# ET4162 LAB 4

This lab session introduces the ARM Development Studio (ARM DS). This is similar to the DS-5 software used last semester, but is a more advanced environment, and contains a compiler within the tool chain.

The ARM DS allows us to import an outline project, which reduces the tasks involved in setting up a project. We will import a project from a zipped archive file and modify the source files as required.

We will import a modified ‘Hello World’ C language program, which includes an ARMv8 Assembly Language module. This module contains a short program to add numbers using two of the ARM’s internal registers. The assembly language program is called by a modified version of ‘Hello World’ and returns a result to be printed using the calling C program’s *printf* function.

We will compile the program with its two modules and use the ARM-DS Debugger to see how the assembly program is executed and how it uses the CPU registers for arithmetic operations.

## Before you start

Download the file ‘Lab1Test.zip’ from Sulis and store it in a folder on your computer. Note the location where you have saved the file. Do NOT click on or open the file. We will extract the file from within ARM DS later.

## Download or use the ARM-DS environment

2.1 Download the ARM DS software

**(ONLY NECESSARY IF YOU ARE WORKING FROM HOME)**

If you have not already done so, download ARM DS from the following location:

<https://developer.arm.com/tools-and-software/embedded/arm-development-studio/downloads>

You will need to register and log in, but this is free. UL has licences for 110 seats for the ARM DS Gold Edition, so you may use that version, but you must follow the instructions in the document ‘Using ARM DS from home’.

Please refer to the document ‘Arm® Development Studio Getting Started Guide’ on Sulis. Chapter 2 describes how to download the ARM DS software. The document ‘Using ARM DS from home, guides you through the software licensing steps.

* 1. Using the ARM DS software from UL ECE labs

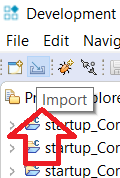
UL has licences for 110 seats for the ARM DS Gold Edition, so you may use that version, but you must follow the instructions in the document ‘Using ARM DS from UL ECE Labs’.

## Import a modified ‘Hello World’ C project

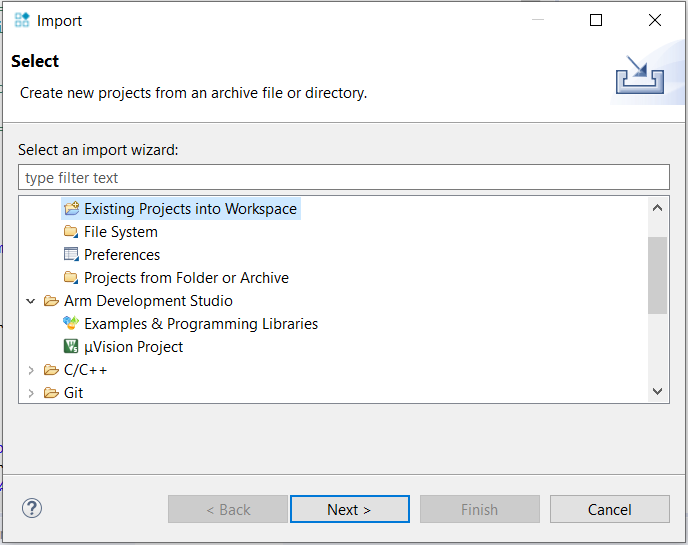
When you open ARM-DS, ensure you are in the Development Studio Perspective. This should be the default, but if you are not in it, click on the Development Studio button on the top right hand corner.



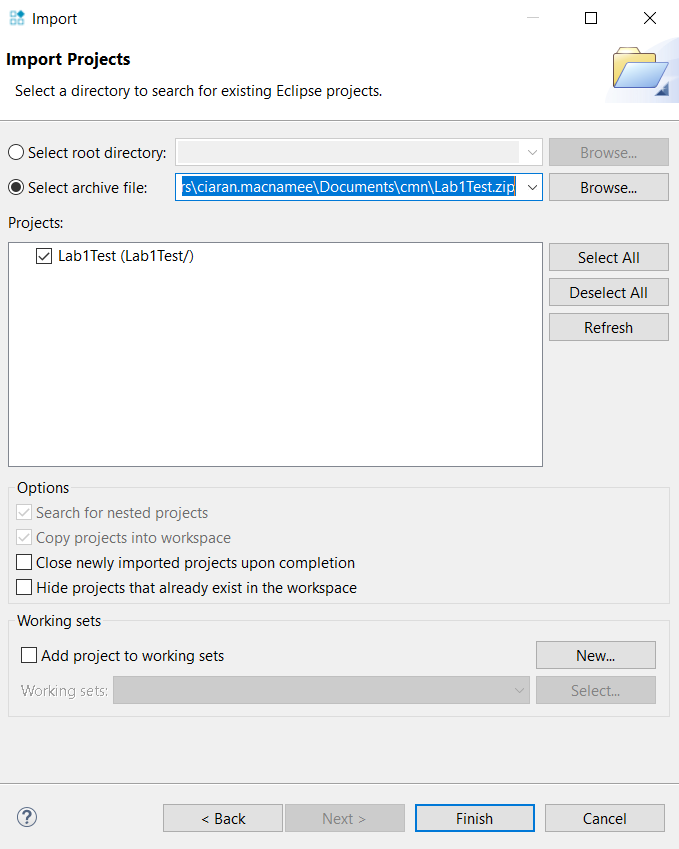
From the ARM-DS main menu, select the File Import Icon (below)



or select **File > Import** to display the Import Wizard. In the Wizard, select General -> Existing Projects into Workspace as shown below:



Select Next to bring up the Import Projects Screen

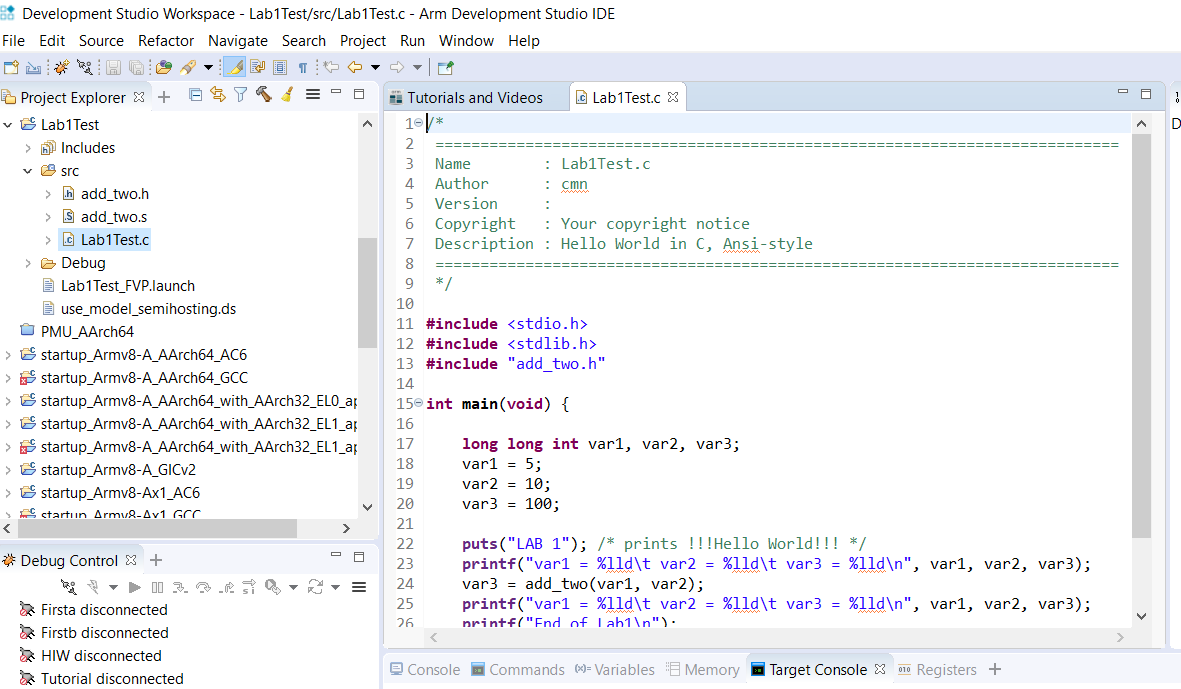


Click Select archive file and browse to the folder where you stored Lab1Test.zip in Step 1. I stored it in Documents\cmn\ as you will see above.

When you have selected this file, click Finish to Import the project into ARM DS.

## Open Lab1Test in Project Explorer

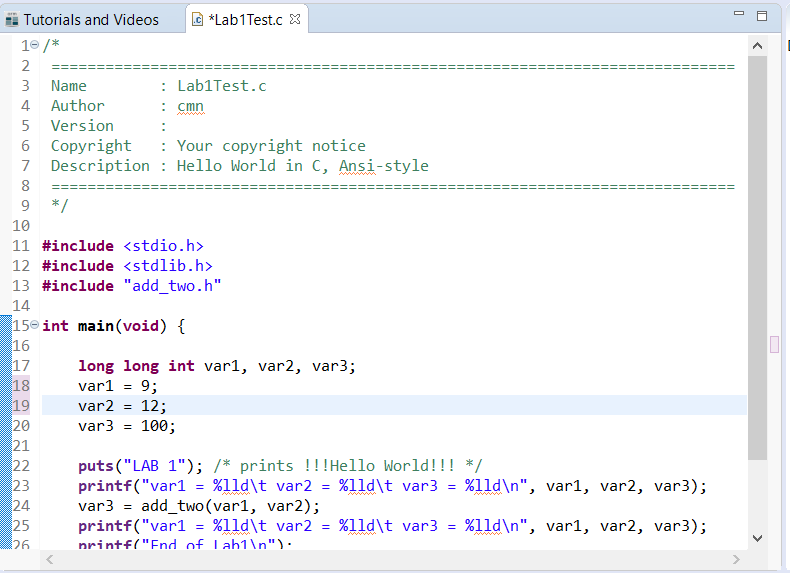
In Project Explorer Click on Lab1Test and expand it until you see the src folder. Click on Lab1Test.c to bring the C source file into the Editor Window.



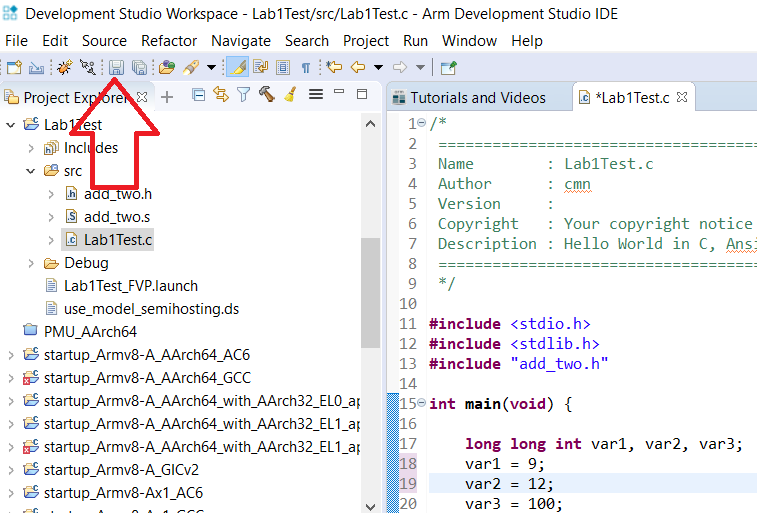
(Your Project Explorer will have fewer projects, since it is a new install).

## Modify the source file and rebuild the project

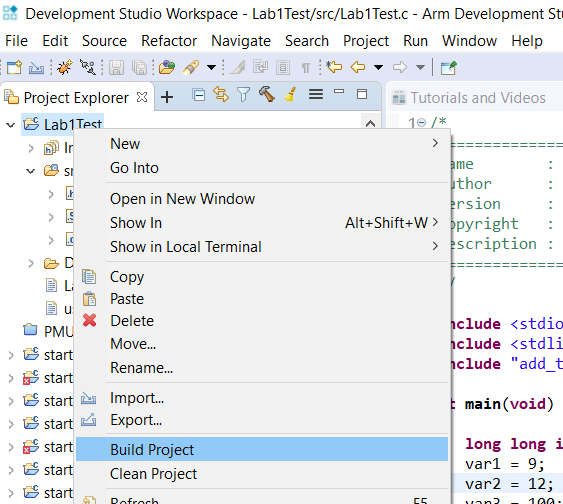
In the source editor, change the settings of var1 and var2 to 9 and 12 as below:



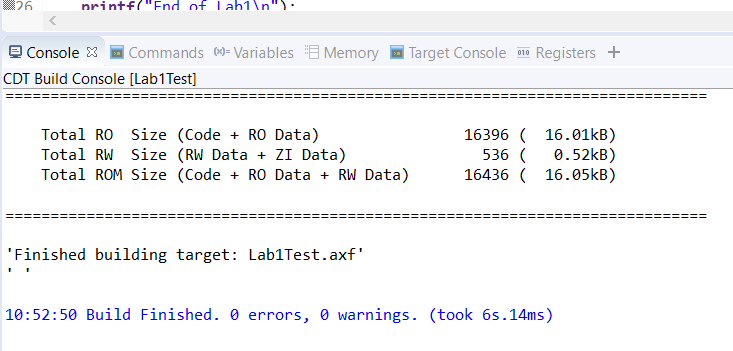
Note that the \* beside the file name indicates that the file has not been saved on disk. Save the file using **Ctrl-S** or with the save icon in the Save File icon in the menu.



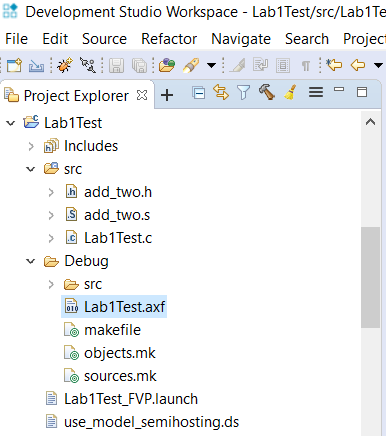
Now Build your project, to see that the tools are working. Right Click on the Lab1Test Project (not the C source file) and select the Build Project option.



You should see a message in the Console View like the one below (times will be different).



You should also see a file called ‘Lab1Test.axf’ in the Debug folder.



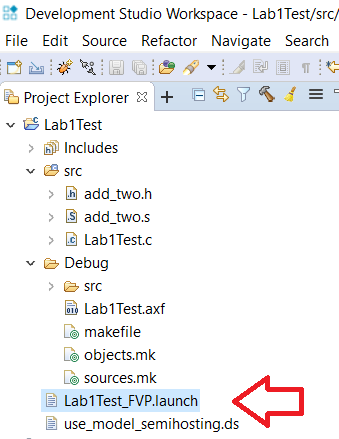
## Explanation of the program

We cover how programs can pass parameters in registers to functions or subroutines in lectures. For now we note that the integers to be added in the assembly language program are passed in the registers X0 and X1. X0 gets the value of the first register in the list and X1 gets the second one.

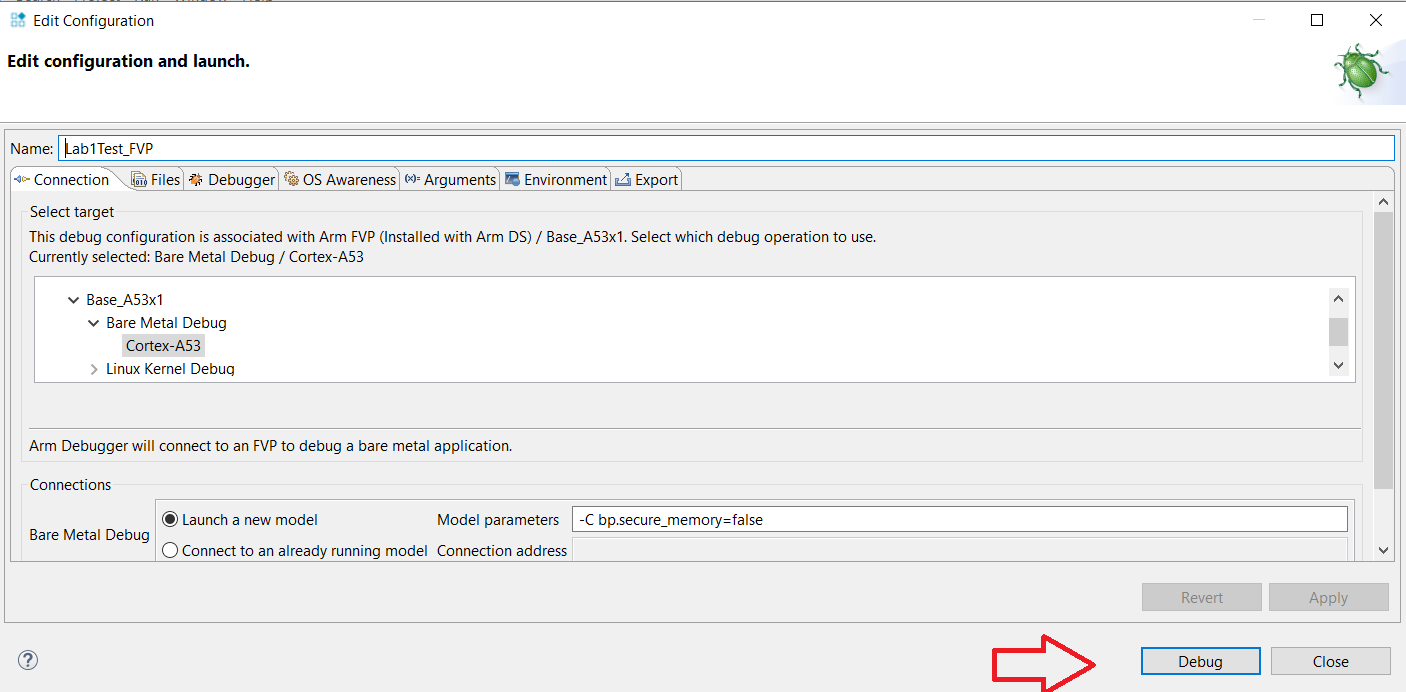
When the add\_two function returns to the code that called it, the calling code expects to get the returned value in the X0 register. The instruction add (X0, X0, X1), does exactly what is required: the data in X0 and X1 are added together and the result is stored in X0. So, all that is left to do is to return to the called program, which is done using the BR instruction. As we will see, the return destination has been stored in X30 (the Link Register) by the calling instruction, so X30 gets the code back to where it came from.

## Use the Debug tool to view your program’s operation

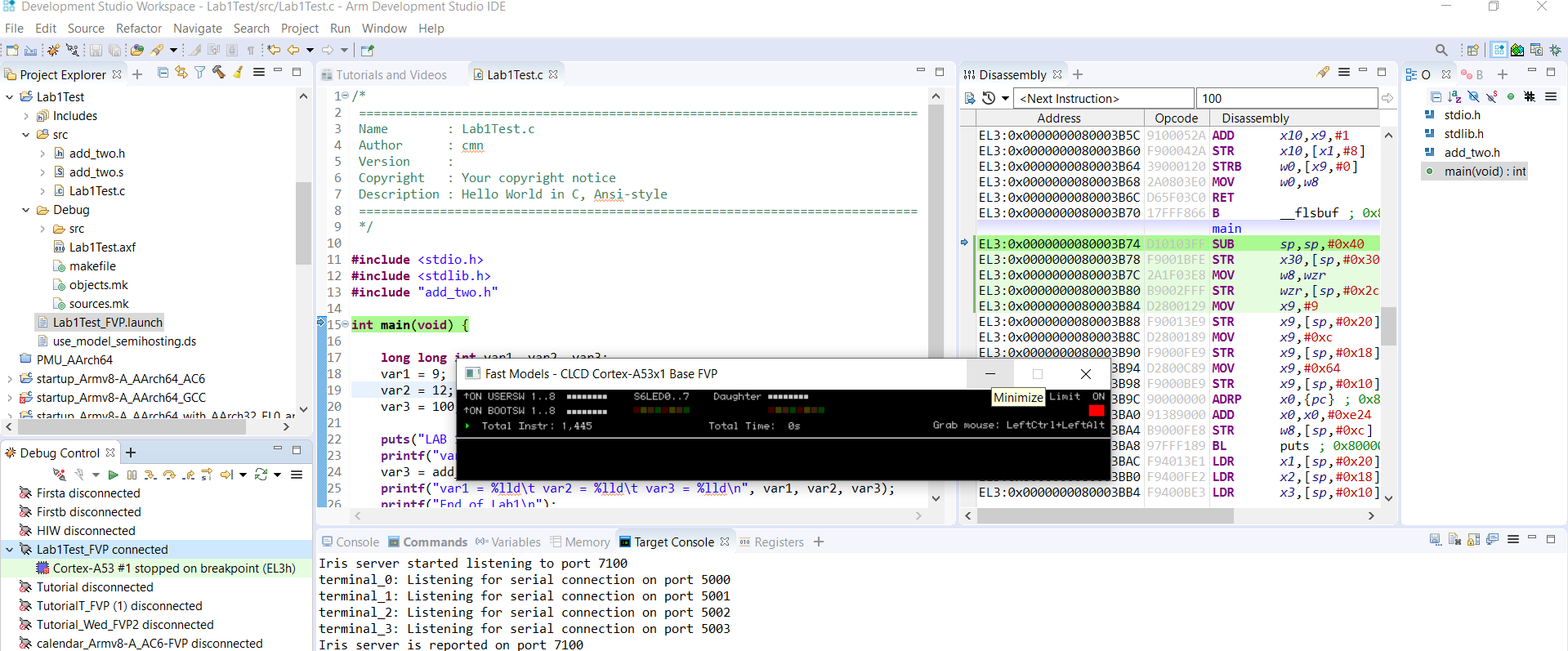
In the Project Explorer, select Lab1Test\_FVP.launch, see below.



This brings up the Edit Configuration Screen. This should be set up correctly from the Project Import so select Debug to continue.



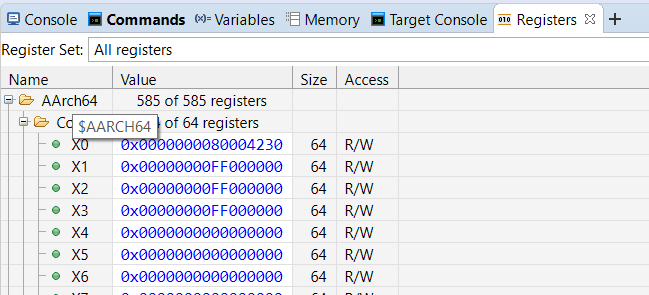
This invokes the debugger and after a few seconds you should see a screen similar to the following one.



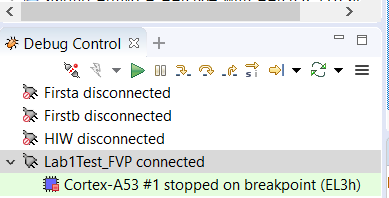
(You can minimise the mouse simulator (black background window) since we don’t use it).

If they are not already present in your screen set up, use the Window -> View Menu drop downs to bring up the Disassembly and Registers Windows (seen in the screen above).

Select the registers select AARCH64->Core:



The debug control view is in a different default position in ARM DS than DS-5 but it works in the same way.



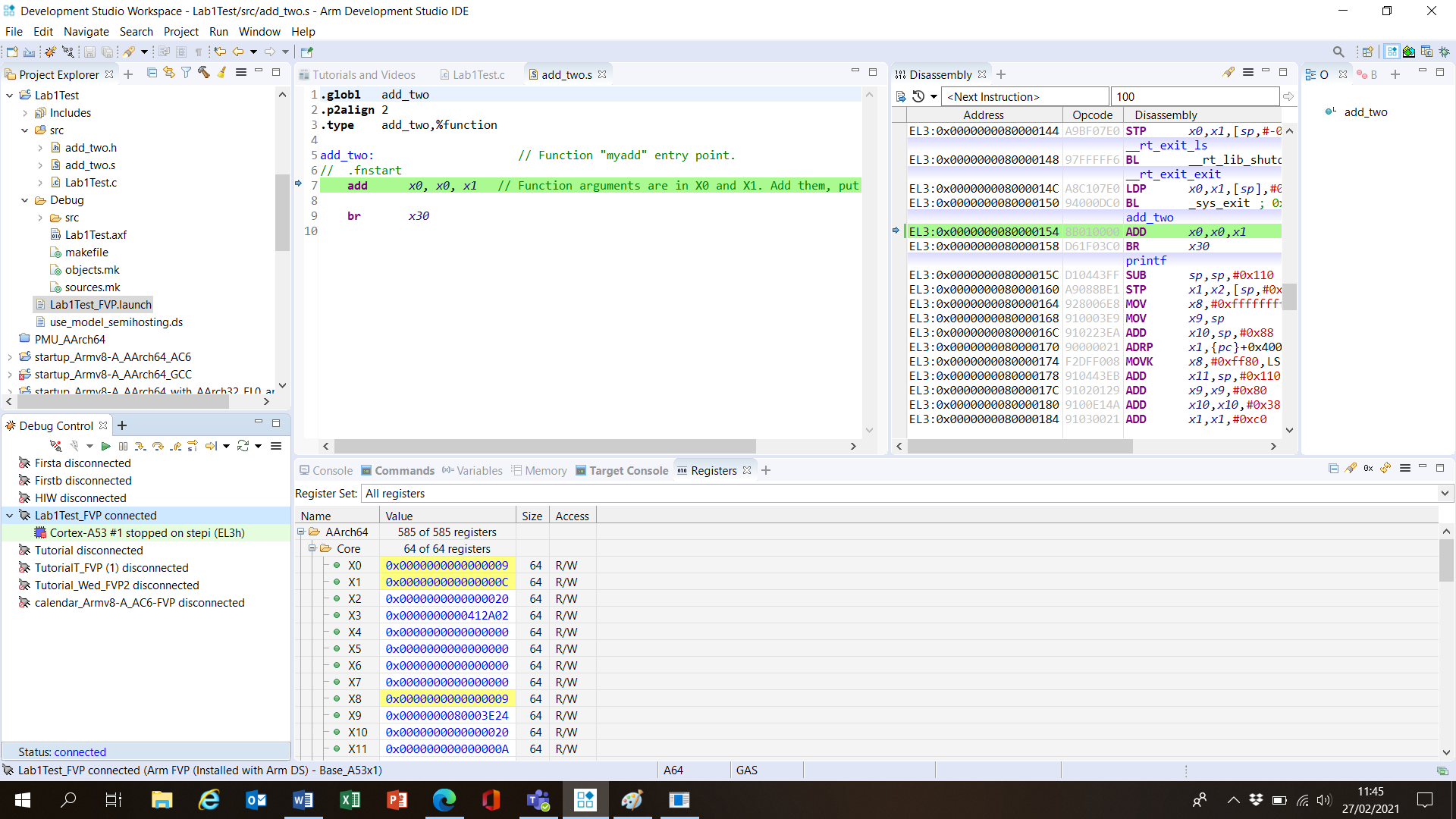


Now step through your program using the Step Over icon until you get to the line

var3 = add\_two(var1, var2);



Use the Step Into icon to step into the assembly language add function. You should see the following screen:



Note the values in X0 and X1 and fill them in the table below. Also note the value in X30 or LR.

**Exercise Set 1.**

Fill in the values in the tables below. Fill in the values and upload your file to Sulis.

Note the values contained in X0 and X1. Are these values what you expect?

|  |
| --- |
| X0 (before add) = 0x0000000000000009 |
| X1 (before add) =0x000000000000000C |

Also note the value in LR (X30). Can you find what instruction the data in LR will return to?

|  |
| --- |
| LR (X30) = 0x0000000080003C34 |

Step over the add instruction and note the new values in the X0 and X1 registers. Can you explain what has happened here?

|  |
| --- |
| X0 (after add) =0x0000000080003E60 |
| X1 (after add) = 0x00000000FFFEFCE0 |

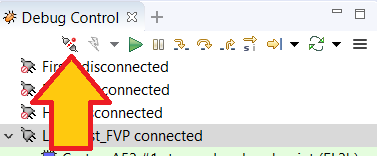
Continue stepping (Step Over icon) and note the return to the calling program.

Select the Variables View to see the state of the variables var1, var2, and var3.

|  |
| --- |
| var1= 9 |
| var2=12 |
| var3=21 |

If you have time, continue to Exercise Set 2 below. Otherwise finish and close ARM DS.

**IMPORTANT: ALWAYS DISCONNECT THE DEBUGGER BEORE QUITTING THE ARM DS SOFTWARE.**



Disconnect Debugger Connection icon.

**Exercise Set 2 (if time allows)**

Disconnect the debugger and close the project. Modify Lab1Test.c as shown below (you can copy and paste the text below, if you wish). The C file will call the assembly code add\_two with four parameters, var1, var2, var3 and control\_var. It’s given below:

**#include** <stdio.h>

**#include** <stdlib.h>

**#include** "add\_two.h"

**int** **main**(**void**) {

**long** **long** **int** var1, var2, var3, var4, control\_var;

var1 = 5;

var2 = 10;

var3 = 6;

control\_var = 1;

**puts**("LAB 1"); /\* prints !!!Hello World!!! \*/

**printf**("var1 = %lld\t var2 = %lld\t var3 = %lld\n", var1, var2, var3);

var4 = add\_two(var1, var2, var3, control\_var);

**printf**("var1 = %lld\t var2 = %lld\t var3 = %lld\t var4 = %lld\n", var1, var2, var3, var4);

**printf**("End of Lab1\n");

**return** EXIT\_SUCCESS;

}

Now modify the header file add\_two.h as follows (again you may copy and paste):

**##ifndef** ADD\_TWO\_H\_

**#define** ADD\_TWO\_H\_

**extern** **long** **long** **int** **add\_two**(**long** **long** **int** A, **long** **long** **int** B, **long** **long** **int** C, **long** **long** **int** control);

**#endif** /\* ADD\_TWO\_H\_ \*/

Finally modify add\_two.s so that it uses the state of ‘control’ to change the way the add\_two operates. The assembly file should implement the following:

if (control == 0)

f = A + (B + C);

else

f = A + (B-C);

return f;

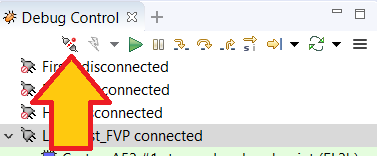
Use X9 as a temporary value but don’t try to save it on the stack. The shell of the assembly file is:

Note that the returned data should be returned in register X0.

Save all your files, and rebuild your project using the steps we followed earlier. Verify its operation using the Debug tool. If you’re happy with the results, submit your modified \*.c, \*.s. and \*.h files on Sulis. Remember to disconnect the debug tool before closing ARM DS.

Finish and close ARM DS.

**IMPORTANT: ALWAYS DISCONNECT THE DEBUGGER BEORE QUITTING THE ARM DS SOFTWARE.**



Disconnect Debugger Connection icon.