Lab 1: Introduction to ARM Programming

ECSE 324 - Computer Organization

Winter 2019

Introduction

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

1 Working with the DE1-SoC Computer System

For this course, we will be working with the DE1-SoC Computer System, which is composed of an ARM 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 ARM*. You will also find it useful to refer to the *ARM Architecture Reference Manual*. These documents can be found on myCourses. It may help to keep these manuals open as you work.

1.2 Your first assembly program

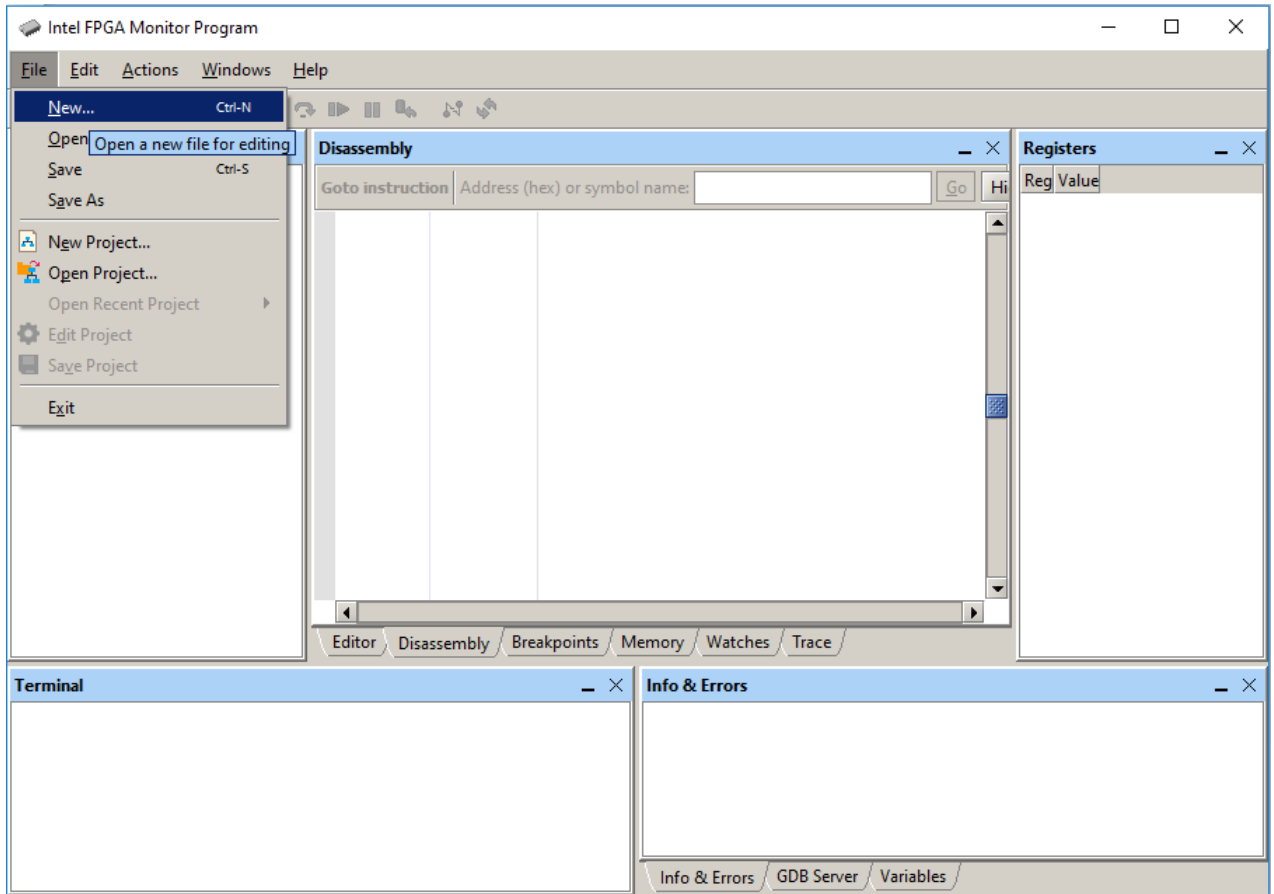1. Open the 'Intel FPGA Monitor Program 16.1' from the desktop icon and select File->New.

Figure 1: Your first assembly program - Step 1

2. In the new editor window, type out the code as shown in Figure 2 and save this file as 'part1.s' within a new folder 'GXX_Lab1' on your network drive. **Here, GXX stands for your group number! eg. 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 be used throughout this course, especially with assembly programming!

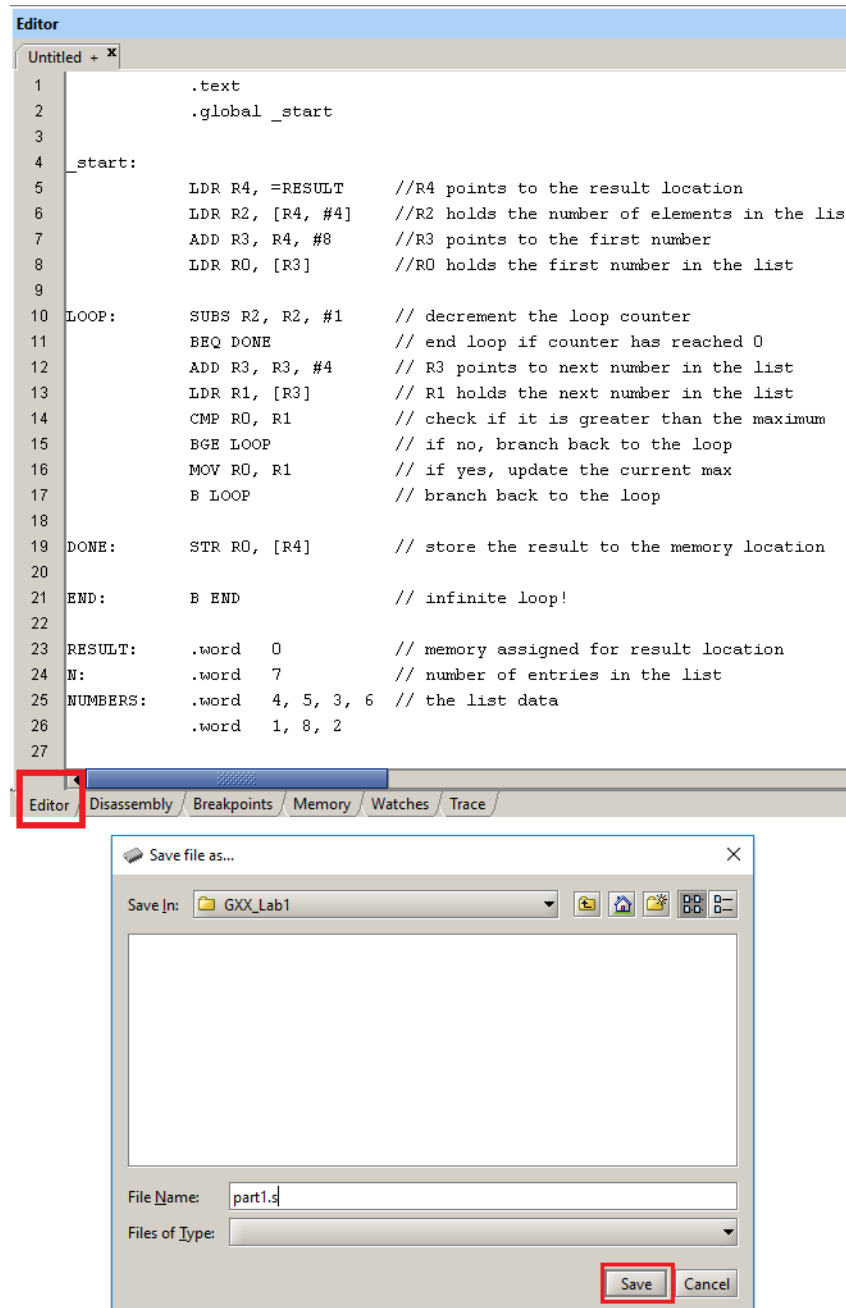**NOTE: The indentation is important. The code will not compile if not indented as shown.**

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.
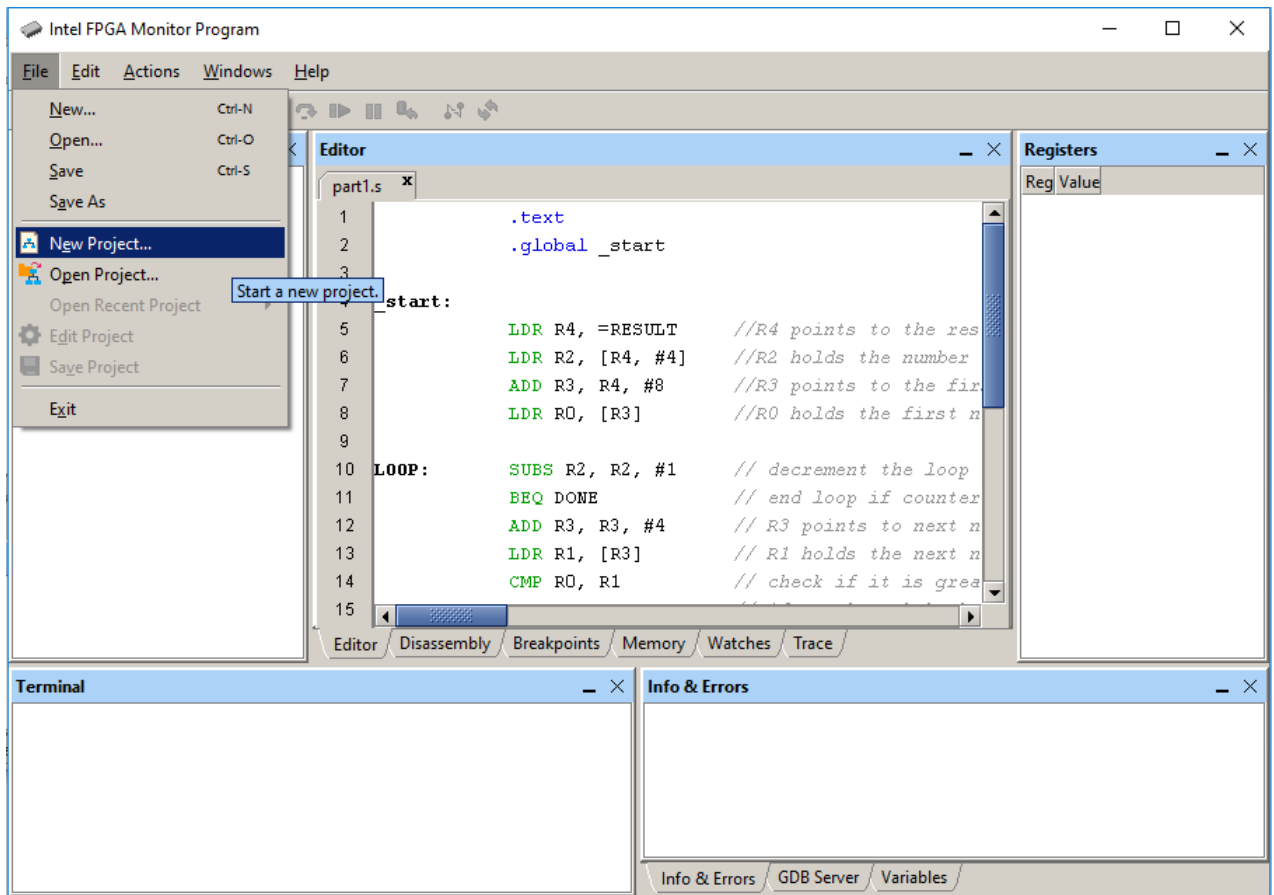

Figure 3: Your first assembly program - Step 3

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'.
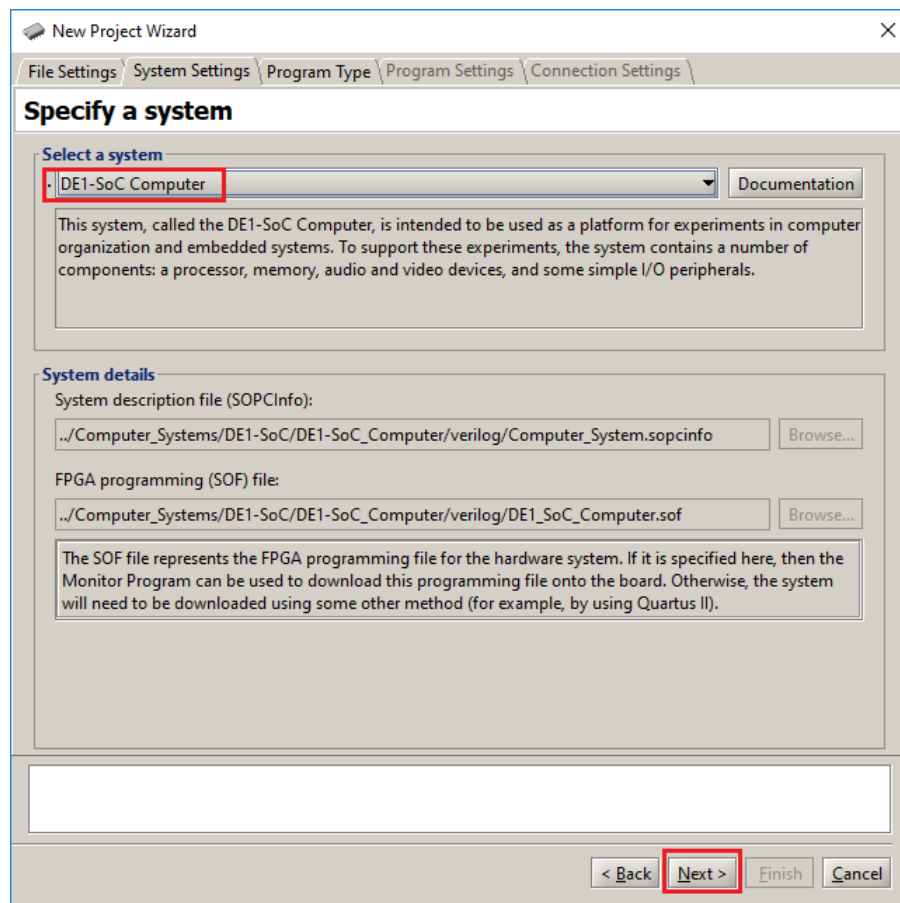

Figure 5: Your first assembly program - Step 5

6. In the next window, under 'Program type' select the 'Assembly Program' and click 'Next'.
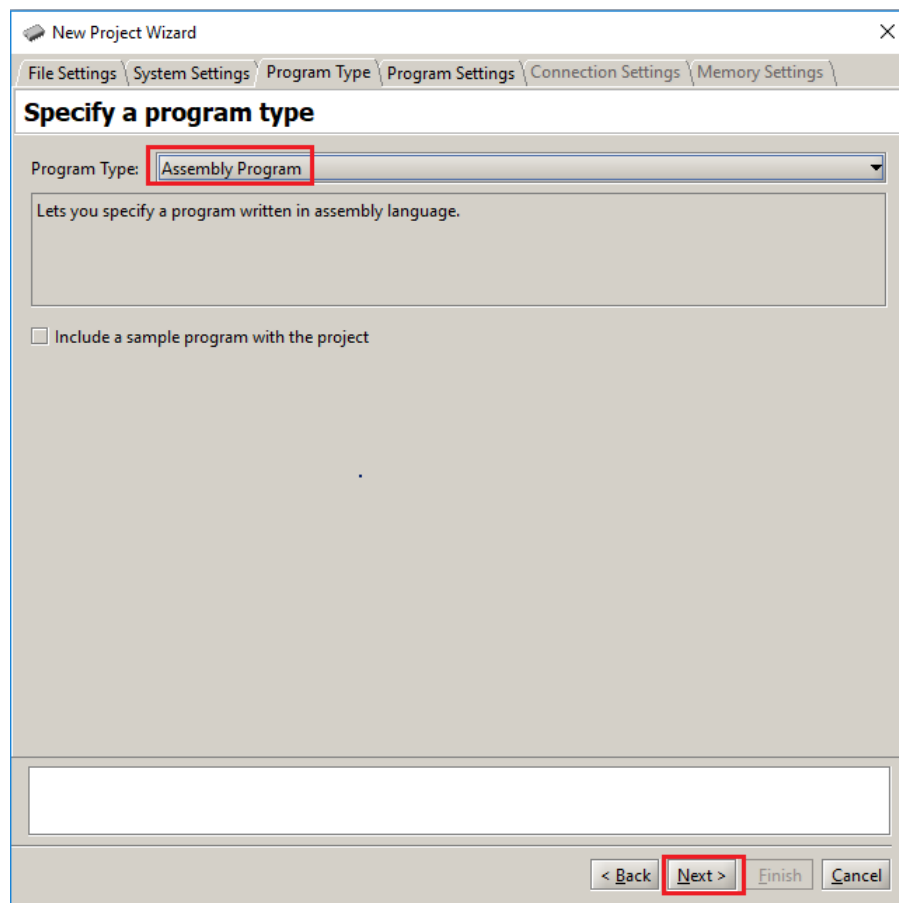
Figure 6: Your first assembly program - Step 6

7. In the next window 'Specify program details', click on 'Add...' and select the file 'part1.s' created in step 1, and click 'Next'.
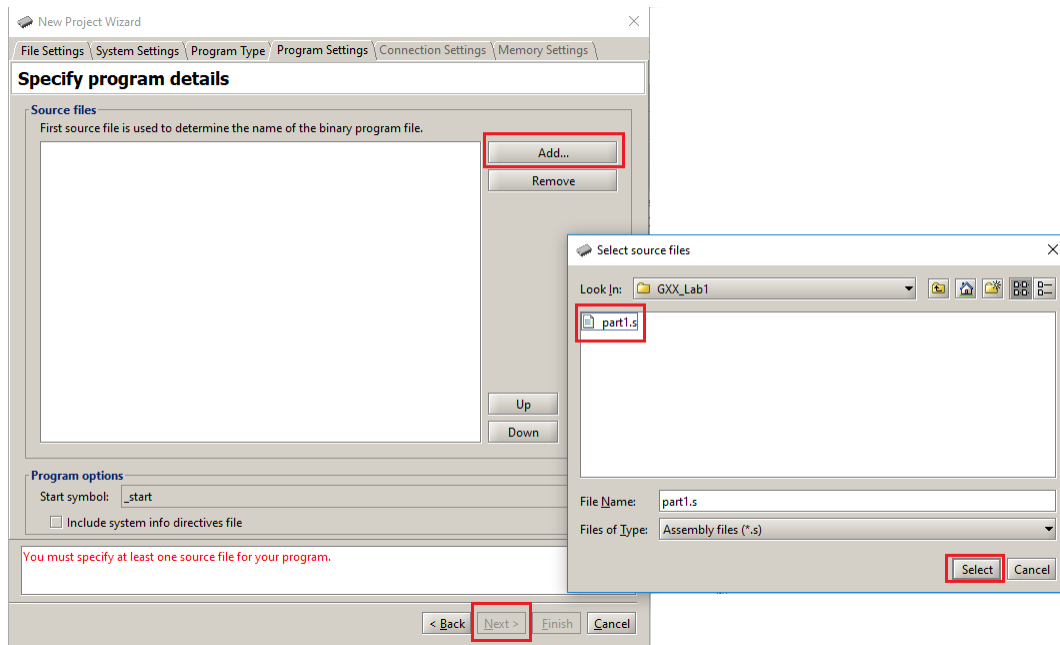
Figure 7: Your first assembly program - Step 7

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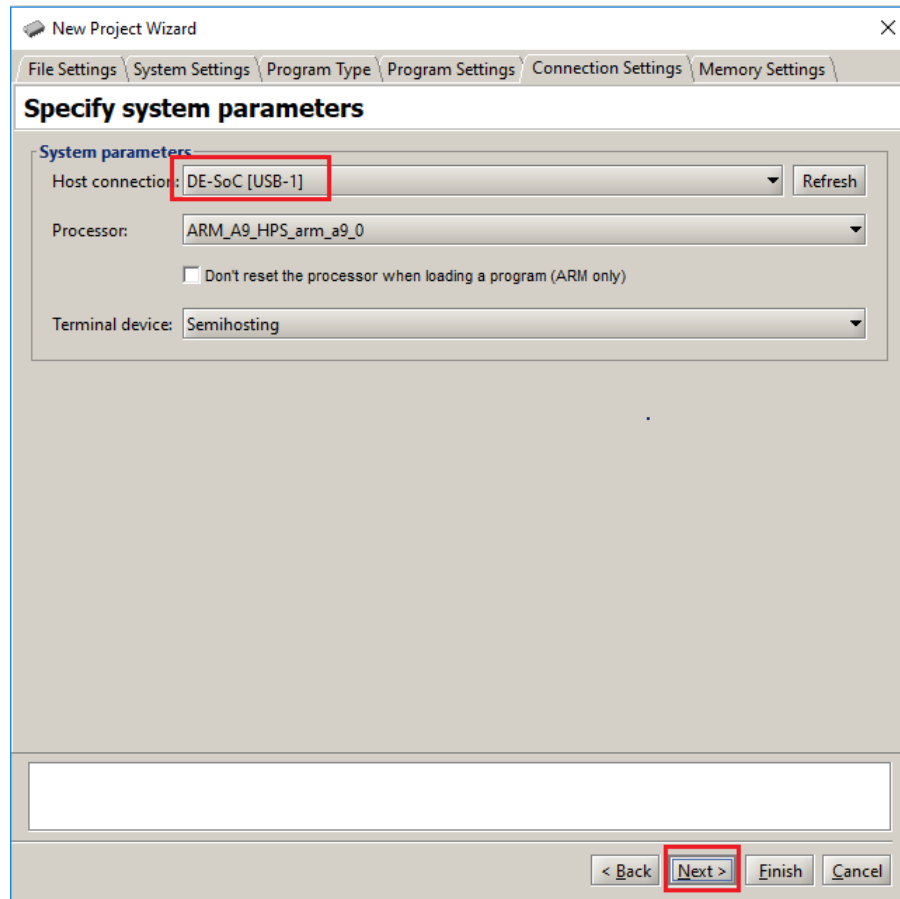
9. In the next window 'Specify program memory settings', simply click 'Finish'.
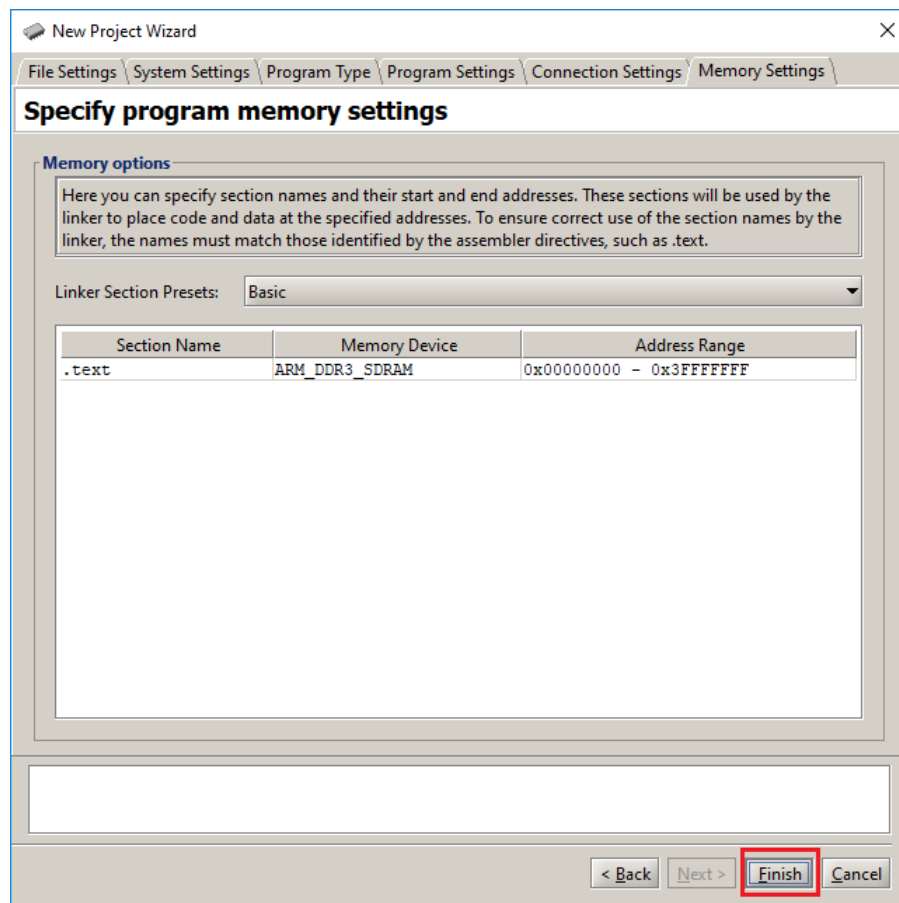

Figure 9: Your first assembly program - Step 9

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'.
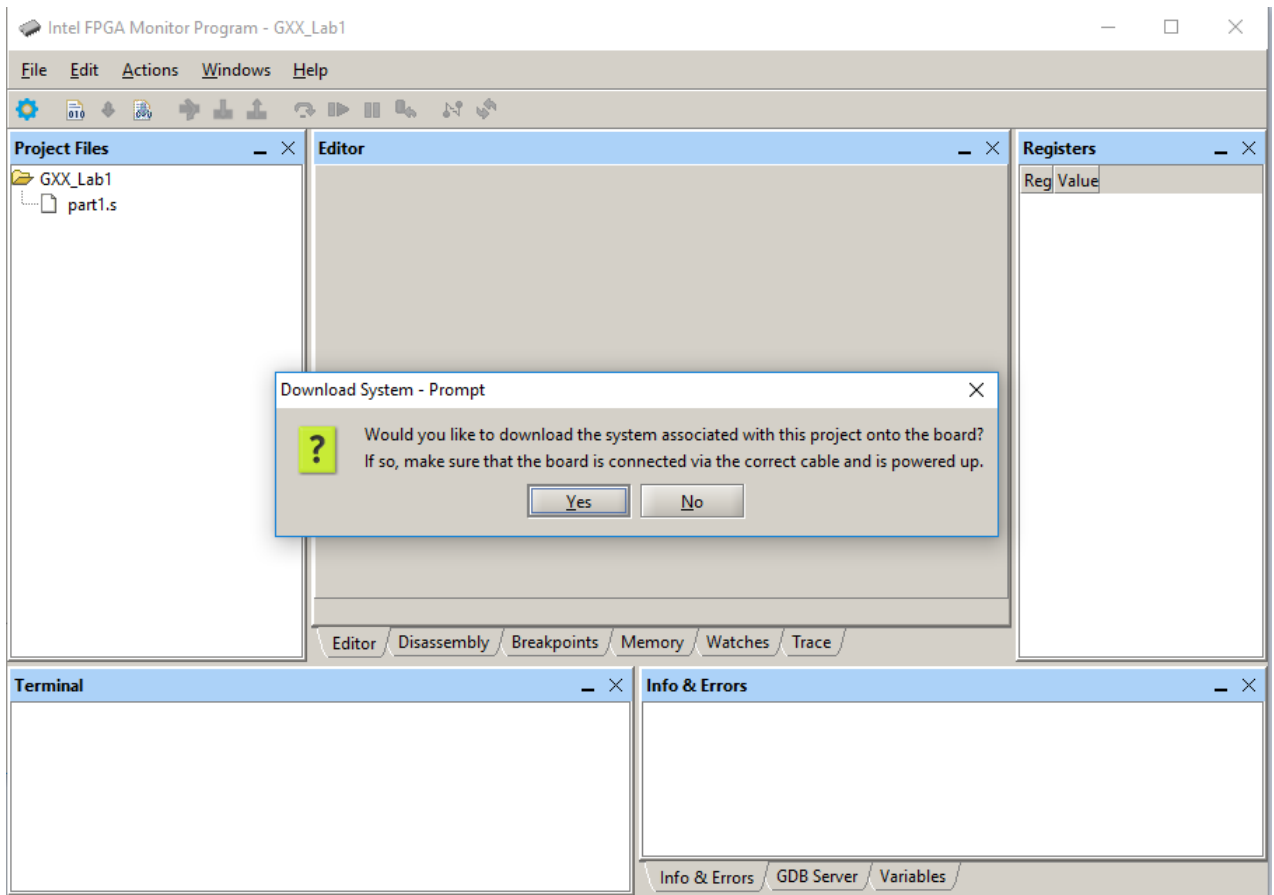

Figure 10: Your first assembly program - Step 10

1.3 Using the IDE

Now that we have created our first 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. Figure 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 the 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.

    The red box highlights three useful buttons in development mode - 'Compile', 'Load', and 'Compile & Load'. Their functions are self-explanatory.

    Actually, 'compiling' refers to converting higher level computer code (such as C code) 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.
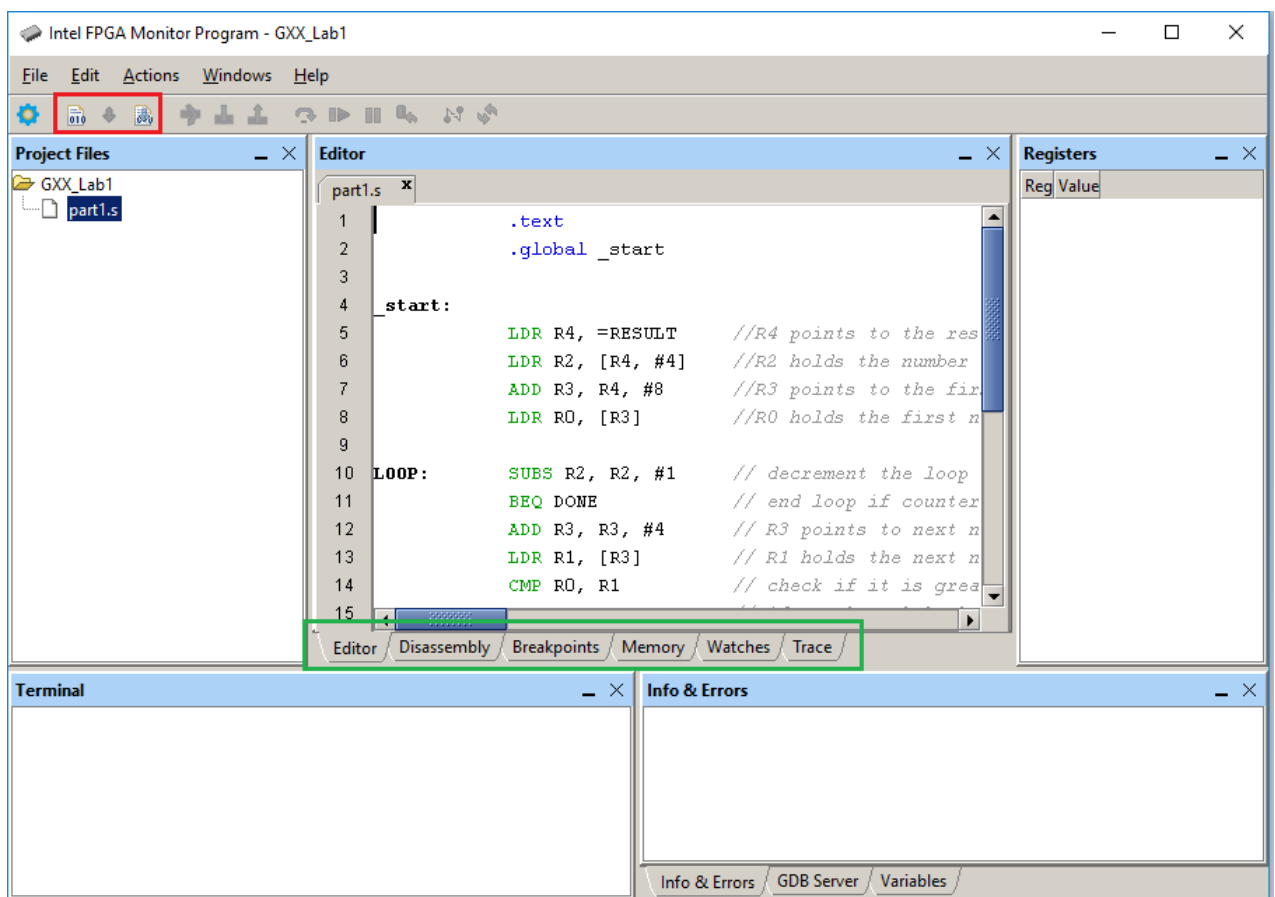

Figure 11: Using the IDE - Development mode

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 contents 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.
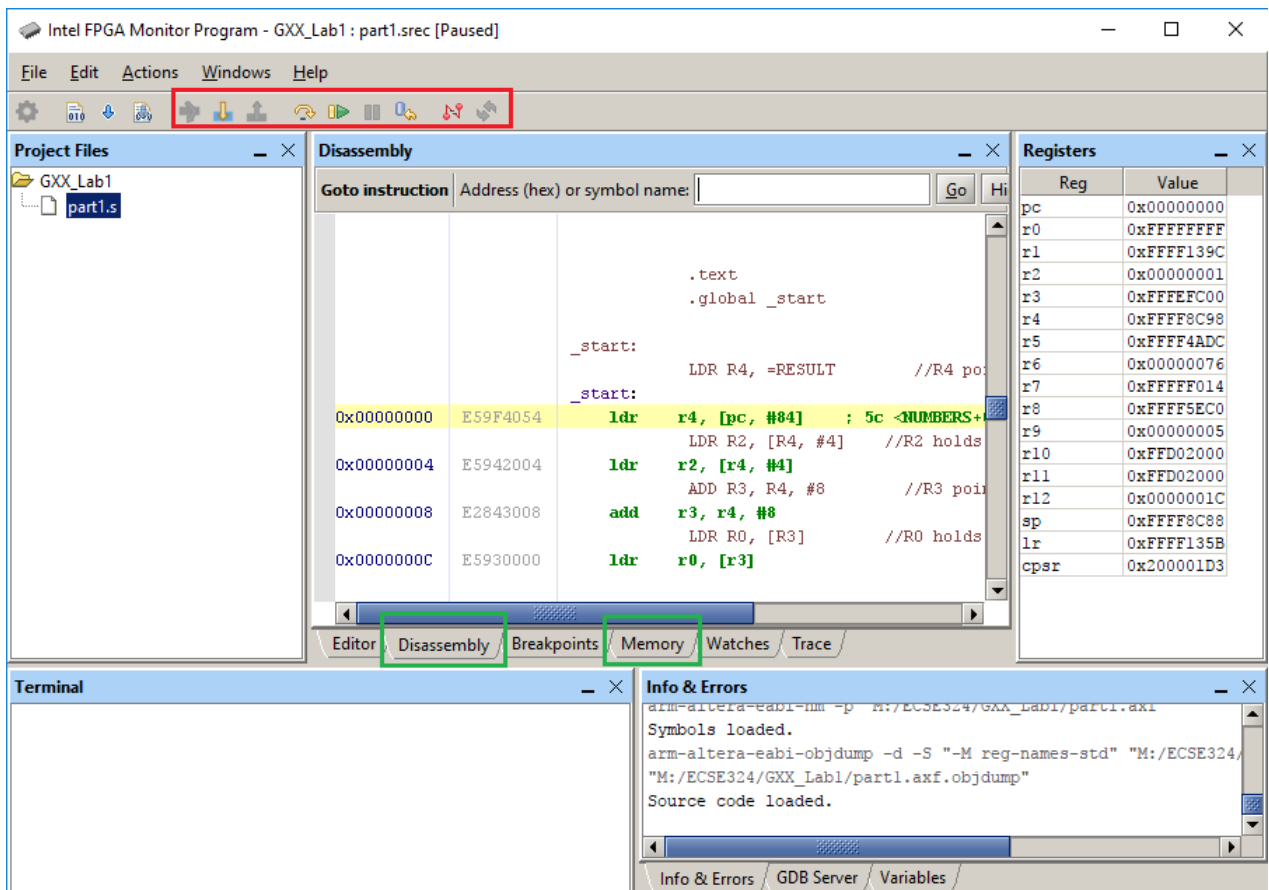
Figure 12: Using the IDE - Debug mode

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're 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.
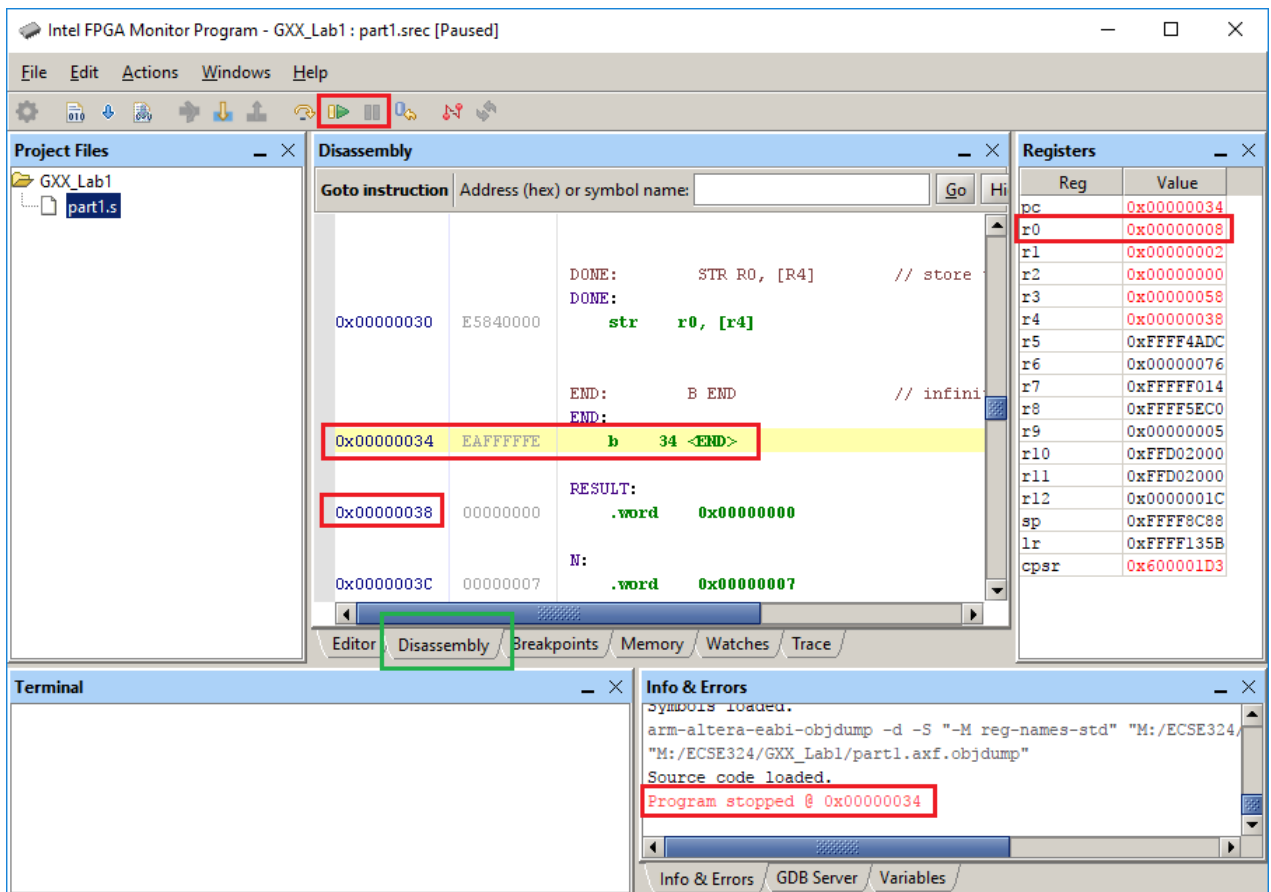

Figure 13: Using the IDE - The Disassembly window

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.
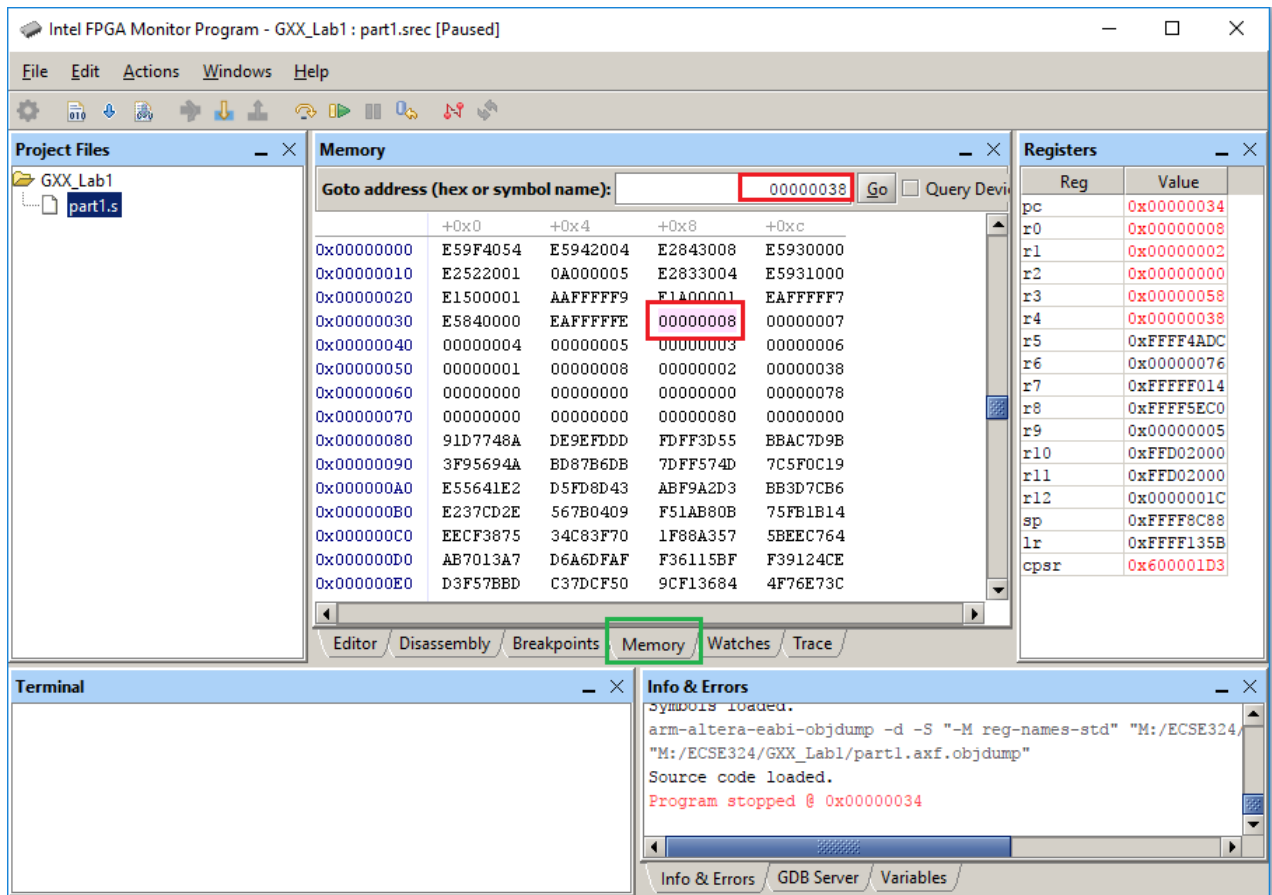

Figure 14: Using the IDE - The Memory window

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 the new files you will create to 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 Fast standard deviation computation

Suppose that you would like to use the ARM processor to compute the standard deviation of a signal $X = \{x_1, x_2, \ldots, x_N\}$. The formula for the standard deviation is:

$$\hat{\sigma} = \sqrt{\frac{\sum_{i=1}^{N}(x_i - \hat{\mu})^2}{N - 1}} \tag{1}$$

where $\hat{\mu}$ is the average value of the signal. Unfortunately, implementing this formula requires multiplication, division, and square root operations, which are not available as instructions on all processors and are slow to emulate using other instructions. The standard deviation can be approximately computed in a more hardware-friendly way using the so-called "range rule":

$$\hat{\sigma} \approx \frac{x_{max} - x_{min}}{4} \tag{2}$$

where x_{max} and x_{min} are the maximum value and minimum value of the signal, respectively.

Write an ARM assembly program which computes the standard deviation of a signal, using the range rule. The program should accept input values - more specifically, the number of samples in the signal and their values - using a similar approach as shown in Part 1. Save your code in a file named 'stddev.s'

(*Hint:* you can reuse your code from Part 1 to compute the maximum value. Then, you can make a simple modification to this code to get code which computes the minimum. Also, remember that dividing by a power of 2 can be implemented using shift instructions.)

2.2 Centering an array

It is often necessary to ensure that a signal is "centered" (that is, its average is 0). For example, DC signals can damage a loudspeaker, so it is important to center an audio signal to remove DC components before sending the signal to the speaker.

You can center a signal by calculating the average value of the signal and subtracting the average from every sample of the signal. Write an ARM assembly program to center a signal. In this example, store the resulting centered signal 'in place' - i.e. in the same memory location that the input signal is passed in. The program should be able to accept the signal length as an input parameter. **In order to simplify calculations, work with the assumption that only signal lengths that are powers of two can be passed to the program**. Save your code in a file named 'center.s'

2.3 Sorting

Write an ARM assembly program which sorts an array in ascending order. You could use the simple bubble sort algorithm:

```
// Given an array A of length N
sorted = false
while not sorted:
    sorted = true
    for i = 2 to N:
        if A[i] < A[i-1], swap A[i] with A[i-1] and set sorted = false
```

You could also implement a more sophisticated sorting algorithm. Store the resulting sorted array 'in place'. The program should be able to accept the array length as an input parameter. Save your code in a file named 'sort.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 program (10%)

- Standard deviation program (15%)

- Centering program (25%)

- Sorting program (30%)

A portion of the grade is reserved for answering questions about the code, which is awarded individually 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. Full marks are awarded for a deliverable only if the program functions correctly and the TA's questions are answered satisfactorily.

Finally, the remaining 20% of the grade for this Lab will go towards a report. Write up a short (2-3) page report that given a brief description of each part completed, the approach taken, and the challenges faced, if any. Please don't 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, such as a feature to detect empty (length 0) arrays, etc.

Your final submission should be a **single compressed folder** that contains your report and the four assembly files - 'part1.s', 'stddev.s', 'center.s', and 'sort.s'.

You should demo your code before Friday, February 8th, **within your assigned lab period.** The report for Lab 1 is due by 11:59 pm, February 15th.