Lab 1: Introduction to ARM Programming

ECSE 324 - Computer Organization

Winter 2020

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* as well as the *ARM Instruction Set Quick Reference Card*. 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.

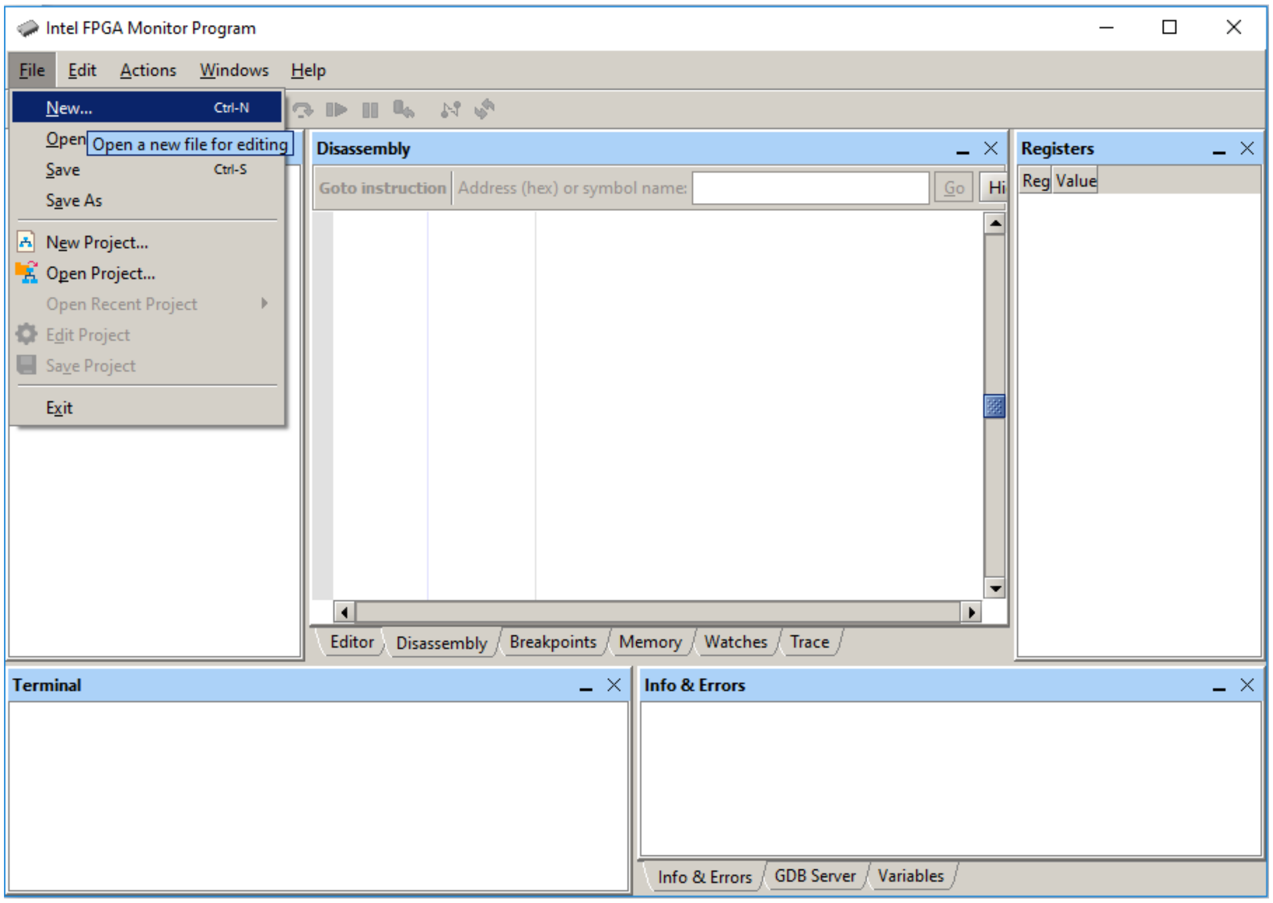
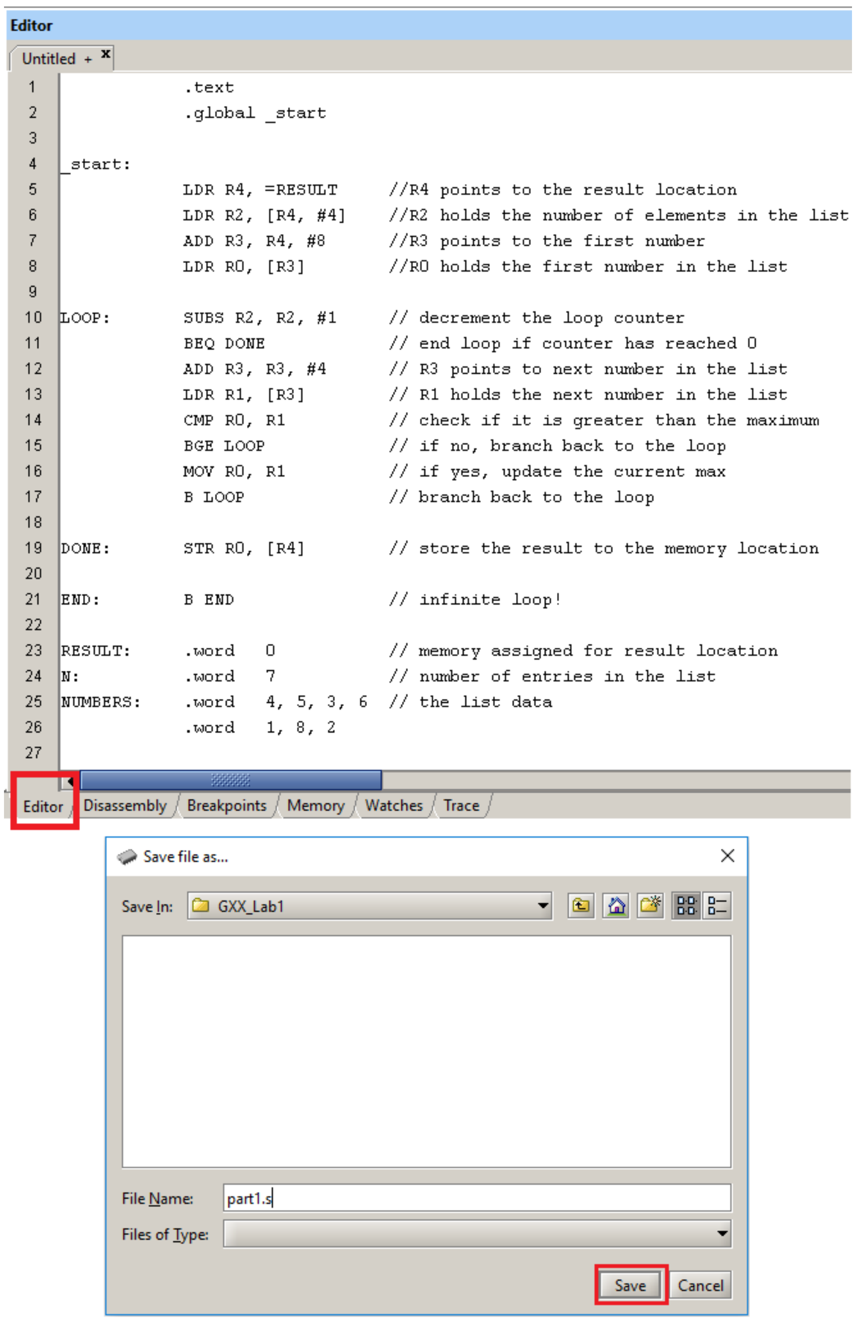


Figure 1: Your first assembly program - Step 1

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.



```
            .text
            .global _start

_start:
            LDR R4, =RESULT     //R4 points to the result location
            LDR R2, [R4, #4]    //R2 holds the number of elements in the list
            ADD R3, R4, #8      //R3 points to the first number
            LDR R0, [R3]        //R0 holds the first number in the list

LOOP:       SUBS R2, R2, #1     // decrement the loop counter
            BEQ DONE            // end loop if counter has reached 0
            ADD R3, R3, #4      // R3 points to next number in the list
            LDR R1, [R3]        // R1 holds the next number in the list
            CMP R0, R1          // check if it is greater than the maximum
            BGE LOOP            // if no, branch back to the loop
            MOV R0, R1          // if yes, update the current max
            B LOOP              // branch back to the loop

DONE:       STR R0, [R4]        // store the result to the memory location

END:        B END               // infinite loop!

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

```

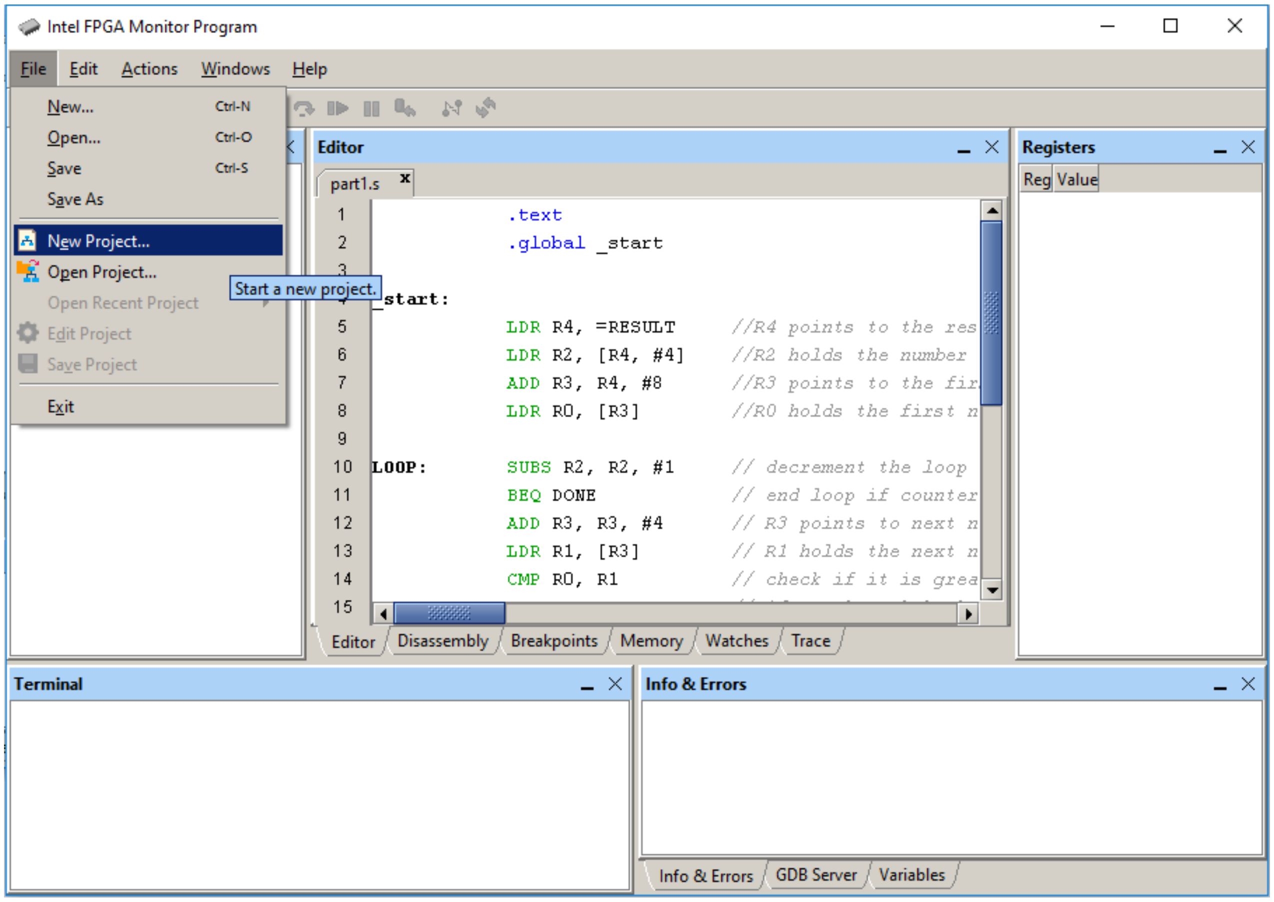
Save In: GXX_Lab1

File Name: part1.s

Files of Type:

Figure 2: Your first assembly program - Step 2

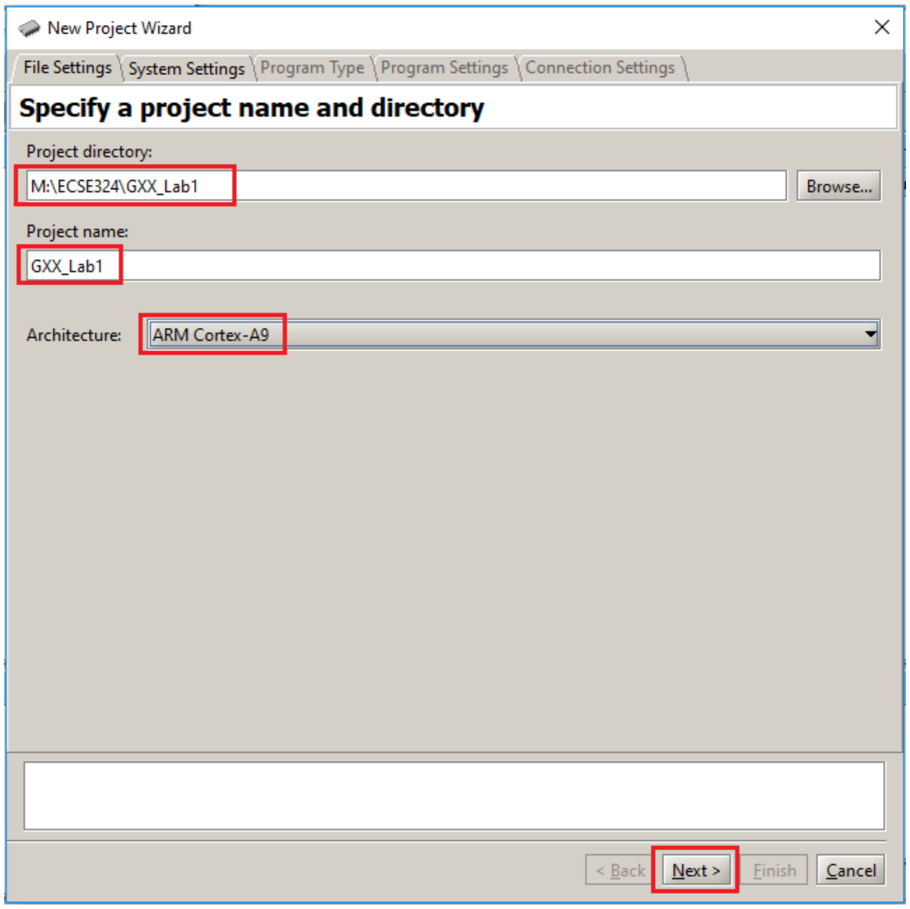
3. Open the 'Intel FPGA Monitor Program 16.1' from the desktop icon and select File->New Project.



```
            .text
            .global _start

_start:
            LDR R4, =RESULT     //R4 points to the res
            LDR R2, [R4, #4]    //R2 holds the number
            ADD R3, R4, #8      //R3 points to the fir
            LDR R0, [R3]        //R0 holds the first n

LOOP:       SUBS R2, R2, #1     // decrement the loop
            BEQ DONE            // end loop if counter
            ADD R3, R3, #4      // R3 points to next n
            LDR R1, [R3]        // R1 holds the next n
            CMP R0, R1          // check if it is grea
```

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'.



Specify a project name and directory

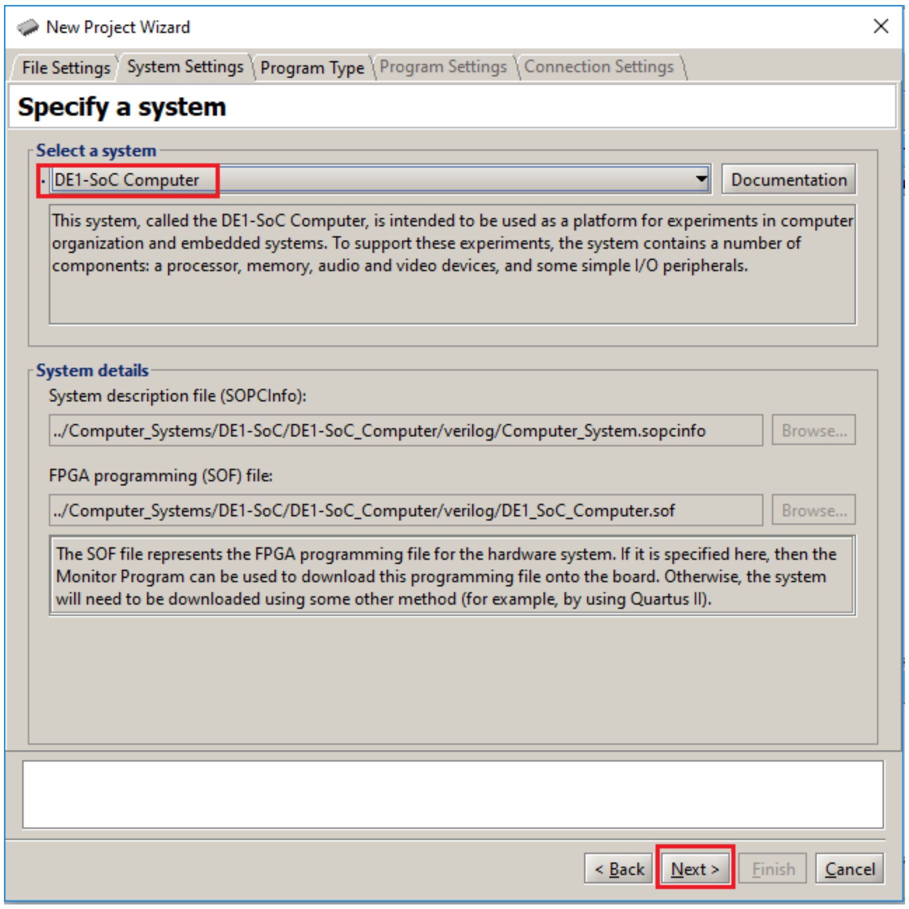
Project directory: M:\ECSE324\GXX_Lab1

Project name: GXX_Lab1

Architecture: ARM Cortex-A9

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'.



Specify a system

Select a system

DE1-SoC Computer

This system, called the DE1-SoC Computer, is intended to be used as a platform for experiments in computer organization and embedded systems. To support these experiments, the system contains a number of components: a processor, memory, audio and video devices, and some simple I/O peripherals.

System details

System description file (SOPCInfo):

../Computer_Systems/DE1-SoC/DE1-SoC_Computer/verilog/Computer_System.sopcinfo

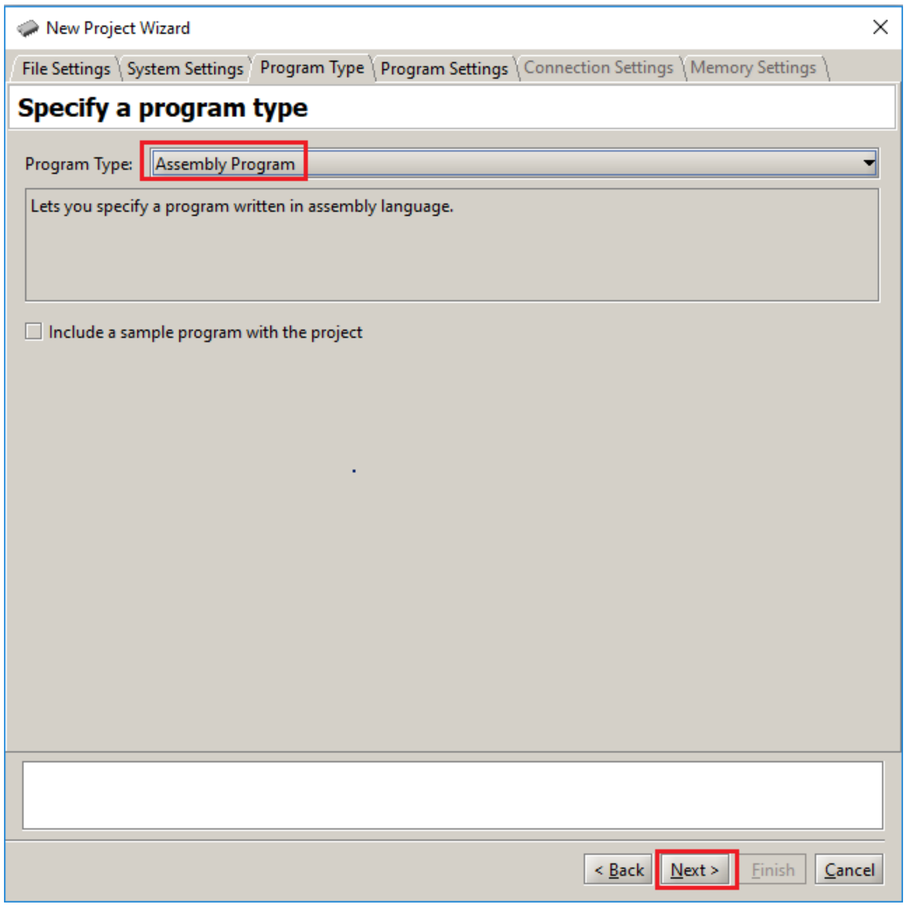
FPGA programming (SOF) file:

../Computer_Systems/DE1-SoC/DE1-SoC_Computer/verilog/DE1_SoC_Computer.sof

The SOF file represents the FPGA programming file for the hardware system. If it is specified here, then the Monitor Program can be used to download this programming file onto the board. Otherwise, the system will need to be downloaded using some other method (for example, by using Quartus II).

Figure 5: Your first assembly program - Step 5

6. In the next window, under ‘Program type’ select the ‘Assembly Program‘ and click ‘Next’.


New Project Wizard

File Settings | System Settings | Program Type | Program Settings | Connection Settings | Memory Settings

Specify a program type

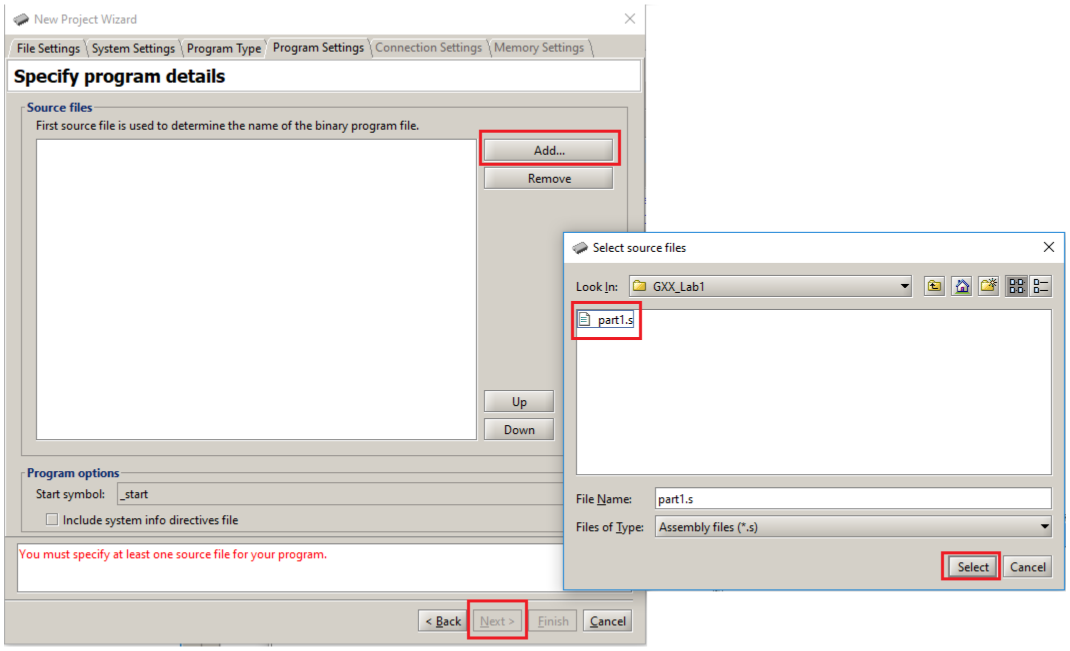
Program Type: Assembly Program

Lets you specify a program written in assembly language.

☐ Include a sample program with the project

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’.



Specify program details

Source files

First source file is used to determine the name of the binary program file.

Program options

Start symbol: _start

Include system info directives file

You must specify at least one source file for your program.

Look In: GXX_Lab1

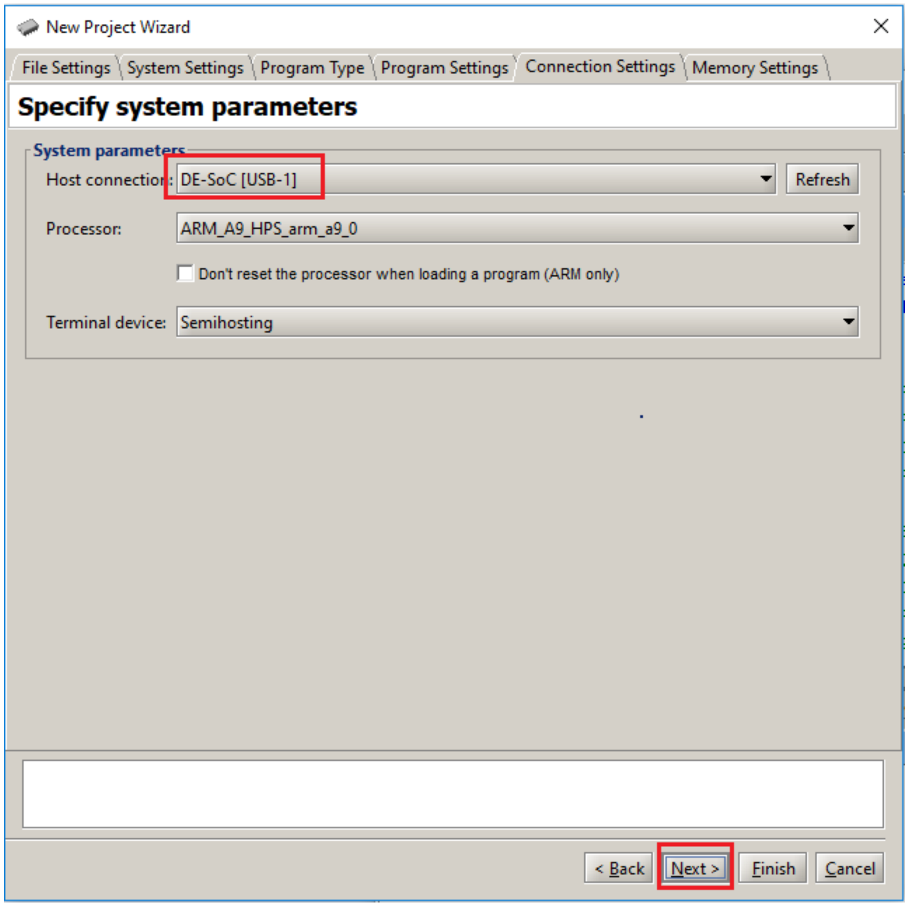
part1.s

File Name: part1.s

Files of Type: Assembly files (*.s)

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.



New Project Wizard

File Settings | System Settings | Program Type | Program Settings | Connection Settings | Memory Settings

Specify system parameters

System parameters

Host connection: DE-SoC [USB-1]

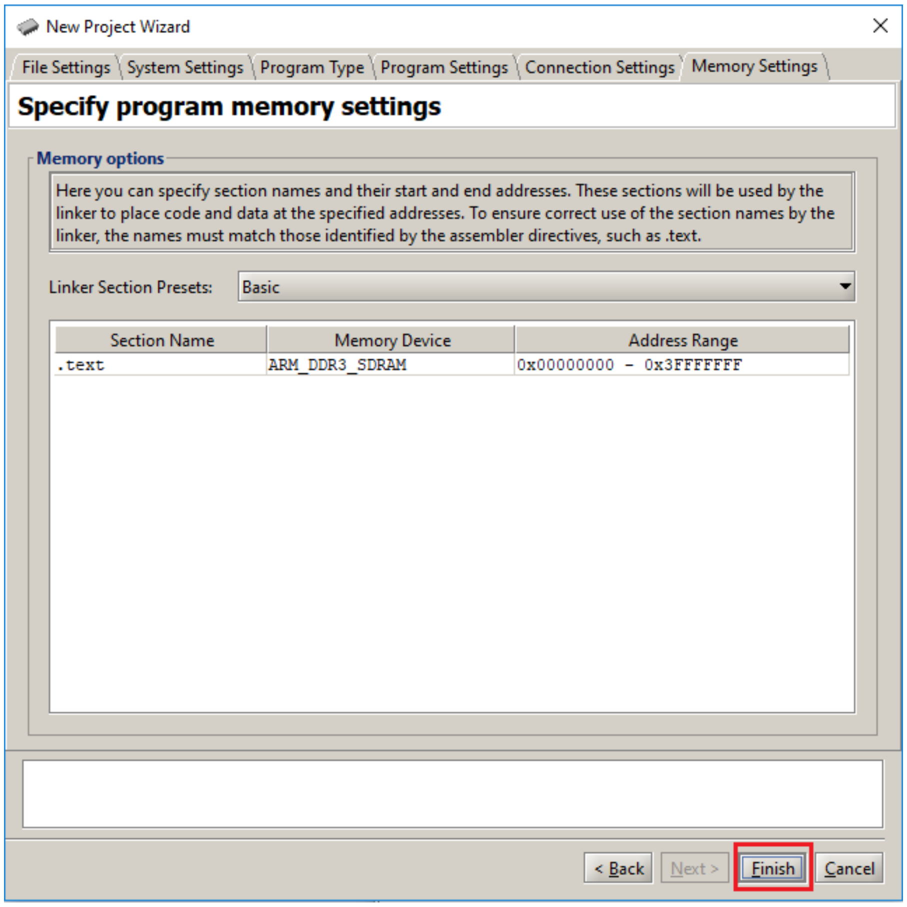
Processor: ARM_A9_HPS_arm_a9_0

☐ Don't reset the processor when loading a program (ARM only)

Terminal device: Semihosting

Figure 8: Your first assembly program - Step 8

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



Specify program memory settings

Memory options

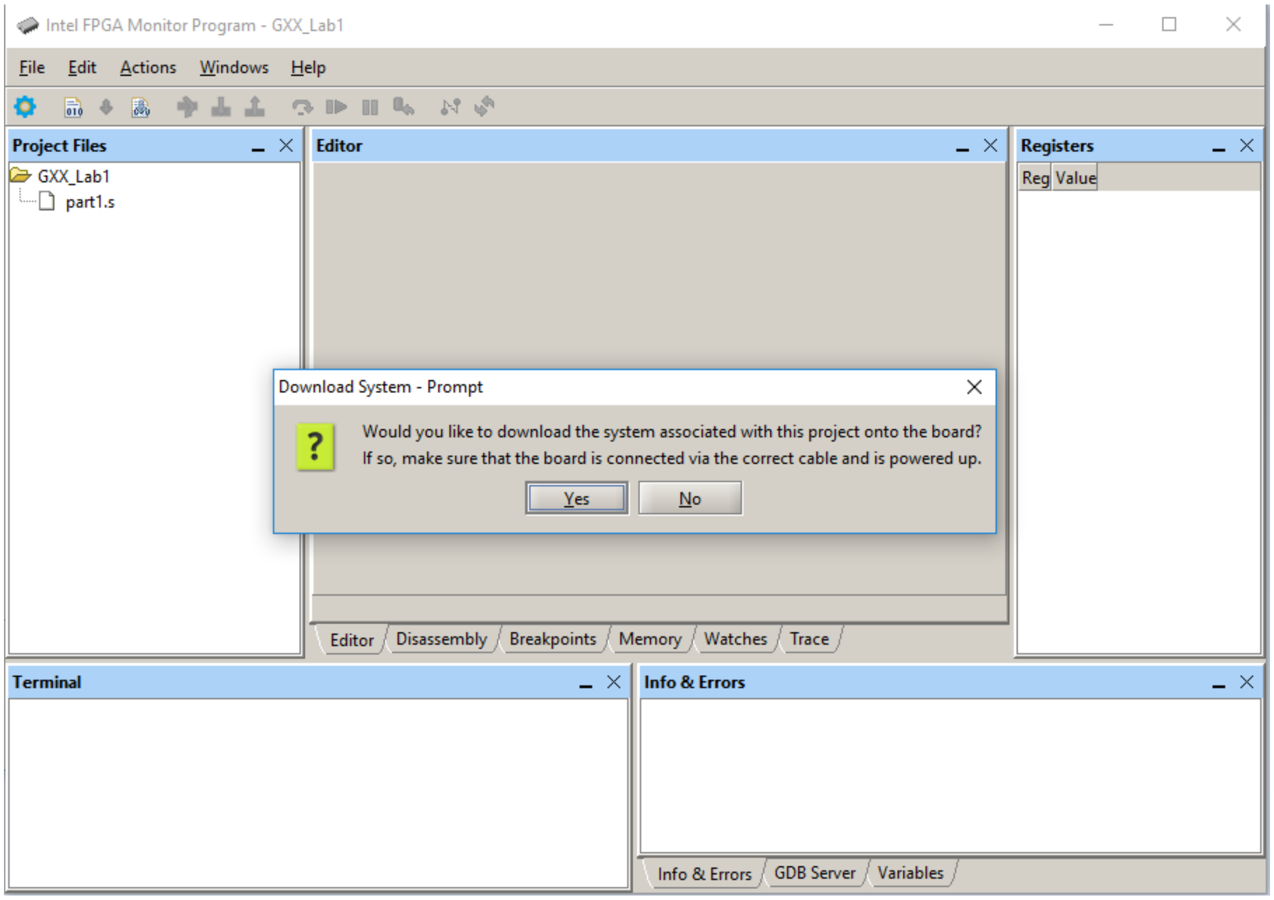
Here you can specify section names and their start and end addresses. These sections will be used by the linker to place code and data at the specified addresses. To ensure correct use of the section names by the linker, the names must match those identified by the assembler directives, such as .text.

Linker Section Presets: Basic

Section Name	Memory Device	Address Range
.text	ARM_DDR3_SDRAM	0x00000000 - 0x3FFFFFFF

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'.



Would you like to download the system associated with this project onto the board?
If so, make sure that the board is connected via the correct cable and is powered up.

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

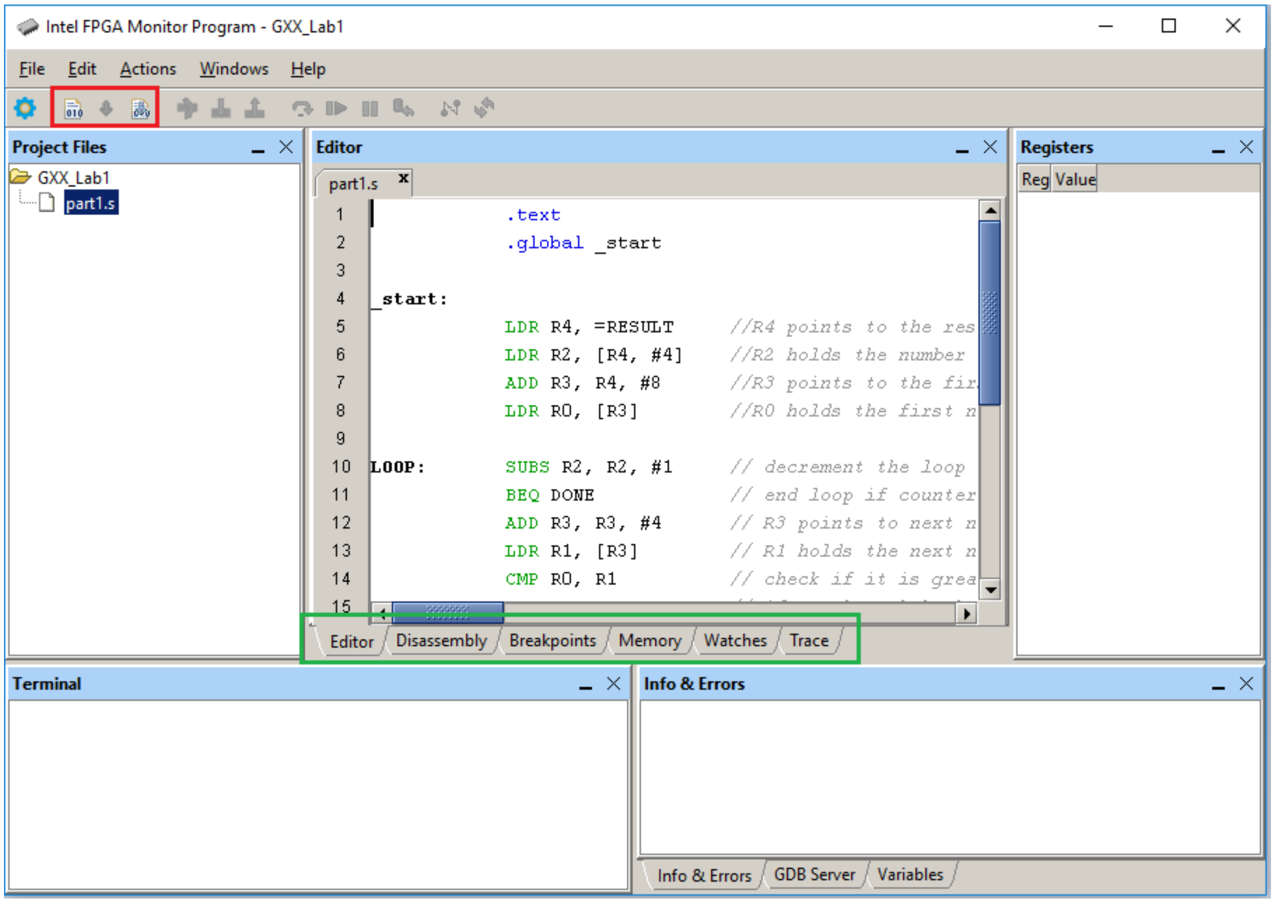
NOTE: This section only provides a very brief introduction to the IDE. More detailed information can and should be obtained in the documentation and by experience!

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.


Intel FPGA Monitor Program - GXX_Lab1
File Edit Actions Windows Help
Project Files
GXX_Lab1
part1.s
Editor
part1.s
Registers
Reg Value
Editor Disassembly Breakpoints Memory Watches Trace
Terminal
Info & Errors
Info & Errors GDB Server Variables

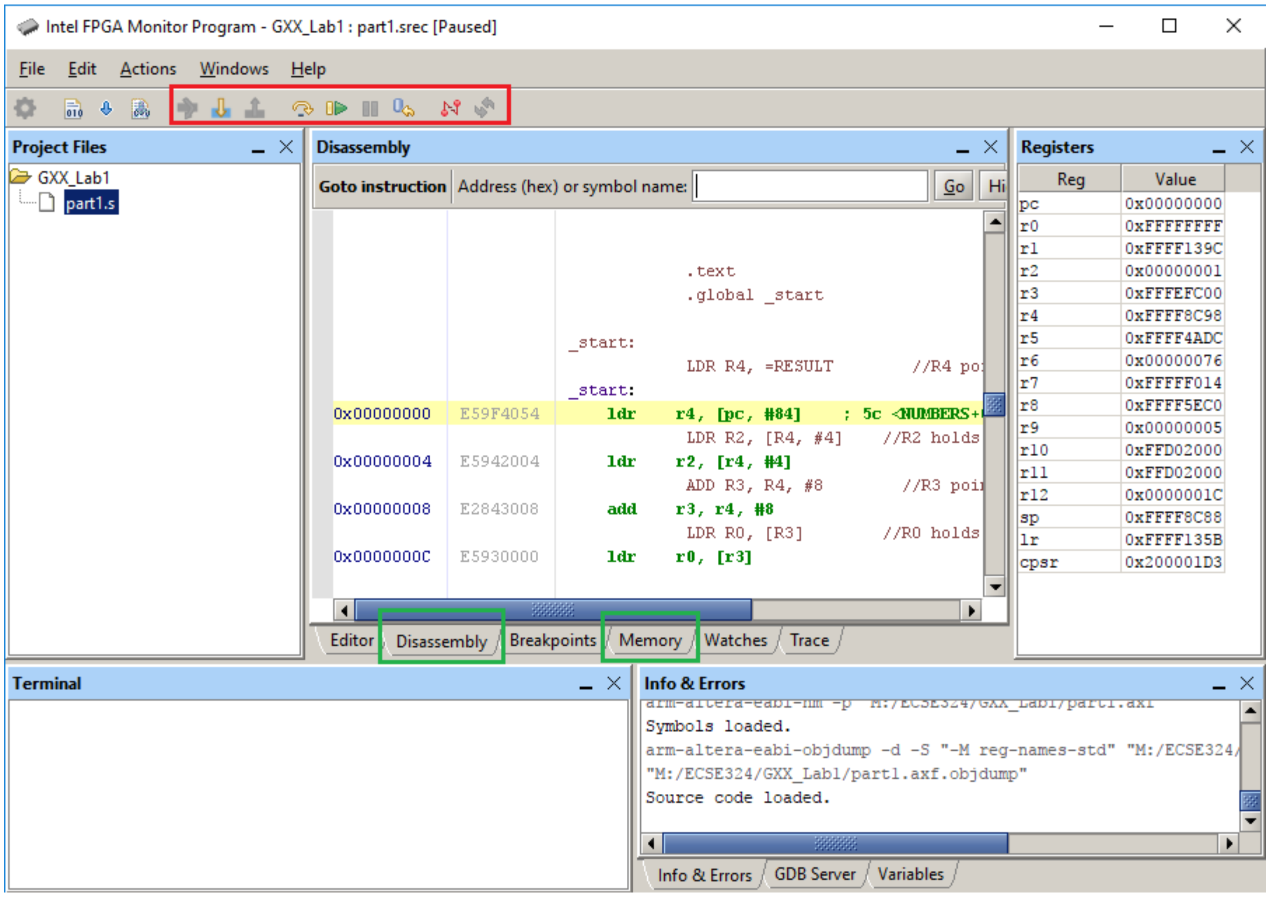
```
            .text
            .global _start

_start:
            LDR R4, =RESULT     //R4 points to the res
            LDR R2, [R4, #4]    //R2 holds the number
            ADD R3, R4, #8      //R3 points to the fir
            LDR R0, [R3]        //R0 holds the first n

LOOP:       SUBS R2, R2, #1     // decrement the loop
            BEQ DONE            // end loop if counter
            ADD R3, R3, #4      // R3 points to next n
            LDR R1, [R3]        // R1 holds the next n
            CMP R0, R1          // check if it is grea
```

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.



```
                                         .text
                                         .global _start

                              _start:
                                         LDR R4, =RESULT        //R4 po
                              _start:
0x00000000   E59F4054             ldr    r4, [pc, #84]     ; 5c <NUMBERS+
                                         LDR R2, [R4, #4]    //R2 holds
0x00000004   E5942004             ldr    r2, [r4, #4]
                                         ADD R3, R4, #8        //R3 poi
0x00000008   E2843008             add    r3, r4, #8
                                         LDR R0, [R3]        //R0 holds
0x0000000C   E5930000             ldr    r0, [r3]
```

Reg	Value
pc	0x00000000
r0	0xFFFFFFFF
r1	0xFFFF139C
r2	0x00000001
r3	0xFFFEFC00
r4	0xFFFF8C98
r5	0xFFFF4ADC
r6	0x00000076
r7	0xFFFFF014
r8	0xFFFF5EC0
r9	0x00000005
r10	0xFFD02000
r11	0xFFD02000
r12	0x0000001C
sp	0xFFFF8C88
lr	0xFFFF135B
cpsr	0x200001D3

```
arm-altera-eabi-nm -p  "M:/ECSE324/GXX_Lab1/part1.axf
Symbols loaded.
arm-altera-eabi-objdump -d -S "-M reg-names-std" "M:/ECSE324/
"M:/ECSE324/GXX_Lab1/part1.axf.objdump"
Source code loaded.
```

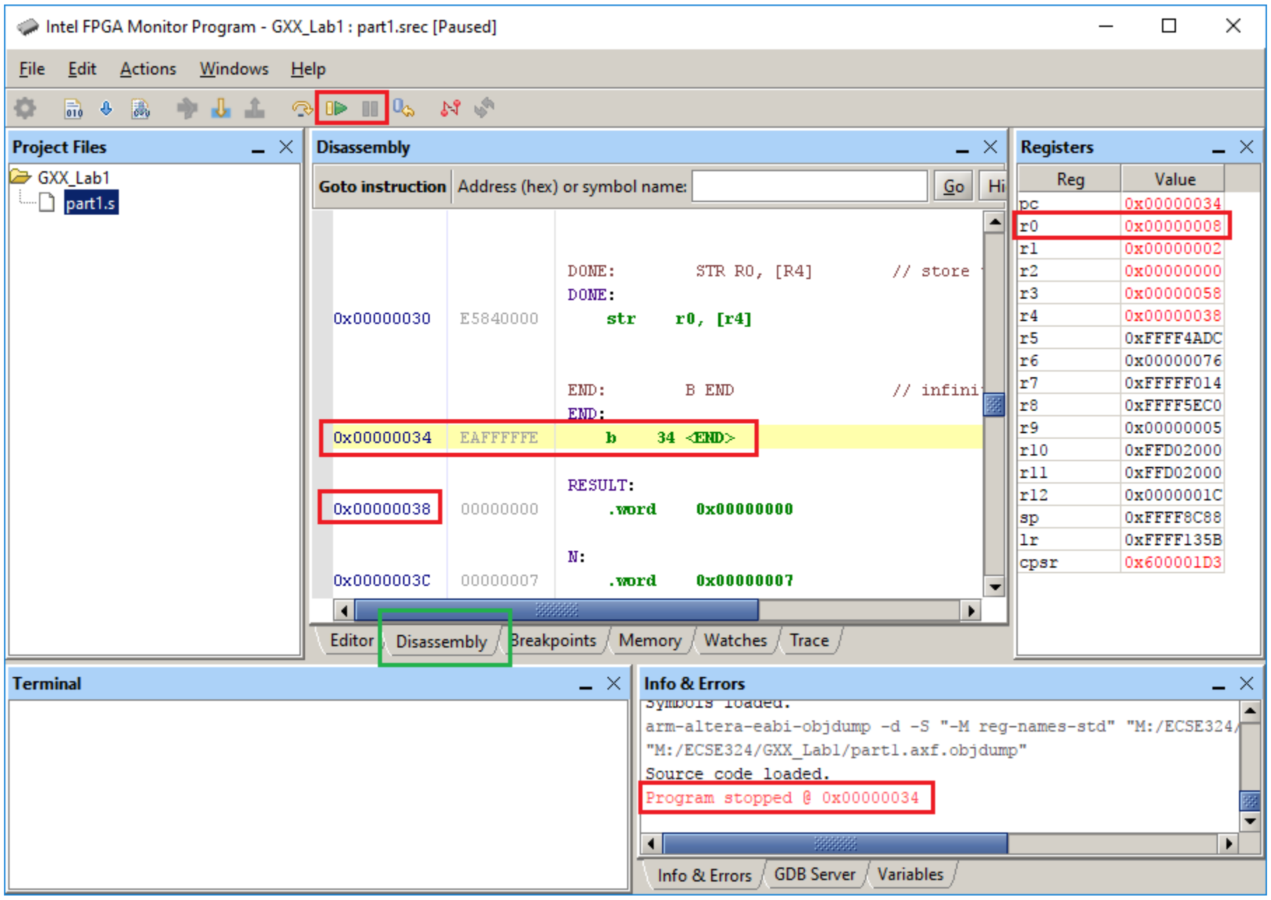
Figure 12: Using the IDE - Debug mode

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.



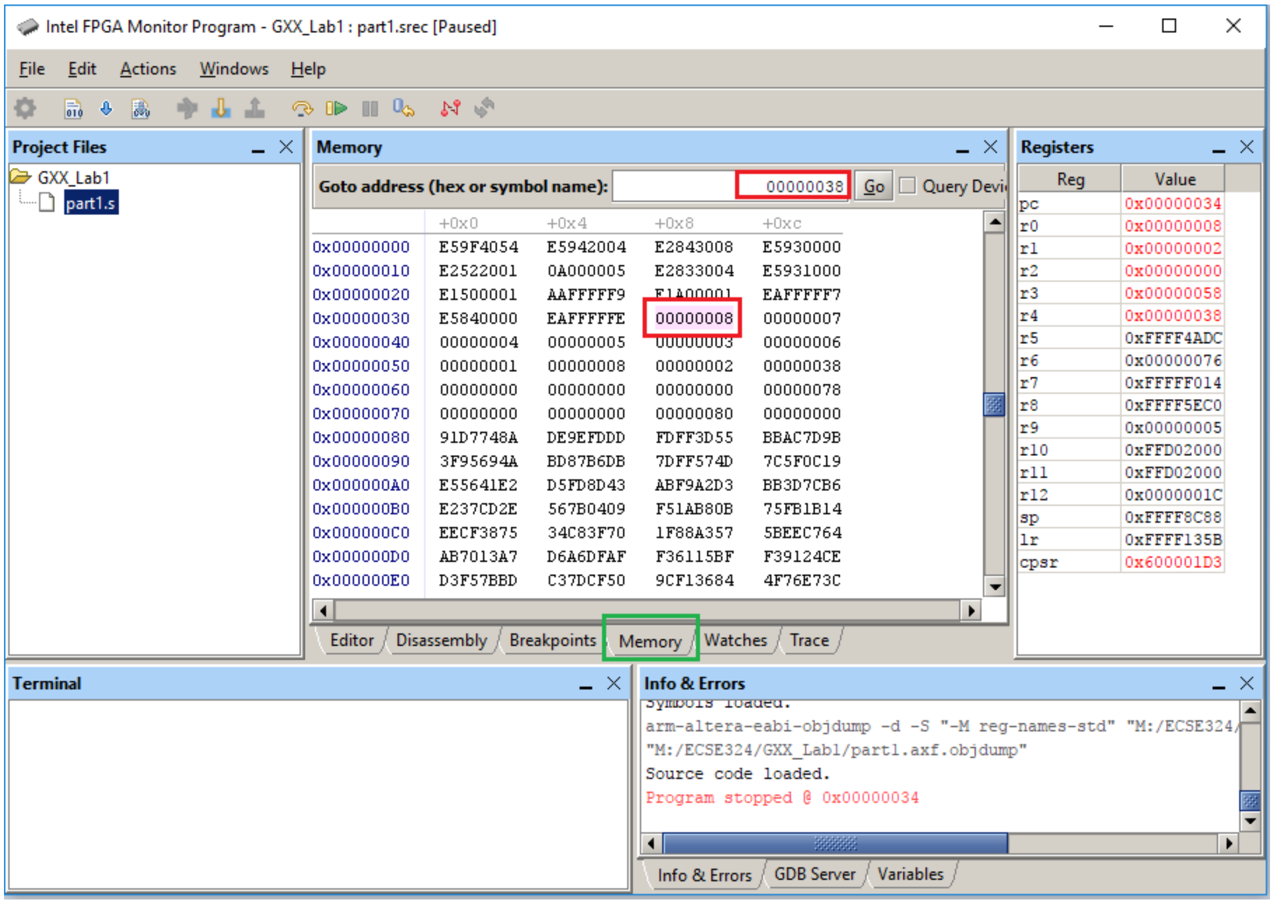
```
                                 DONE:        STR R0, [R4]        // store
                                 DONE:
0x00000030    E5840000              str     r0, [r4]

                                 END:        B END               // infini
                                 END:
0x00000034    EAFFFFFE              b      34 <END>

                                 RESULT:
0x00000038    00000000              .word      0x00000000

                                 N:
0x0000003C    00000007              .word      0x00000007
```

Reg	Value
pc	0x00000034
r0	0x00000008
r1	0x00000002
r2	0x00000000
r3	0x00000058
r4	0x00000038
r5	0xFFFF4ADC
r6	0x00000076
r7	0xFFFFF014
r8	0xFFFF5EC0
r9	0x00000005
r10	0xFFD02000
r11	0xFFD02000
r12	0x0000001C
sp	0xFFFF8C88
lr	0xFFFF135B
cpsr	0x600001D3

```
Symbols loaded.
arm-altera-eabi-objdump -d -S "-M reg-names-std" "M:/ECSE324/
"M:/ECSE324/GXX_Lab1/part1.axf.objdump"
Source code loaded.
Program stopped @ 0x00000034
```

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.


Intel FPGA Monitor Program - GXX_Lab1 : part1.srec [Paused]
File Edit Actions Windows Help
Project Files
GXX_Lab1
part1.s
Memory
Goto address (hex or symbol name): 00000038 Go
Query Devi
Editor Disassembly Breakpoints Memory Watches Trace
Registers
Terminal
Info & Errors
Info & Errors GDB Server Variables

```
              +0x0      +0x4      +0x8      +0xc
0x00000000  E59F4054  E5942004  E2843008  E5930000
0x00000010  E2522001  0A000005  E2833004  E5931000
0x00000020  E1500001  AAFFFFF9  E1A00001  EAFFFFF7
0x00000030  E5840000  EAFFFFFE  00000008  00000007
0x00000040  00000004  00000005  00000003  00000006
0x00000050  00000001  00000008  00000002  00000038
0x00000060  00000000  00000000  00000000  00000078
0x00000070  00000000  00000000  00000080  00000000
0x00000080  91D7748A  DE9EFDDD  FDFF3D55  BBAC7D9B
0x00000090  3F95694A  BD87B6DB  7DFF574D  7C5F0C19
0x000000A0  E55641E2  D5FD8D43  ABF9A2D3  BB3D7CB6
0x000000B0  E237CD2E  567B0409  F51AB80B  75FB1B14
0x000000C0  EECF3875  34C83F70  1F88A357  5BEEC764
0x000000D0  AB7013A7  D6A6DFAF  F36115BF  F39124CE
0x000000E0  D3F57BBD  C37DCF50  9CF13684  4F76E73C
```

Reg	Value
pc	0x00000034
r0	0x00000008
r1	0x00000002
r2	0x00000000
r3	0x00000058
r4	0x00000038
r5	0xFFFF4ADC
r6	0x00000076
r7	0xFFFFF014
r8	0xFFFF5EC0
r9	0x00000005
r10	0xFFD02000
r11	0xFFD02000
r12	0x0000001C
sp	0xFFFF8C88
lr	0xFFFF135B
cpsr	0x600001D3

```
Symbols loaded.
arm-altera-eabi-objdump -d -S "-M reg-names-std" "M:/ECSE324/
"M:/ECSE324/GXX_Lab1/part1.axf.objdump"
Source code loaded.
Program stopped @ 0x00000034
```

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, January 31st, **within your assigned lab period.** The report for Lab 1 is due by 11:59 pm, February 7th (if you submit your report or present your demo late, you will be given a zero!).

Please note that we will check every submission (code and report) for possible plagiarism. All suspected cases will be reported to the faculty.