Lab 1: Introduction to AMR Programming

ECSE 324 – Computer Organization

Fall 2019

Introduction

In this lab you will learn how to work with an ARM processor and the basics of AMR assembly by programming some common routines. After you complete the task, you should demonstrate your work to a TA.

1. Working with the DE-SOC Computer System

For this course, we will be working with the DE1-SoC Computer System, which is composed of an AEM Cortex-A9 processor and peripheral components located on the FPGA on your DE1-SoC board. The IDE we will be using is the Intel FPGA Monitor Program 16.1. In this part of the lab, you will learn how to program the Computer System in ARM assembly.

1.1. Learn about the tools

Before you move on, you should read the Introduction to the ARM Processor Using Altera Toolchain and acquaint yourself with the Altera Monitor Program Tutorial for AMR. You will also find it useful to refer to the ARM Architecture Reference Manual. There documents can be found on myCourses. It may help to keep these manuals open at your work.

1.2. Your first assembly program

1. Open the “Intel FPGA Monitor Program 16.1” from the desktop icon and select File -> New.

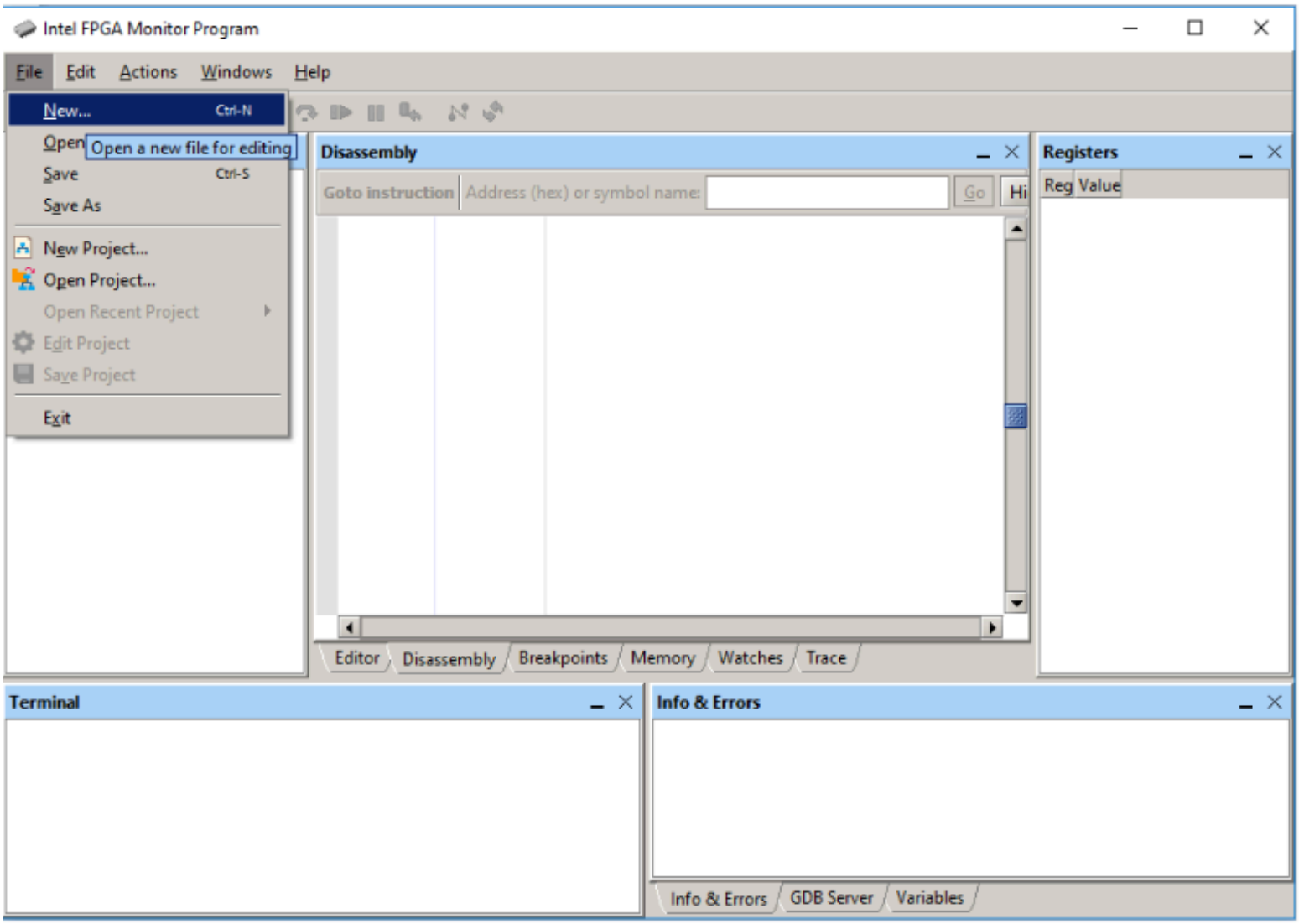
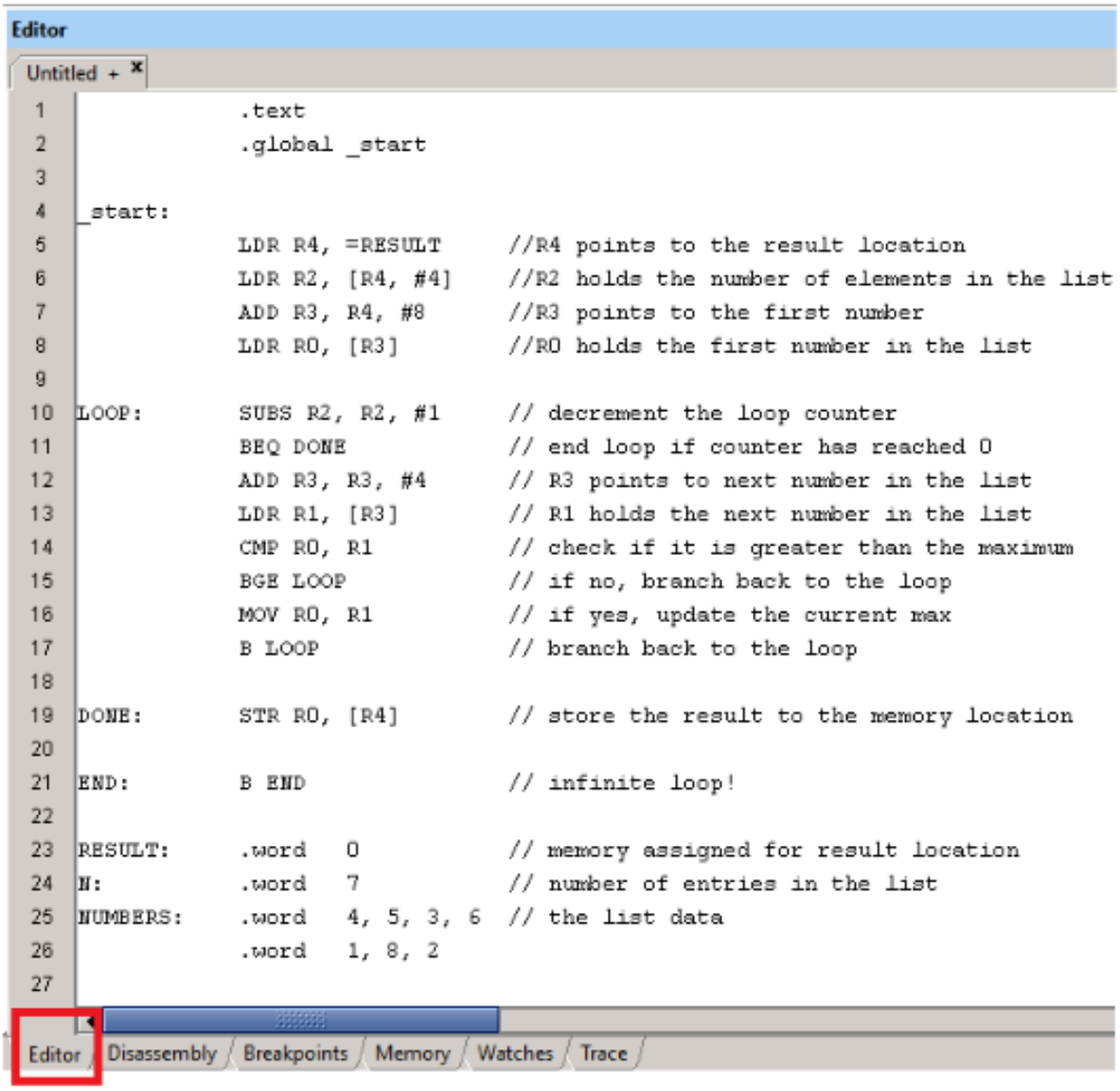


Figure 1: Your first assembly program - Step 1

2. In the new editor window, type out the code as shown in Fig. 2 and save this file as “part1.s” within a new folder “GXX_Lab1” on your network drive. Here, GVV stands for your group number, e.g. Group 1 would be G01_Lab1.

   The code is a simple program to find the maximum number from a list of “Numbers” with length “N”. Notice the extensive use of comments. This practice should become your permanent programming practice.

   Note: The indentation is important. The code will not compile if not intended as shown.



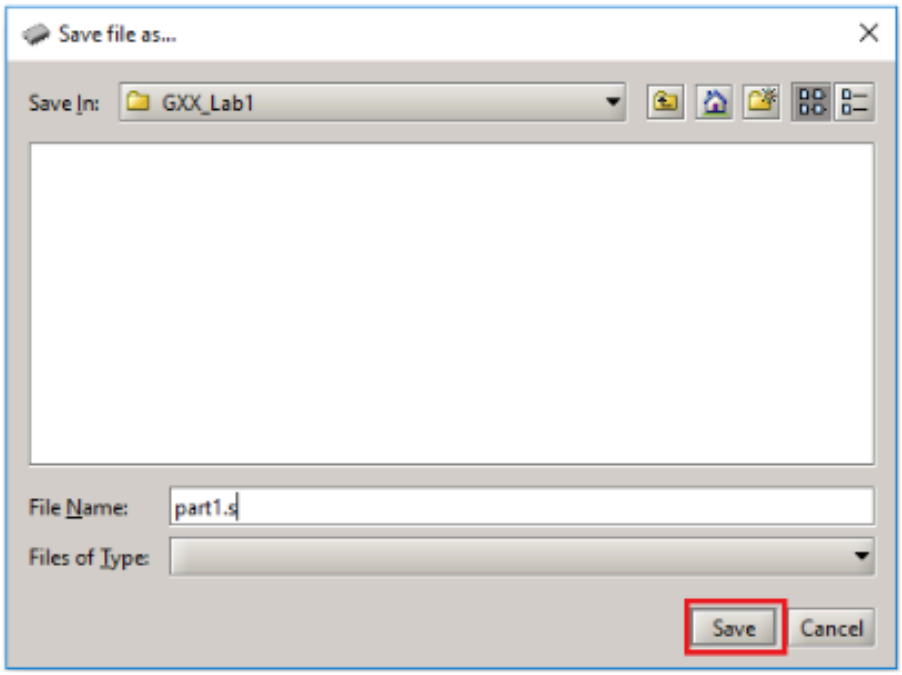


Figure 2: Your first assembly program - Step 2

3. Open the “Intel FPGA Monitor Program 16.1” from the desktop icon and select File –> New Project.

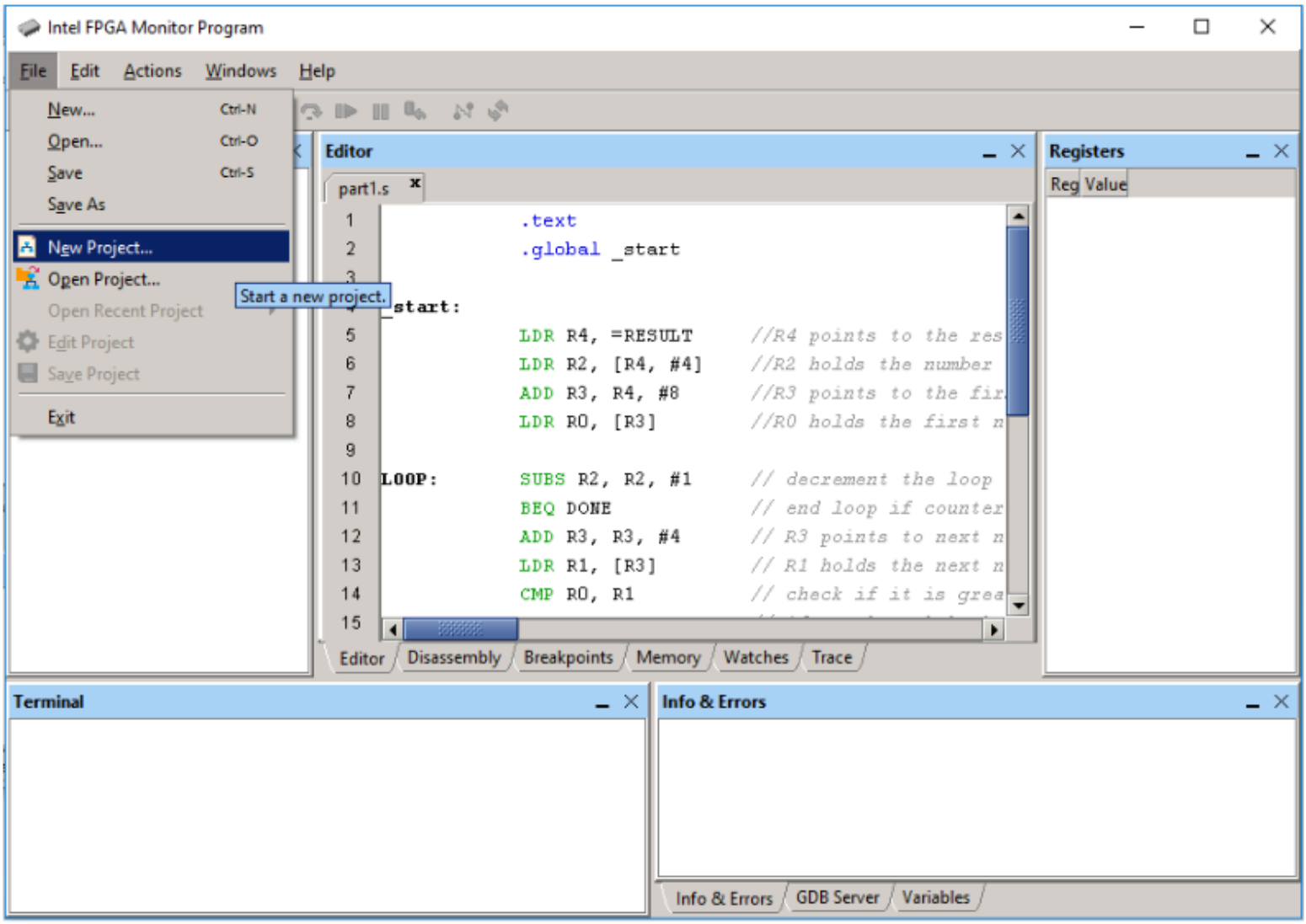


Figure 3: Your first assembly program - Step 3

4. Set the project directory to GXX_Lab1 and set the project name to GXX_Lab1. Select the "ARM Cortex_A9" processor architecture, and click "Next".

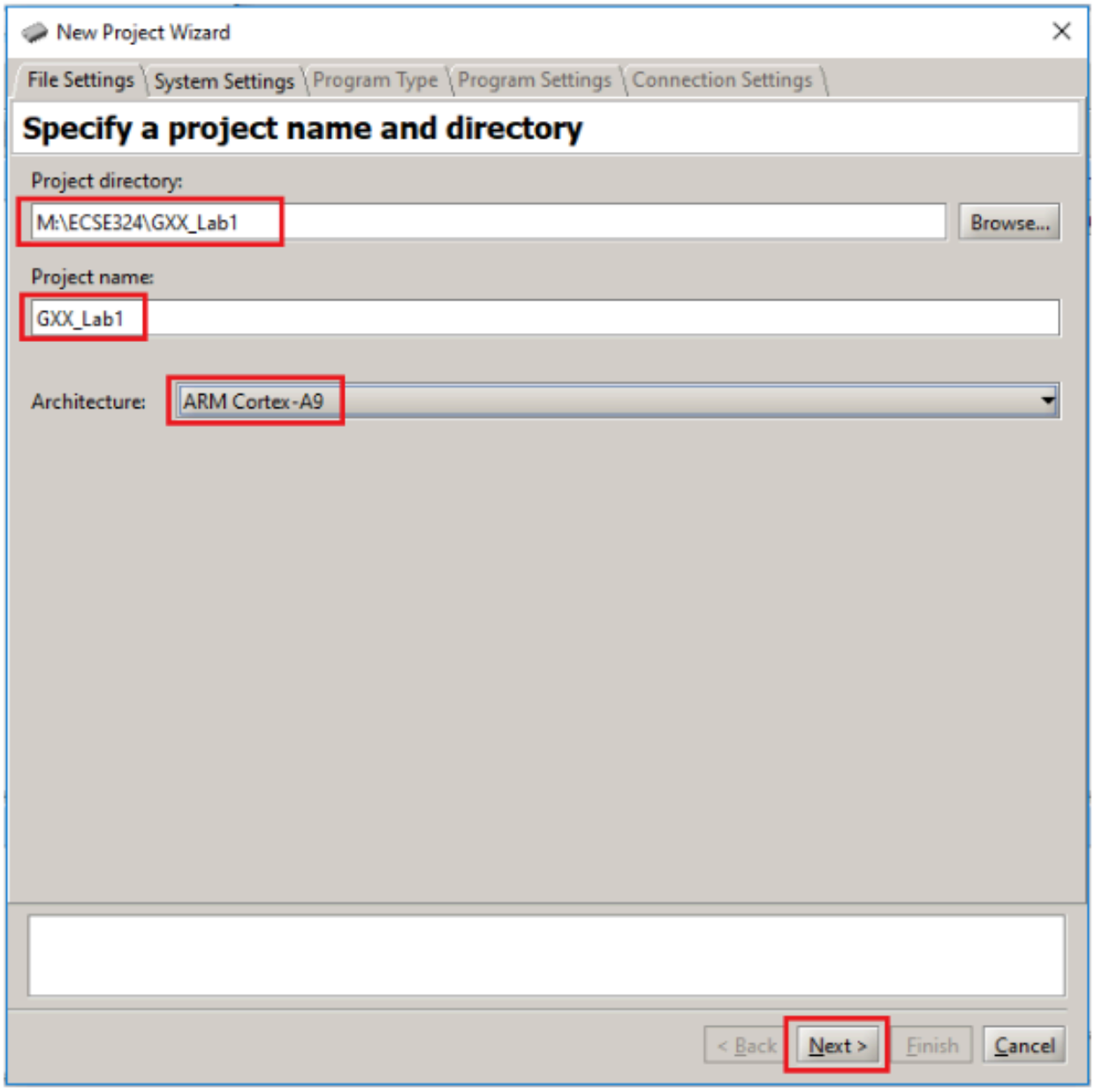


Figure 4: Your first assembly program - Step 4

5. In the next window, under “Select a system” select the DE1-SoC Computer” and click “Next”.

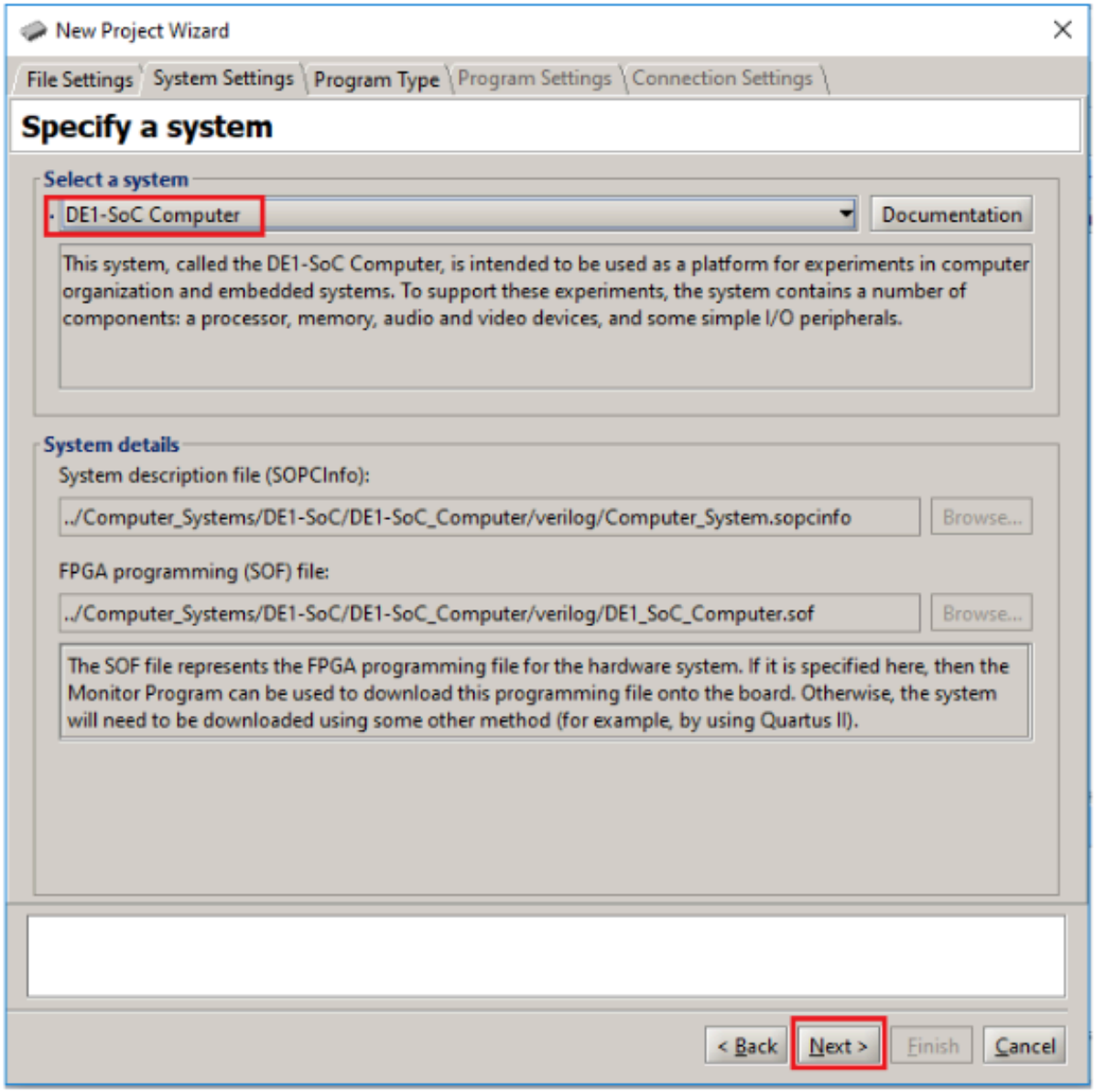


Figure 5: Your first assembly program - Step 5

6. In the next window, under “Program type” select the “Assembly Program” and click “Next”.

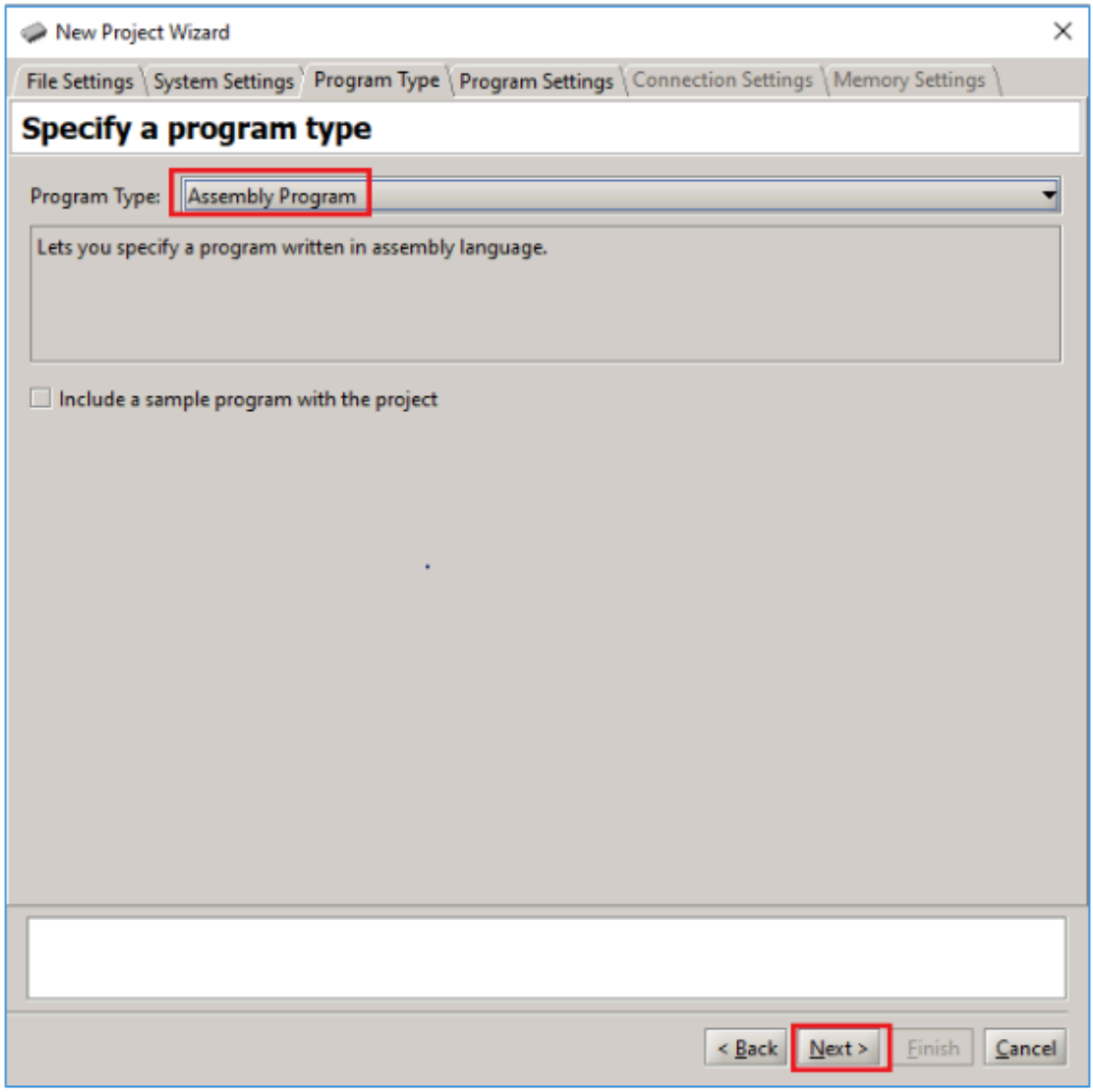


Figure 6: Your first assembly program - Step 6

7. In the next window “Specify program details”, click on “Add …” and select the file “part1.s” created in step 1, and click “Next”.

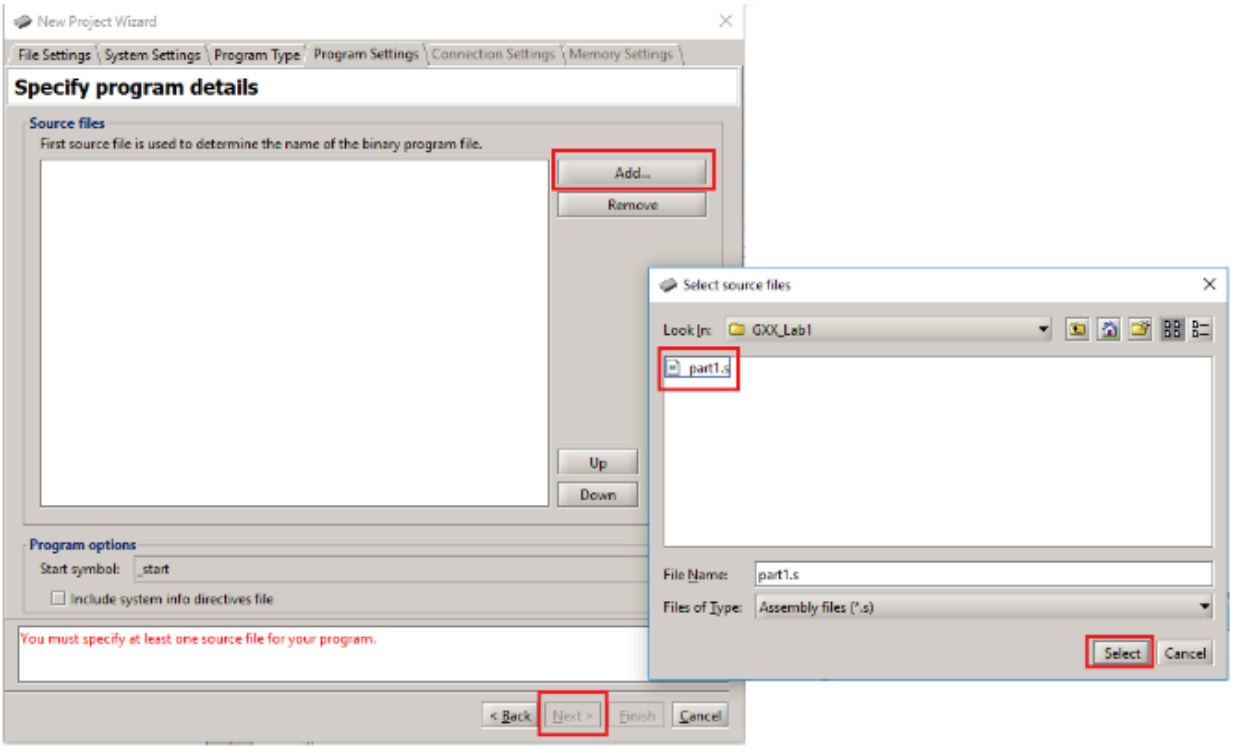


Figure 7: Your first assembly program - Step 7

8. In the next window “Specify system parameters”, ensure that the board is detected in the “Host connection” box, and click “Next”.

Note, that the board has to be plugged in via USB and powered on to be detected.

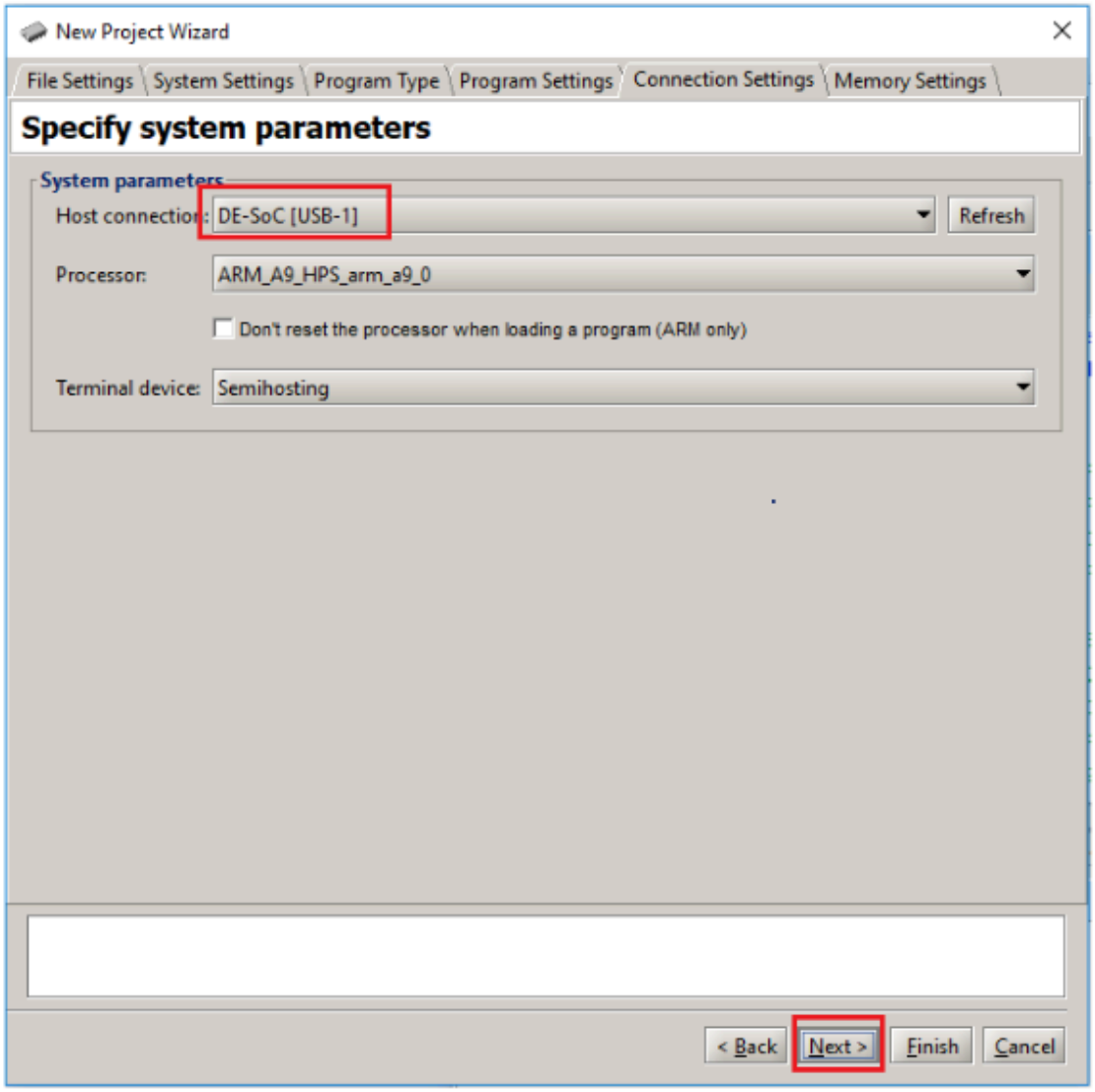


Figure 8: Your first assembly program - Step 8

9. In the next window “Specify program memory settings”, simply click “Finish”.

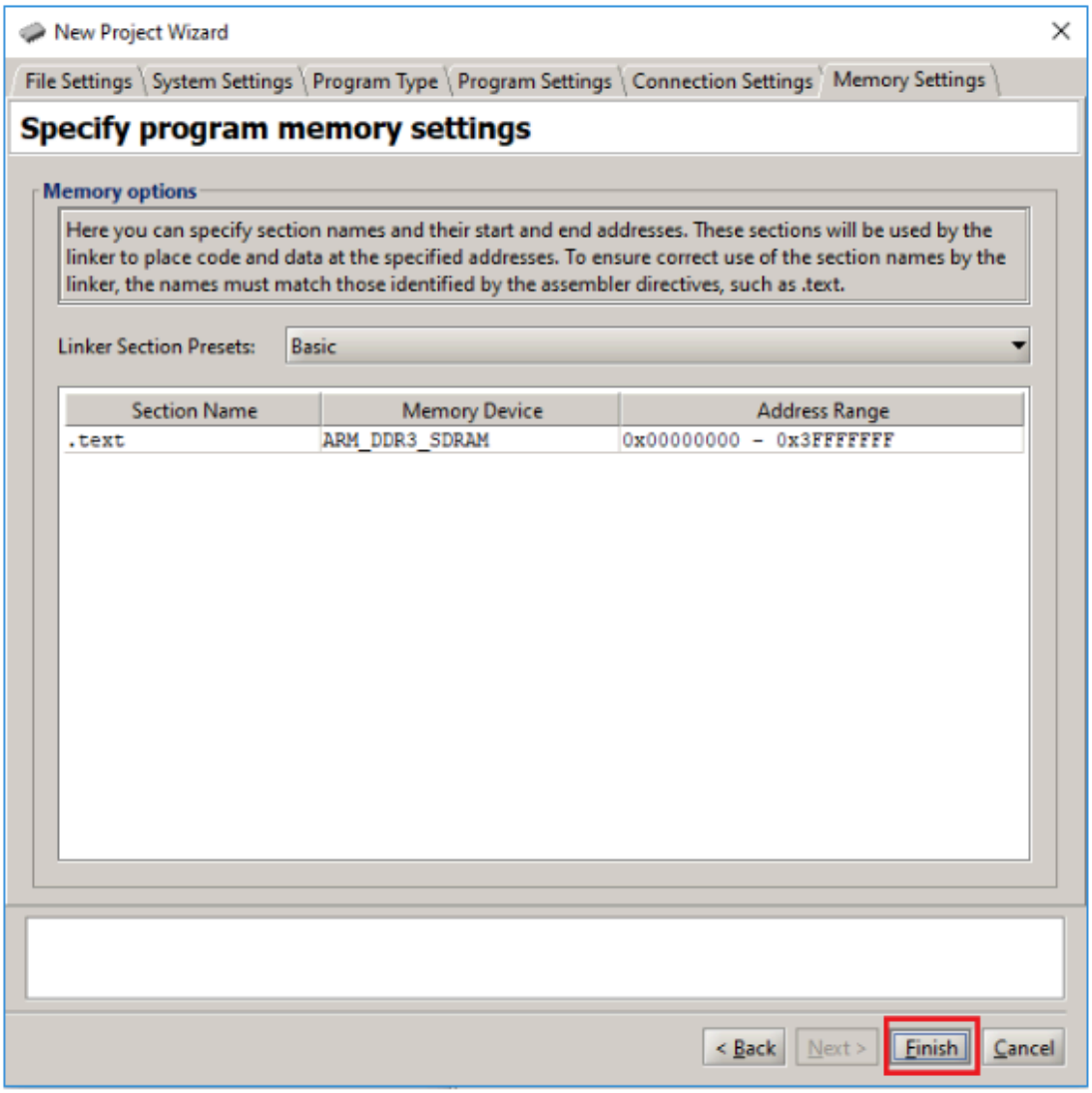


Figure 9: Your first assembly program - Step 9

10. A dialogue box should now pop up, asking whether you would like to download the system onto the board. If you were successfully able to flash your JIC file in Lab0, click “No”, otherwise click “Yes”.

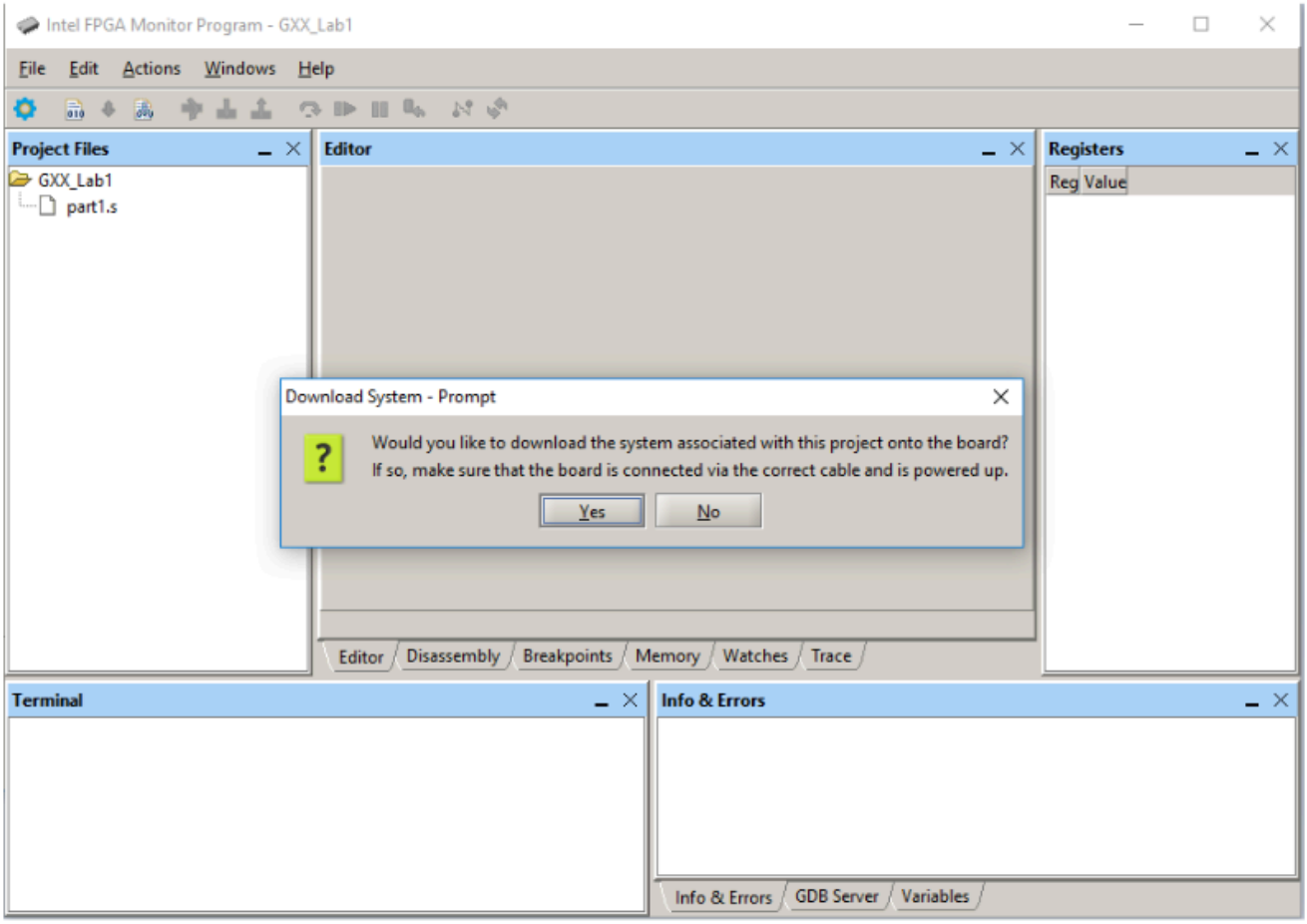


Figure 10: Your first assembly program - Step 10

1.3 Using the IDE

Now that we have created our fist assembly project, let's take a look at some of the features of the IDE and use them in order to debug this program and verify that it works as desired.

Note: This section only provides a very brief introduction to the IDE. More detailed information can and should be obtained in the documentation and by experience.

1. Fig. 11 shows the useful features of the IDE when a project is opened. We can say that we are now in "development mode" – where the code is not loaded onto the board and we are in process of writing code and compiling it to check for errors.

   The green box highlights the different IDE window tabs, and since we are in development mode, the only useful window is the "Editor" window where code can be created/modified. You can add/remove windows using the "Windows" menu at the top.

   Note: "compiling" refers to converting higher level computer code such as C into assembly instructions. What we are doing here is "assembling", which refers to the conversion of assembly instructions into machine code. However, since Altera has decided to call it the "Compile" button, we will stick with that name for the sake of clarity.

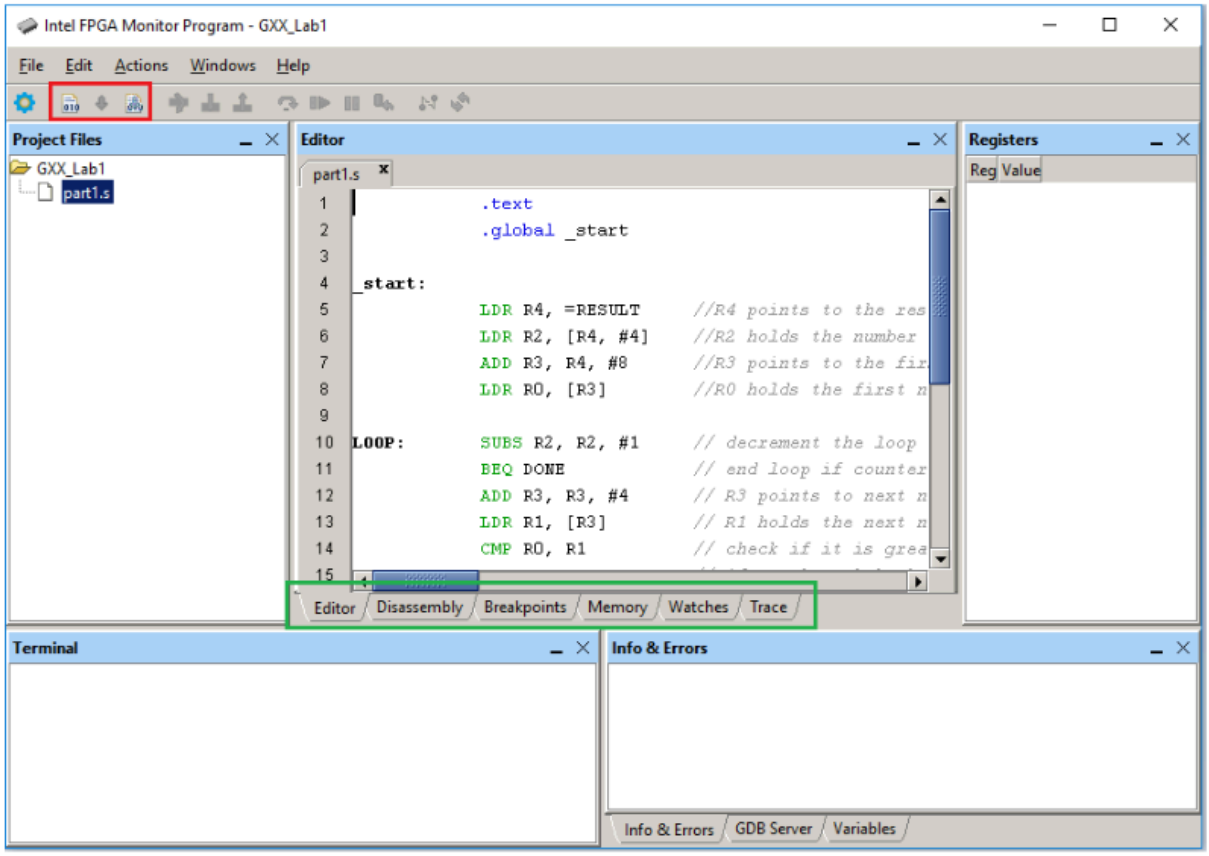


Figure 11: Using the IDE - Development mode

2. When the code is loaded onto the board (by clicking either “Load” or “Compile & Load”), we can say that we are now in ‘debug mode”. The IDE is now connected to the board via a debug server, and we can send execution instructions to the board and receive data (such as register and memory values) back from the board.

   The green box highlights the two important windows in this mode. In the Disassembly window, we can see the code that is being executed, as well as the current instruction when the code is paused. We also have the ability to set/remove breakpoints by clicking on the grey area to the left of the instruction. The Disassembly window is the most important window in debug mode. In the Memory window, we can see the content of a desired memory location, but only when the program is paused!

   The red box highlights the useful buttons in debug mode. Using them, we can “Continue”, “Pause” and “Restart” the program execution. We can also step by a single instruction, or step over multiple instructions. Finally, we can also disconnect from the board.

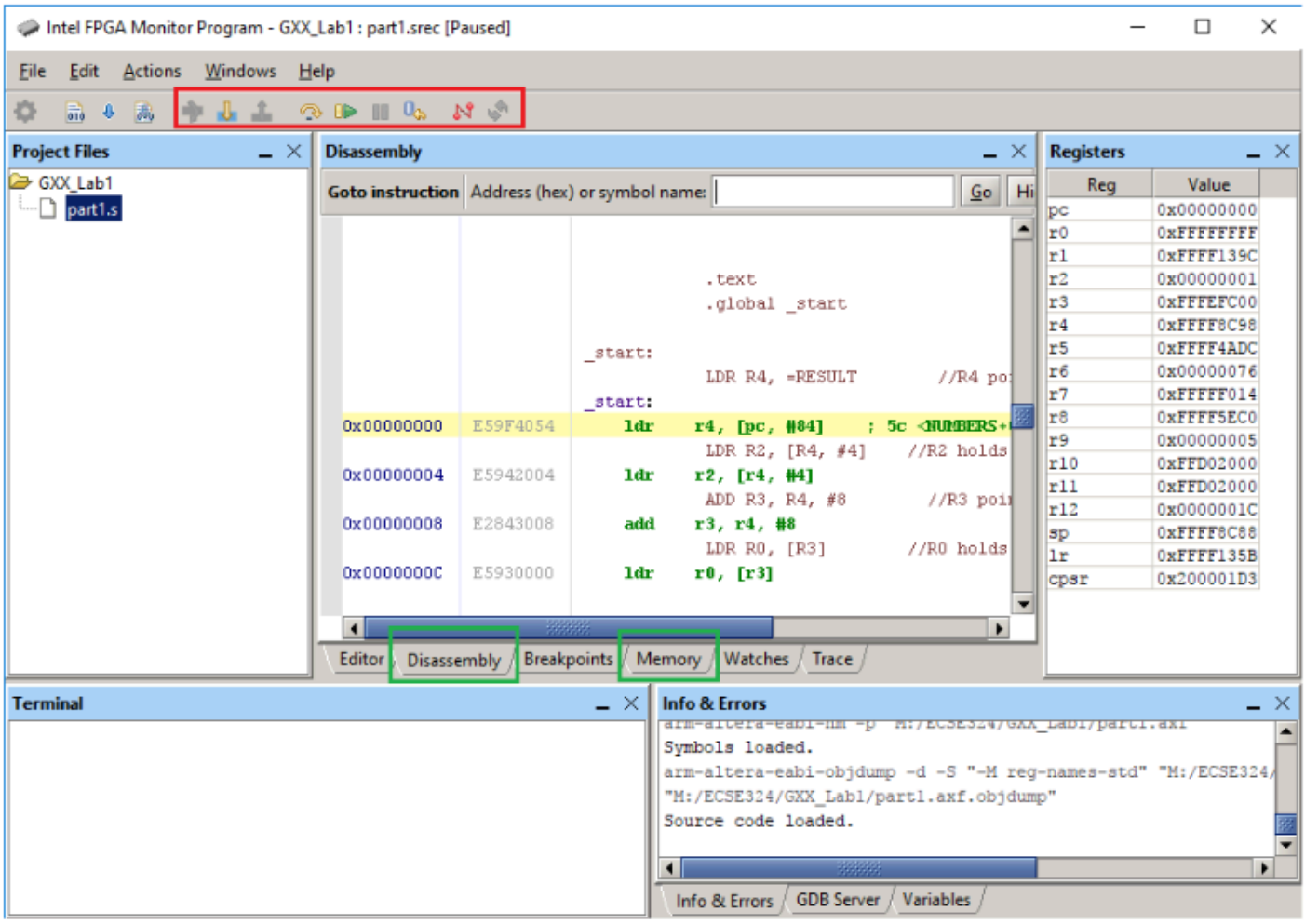


Figure 12: Using the IDE - Debug mode

3. Now let's run the code and verify the result. *Before you do this, make sure you have read the code and understand how it works, otherwise you won't know what it is that you are checking!*

   Ensure that we are in debug mode and looking at the Disassembly window. Click on the "Continue" button, and then click on the "Pause" button. The code should stop at the **B END** instruction. Notice how the contents of the registers have now changed, and R0 contains the expected value!

   Experiment with the IDE features by restarting the program from the first instruction and arriving at the end via steps and breakpoints.

   Finally, note the address **0x00000038** of RESULT, as it will be used in the next part.

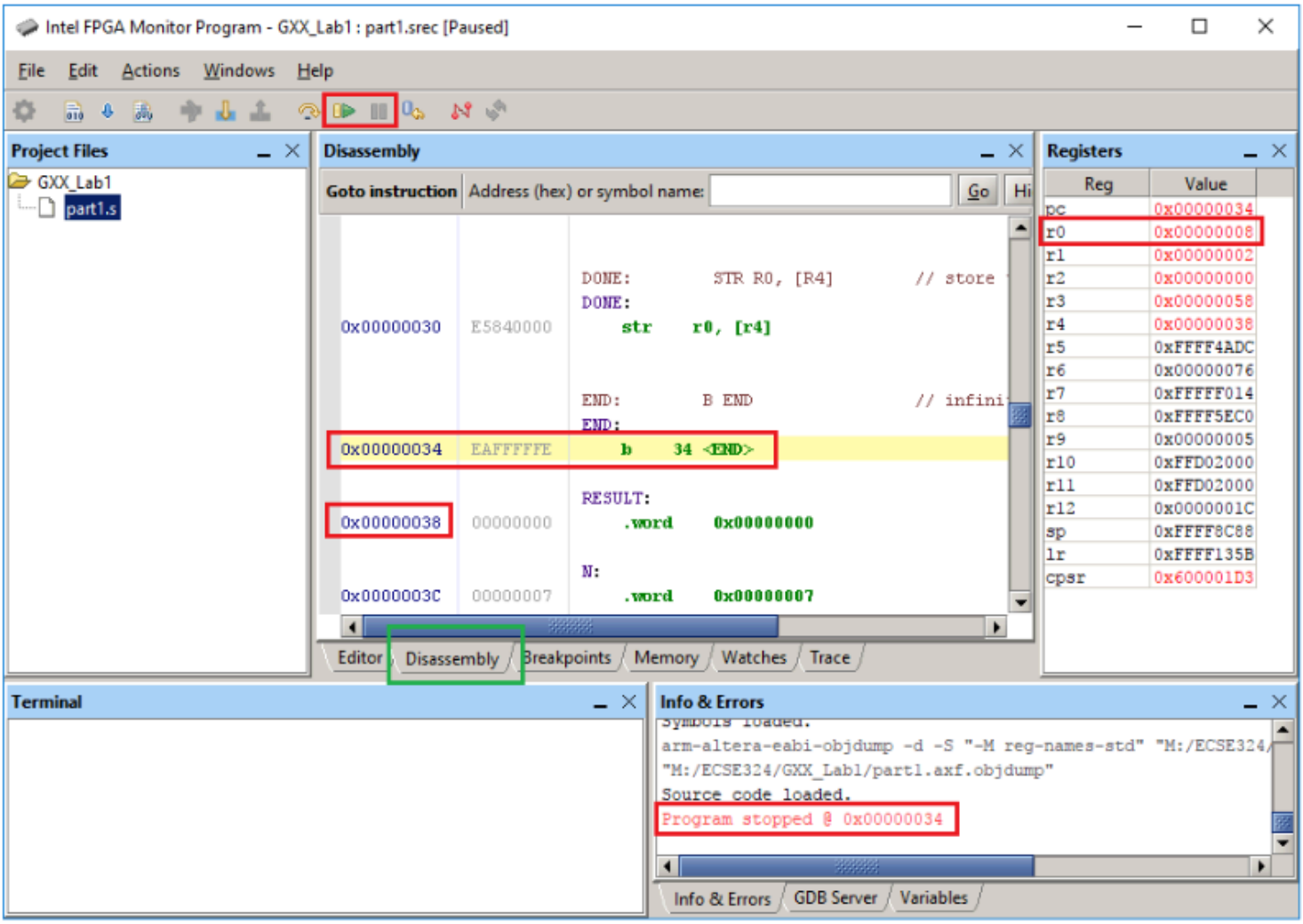


Figure 13: Using the IDE - The Disassembly window

4. Now move over to the Memory window, and search for the value in the address of RESULT. Once again, we can see that the expected value has appeared in that memory location.

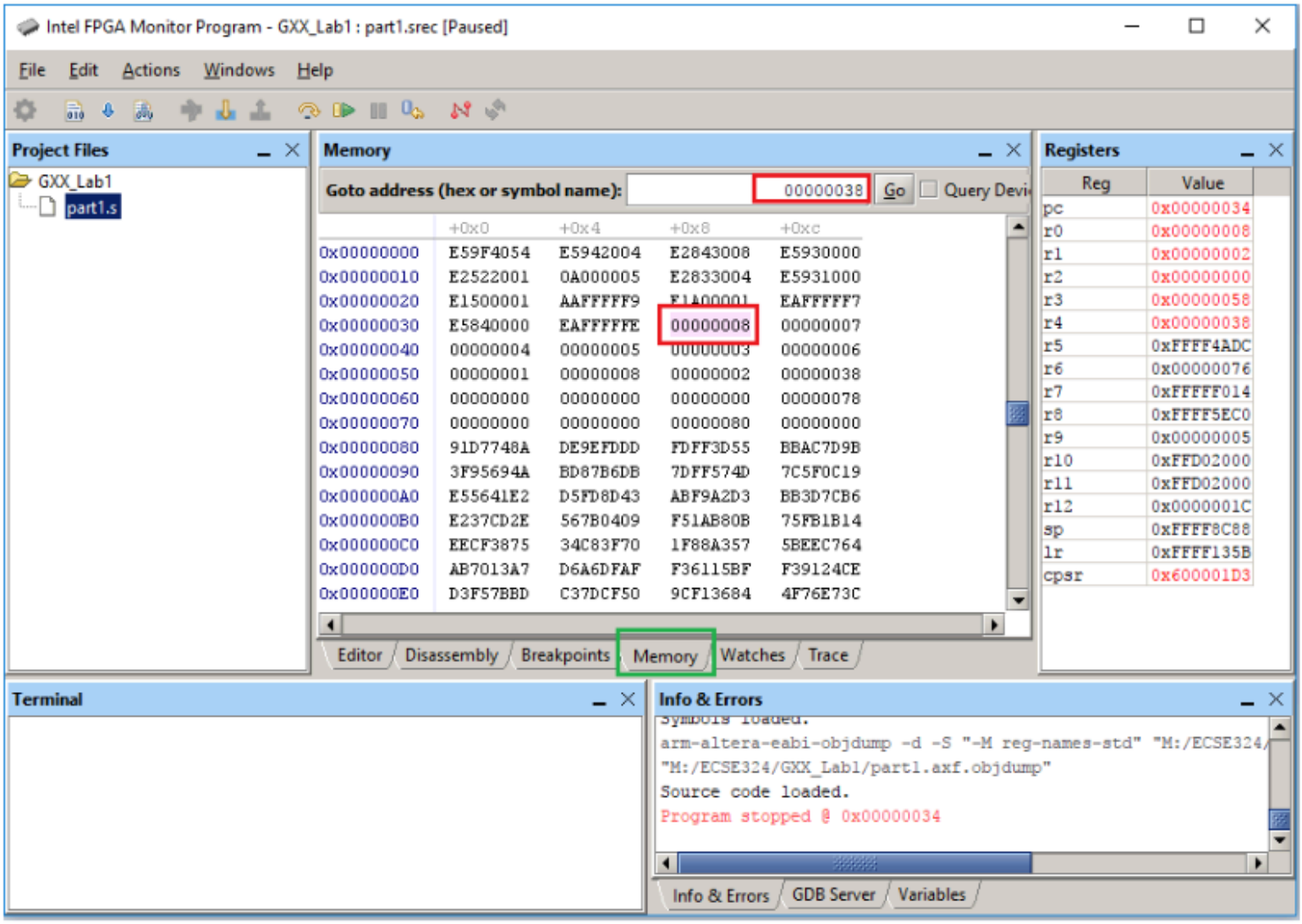


Figure 14: Using the IDE - The Memory window

2. Some programming challenges

Now that you have gone through a simple example, in which we have given you the program to be executed, you should complete the following tasks, which will require you to write your own programs.

Note: You will have to add new files to these created in your current project GXX_Lab1. Since the same label "_start" cannot be used in multiple files, and subroutines are beyond the scope of this lab, the workaround you should use in this lab is to only have one file added to the project at any given time!

2.1 Finding a range of a set of data

The range of a set of data is defined as the difference between the max and the min values in the set.

Example: Jim took 7 math tests. What is the range of his test scores, which are: 89, 73, 84, 91, 87, 77, 94?

Solution: Ordering the test scores from least to greatest, we get: 73, 77, 84, 87, 89, 91, 94

max - min = 94 - 73 = 21

Answer: The range of these test scores is 21 points.

In this part of the lab you will be asked to write an assembly program finding a rage of a set of data. The program should accept input values – using a similar approach as shown in Part 1. Save your code in a file named "rangecal.s"

Hint: You can reuse your code from Part 1 to computer the maximum value. Then, you can make a simple modification to this code to get code, which computer the minimum.

For your program demo calculate the range of 5 numbers of your choice.

2.2 Maximum and minimum values of an algebraic expression

Given an algebraic expression of the form $(x_1 + x_2 + x_3 + \ldots + x_n) * (y_1 + y_2 + \ldots + y_m)$ and (n + m) integers. Find the maximum and minimum value of the expression using the given integers.

A **simple solution**: consider all possible combinations of *n* numbers and remaining *m* numbers and calculate their values, from which maximum value and minimum value can be derived.

Suppose S be the sum of all the $(n + m)$ numbers in the expression and X be the sum of the n numbers on the left of expression. The sum of the m numbers on the right of expression will be

represented as ($S - X$). There can be many possible values of X from the given ($n + m$) numbers and hence the problem gets reduced to simply iterate through all values of X and keeping track of the minimum and maximum value of $X * (S - X)$.

Examples :

Input : n = 2, m = 2

arr[] = {1, 2, 3, 4}

Output : Maximum : 25

Minimum : 21

The expression is (x1 + x2) * (y1 + y2) and

the given integers are 1, 2, 3 and 4. Then

maximum value is (1 + 4) * (2 + 3) = 25

whereas minimum value is (4 + 3) * (2 + 1)

= 21.

Input : n = 3, m = 1

arr[] = {1, 2, 3, 4}

Output : Maximum : 24

Minimum : 9

For your lab demo select 4 numbers (different than in the example above) and calculate **max** and **min** of the arithmetic expression $(x_1 + x_2)*(y_1 + y_2)$. Name your program "maxmin.s"

3. Grading and report

The TA will ask to see the following deliverables during the demo (the corresponding portion of your grade for each is indicated in brackets):

- Largest integer programming (15%)
- Finding a range of set of data (30%)
- Maximum and minimum values of arithmetic expression (35%)

A portion of the grade is reserved for answering questions about the code, which is awarded individual to group members. All members of your group should be able to answer any questions the TA has about any part of the deliverables, whether or not you wrote the particular part of the code the TA asks about.

Finally, the remaining 20% f the grade for this Lab will go towards a report. Write up a short (2-3) page report that gives: a brief description of each part completed, approach taken, and challenges faced, if any. Please do not include the entire code in the body of the report. Save the space for elaborating on possible improvements you made or could have made to the program.

Your final submission should be a single compressed folder that contains your report and the assembly files: “part1.s”, “rangecal.s” and “maxmin.s”.