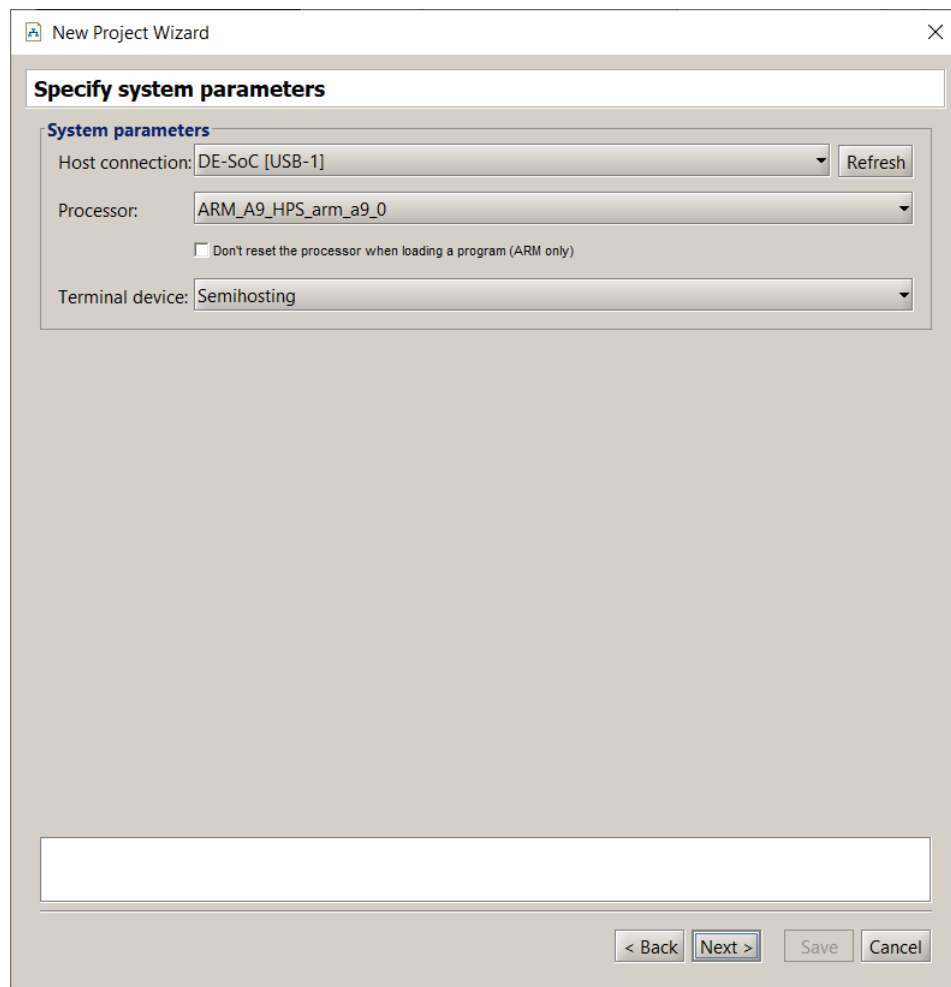Figure 16: Source files used by the application program.

Figure 17: Specify the system parameters.

7. The window in Figure 18 displays the names of Assembly sections that will be used for the program, and allows the user to select a target memory location for each section. In this case only the *.text* section, which corresponds to the program code (and data), will be used. As shown in the figure, the *.text* section is targeted to the DDR3 memory in the DE-series board, starting at address 0. Click Save to complete the specification of the new project.

8. Since you specified a new project, a pop-up box will appear asking you if you want to download the system associated with this project onto the DE-series board. Make sure that the power to the board is turned on and click Yes. After the download is complete, a pop-up box will appear informing you that the circuit has been successfully downloaded—click OK. [1] If the circuit is not successfully downloaded, make sure that the USB connection, through which the USB-Blaster communicates, is established and recognized by the host computer. (If there is a problem, a possible remedy may be to unplug the USB cable and then plug it back in.)

---

[1]The first time that you run the Monitor Program, you may need to respond to a pop-up dialog from MS Windows to allow access to the program (the message may be related to one/both of Java and Quartus).
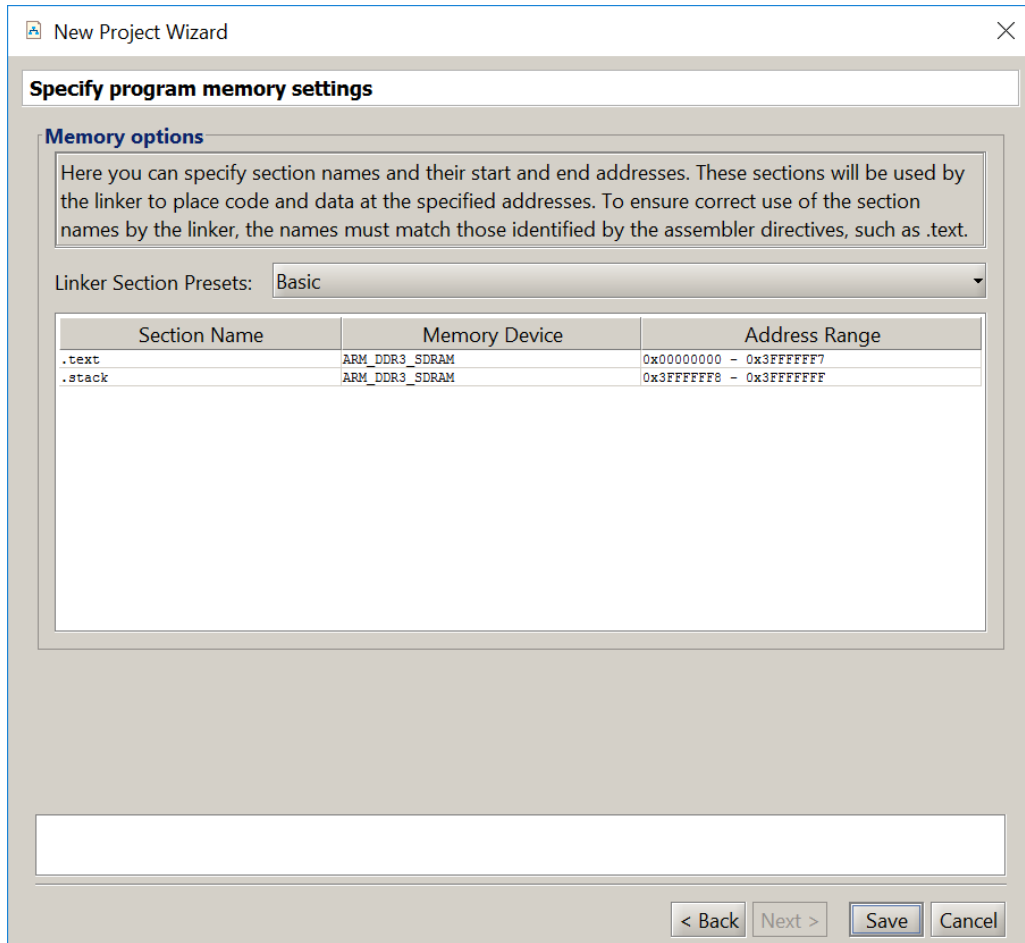
Figure 18: Specify the program memory settings.

9. Having downloaded the computer system into the FPGA SoC chip on your DE1-SoC board, we can now load and run the sample program. In the main Monitor Program window, shown in Figure 19, select Actions > Compile & Load to assemble the program and load it into the memory on the board. Figure 19 shows the Monitor Program window after the sample program has been loaded.

10. Run the program by selecting Actions > Continue or by clicking on the toolbar icon ▯▷, and observe the patterns displayed on the LEDs.

11. Pause the execution of the sample program by clicking on the icon ▯▯, and disconnect from this session by clicking on the icon ⚡,
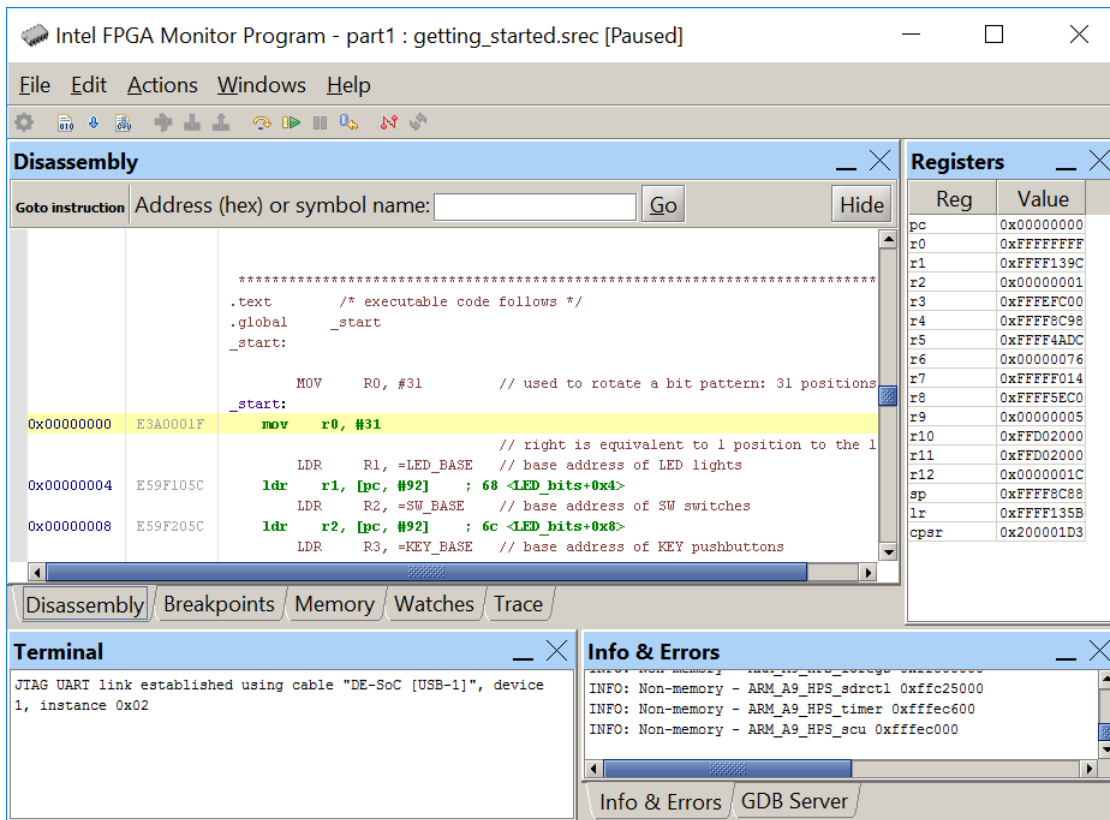
Figure 19: The monitor window showing the loaded sample program.

Part VI

Now, we will explore some features of the Monitor Program by using the ARM assembly-language code from Figure 1, which finds the largest number in a list of 32-bit integers that is stored in the memory. For convenience, the code is reproduced in Figure 20.

```
/* Program that finds the largest number in a list of integers   */

        .text                   // executable code follows
        .global _start
_start:
        MOV     R4, #RESULT     // R4 points to result location
        LDR     R2, [R4, #4]    // R2 holds number of elements in the list
        MOV     R3, #NUMBERS    // R3 points to the list of integers
        LDR     R0, [R3]        // R0 holds the largest number so far

LOOP:   SUBS    R2, #1          // decrement the loop counter
        BEQ     DONE            // if result is equal to 0, branch
        ADD     R3, #4
        LDR     R1, [R3]        // get the next number
        CMP     R0, R1          // check if larger number found
        BGE     LOOP
        MOV     R0, R1          // update the largest number
        B       LOOP
DONE:   STR     R0, [R4]        // store largest number into result location

END:    B       END

RESULT: .word   0
N:      .word   7               // number of entries in the list
NUMBERS: .word  4, 5, 3, 6      // the data
        .word   1, 8, 2

        .end
```

Figure 20: Assembly-language program that finds the largest number.

Perform the following:

1. Create a new folder for this part of the exercise, with a name such as *Part2*. Create a file named *part2.s* and enter the code from Figure 1 into this file. Use the Monitor Program to create a new project in this folder; we have chosen the project name *part2*. When you reach the window in Figure 15 choose Assembly Program but do not select a sample program. Click Next.

2. Upon reaching the window in Figure 16, you have to specify the source code file for your program. Click Add and in the pop-up box that appears indicate the desired file name, *part2.s*. Click Next to get to the window in Figure 17. Again click Next to get to the window in Figure 18. Notice that the DDR3_SDRAM is selected as the memory device. Your program will be loaded starting at address 0 in this memory. Click Save. If you are prompted to download the DE1-SoC computer to your board, you can decline, as you already programmed the FPGA chip when performing Part V.

3. Compile and load the program. The Monitor Program will display a disassembled view of the machine code loaded in the memory, as indicated in Figure 21.

4. Execute the program (▶). When the code is running, you will not be able to see any changes (such as the contents of registers or memory locations) in the display for the Monitor Program, because it does not

communicate with the ARM processor while code is being executed. But, if you pause the program then the Monitor Program window will be updated. Pause the program using the icon 〚Ⅱ〛 and observe that the processor stops within the endless loop **END: B END**. Note that the largest number found in the sample list is 8 as indicated by the contents of register R0. This result is also stored in memory at the label RESULT. The address of the label RESULT for this program is 0x00000038. Use the Monitor Program's Memory tab, as illustrated in Figure 22, to verify that the resulting value 8 is stored in the correct location.

5. You can return control of the program to the start by clicking on the icon 〚⤵〛, or by selecting Actions > Restart. Do this and then single-step through the program by clicking on the icon 〚⤷〛. Watch how the instructions change the data in the processor's registers.

6. Double-click on the pc register in the Monitor Program and then set the program counter to 0. This action has essentially the same effect as clicking on the restart icon 〚⤵〛.

7. Now set a breakpoint at address 0x0000002C by clicking on the gray bar to the left of this address, as illustrated in Figure 23. Restart the program and run it again. Observe the contents of register R0 each time the instruction at the breakpoint, which is B LOOP, is reached.
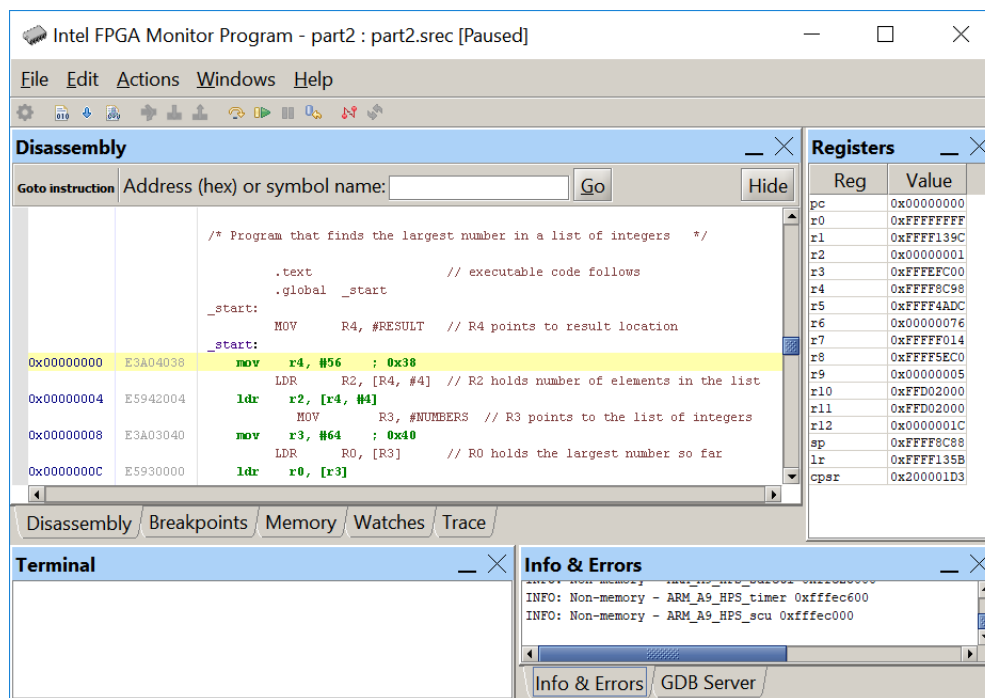

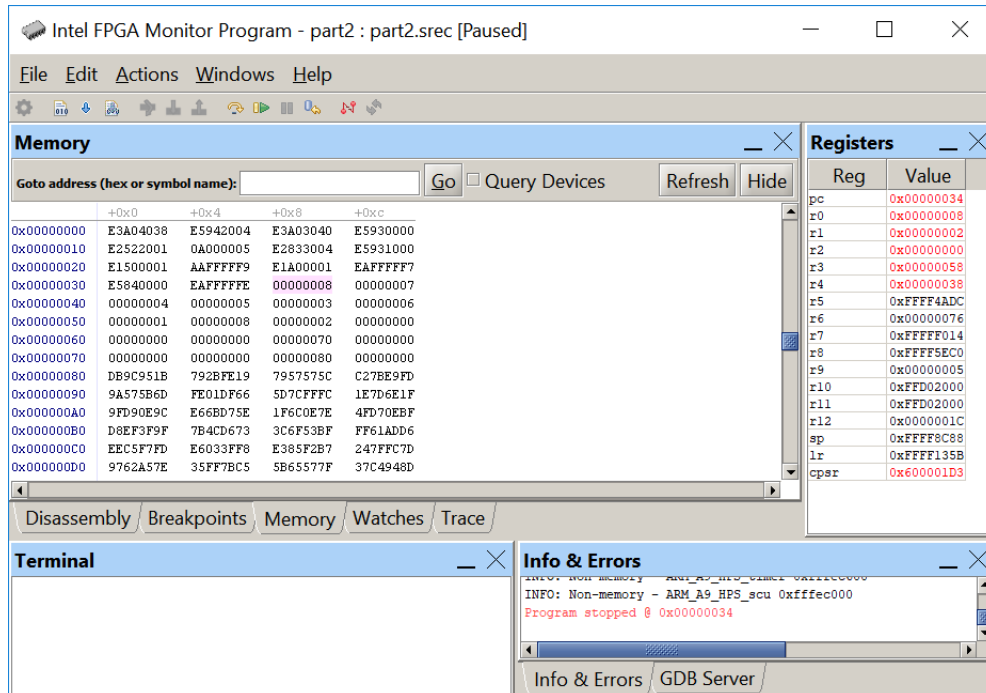Figure 21: The disassembled view of the program in Figure 1.

Figure 22: Displaying the result in the memory tab.
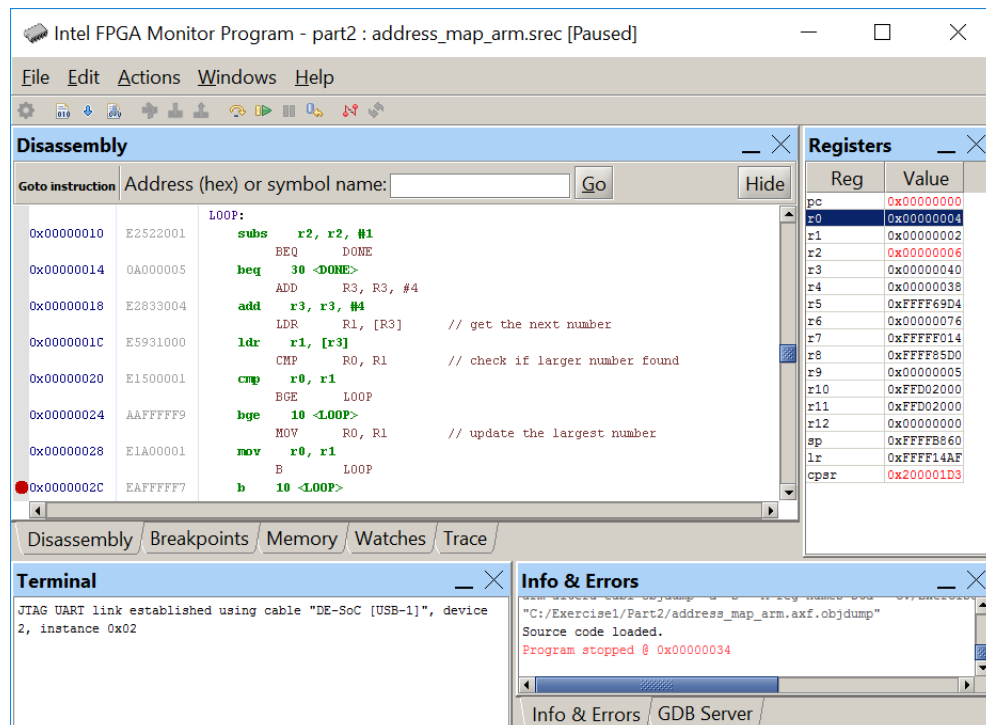

Figure 23: Setting a breakpoint.

Part VII

In this part, you are to run the ARM program that you wrote for Part III of this exercise on a DE1-SoC board, using the Monitor Program. Recall that the program from Part III modifies the code from Figure 20 so that it uses a subroutine, called LARGE, that finds the largest number in a list.

Create a new folder and a new Monitor Program project to compile and download your program. Run your program to verify its correctness.

Part VIII

In this part, you are to run the ARM program that you wrote for Part IV on the DE1-SoC board, using the Monitor Program. Recall that the program from Part IV converts a binary number stored in the memory into four decimal digits, supporting decimal values up to 9999.

Create a new folder and a new Monitor Program project to compile and download your program. Run your program to verify its correctness.

Final Comments

We have provided only a brief introduction to the features of the Monitor Program application software. A more detailed discussion is available in the tutorial *Intel FPGA Monitor Program Tutorial for ARM*, which is provided along with this exercise, in the folder named ARM_DE1_Docs. This tutorial can also be accessed by selecting Help > Tutorial within the Monitor Program.