

ESET 349 Microcontroller Architecture

Lab 1: Understanding the MSP432 architecture – at CPU register and memory level using debugging tools

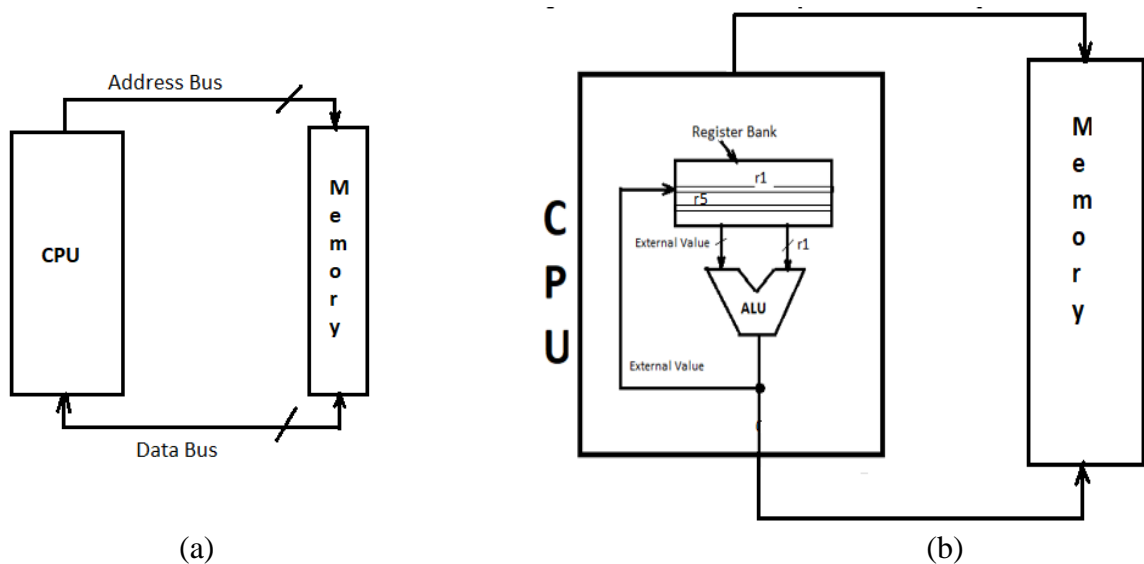


Figure 1. (a) A simple CPU-Memory Architecture. (b) Architecture used in lab 1.

Objective

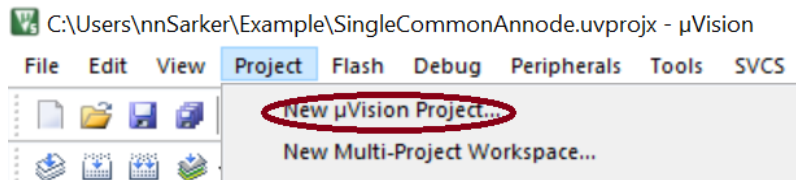
The objective of this lab is to understand:

- Addition and subtraction of numbers at register level
- Storing (moving) data from register to a specific memory location.
- Loading (moving) data from a specific location in the memory to a specific register.

This handout describes the following: **1.** Setting up a project on Keil, **2.** The program to use for this lab, **3.** Using the Keil debugger and **4.** The required tasks to be completed. You will use the Keil Simulator for completing the experiment and do not require an MSP432.

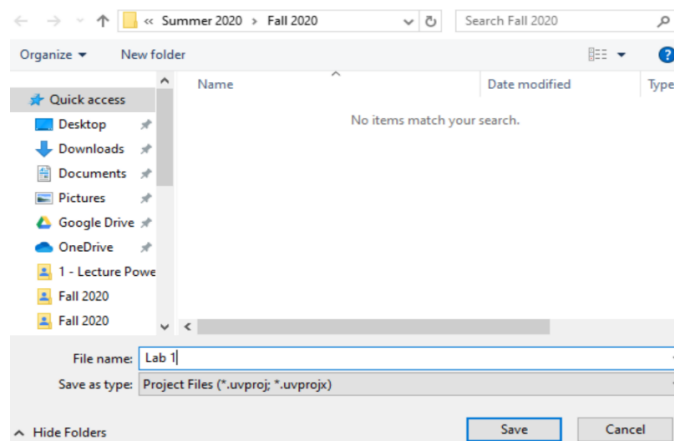
Keil Project Setup

Step 1: Create a new project.

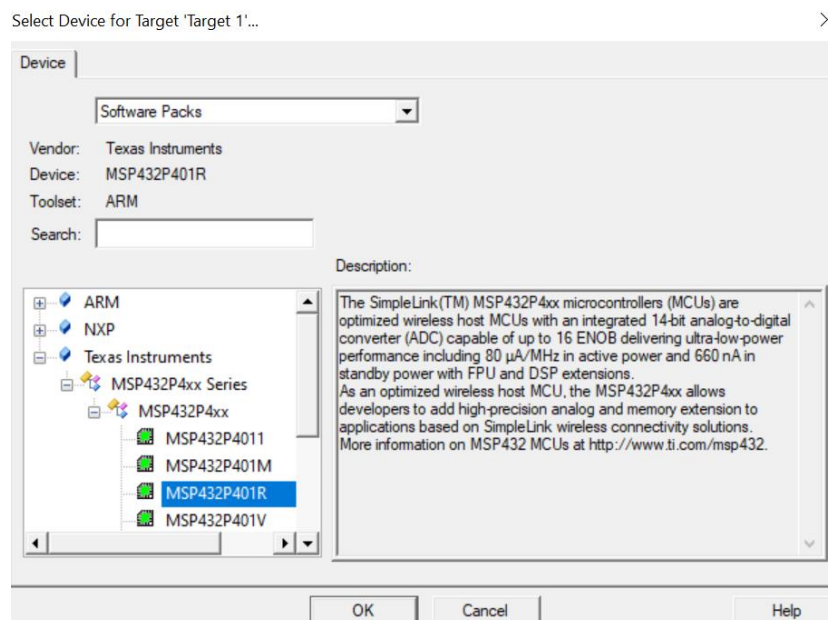


Note: Create project for Lab1. Name the assembly language file as “lab1.s”.

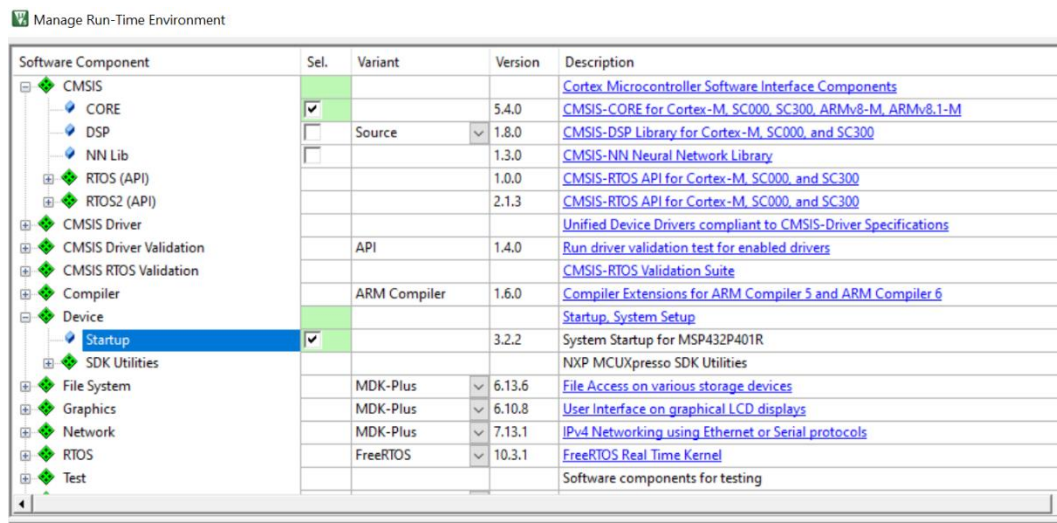
Name the project as “Lab 1” and save at your desired location.



Select MSP432P401R as the target device.



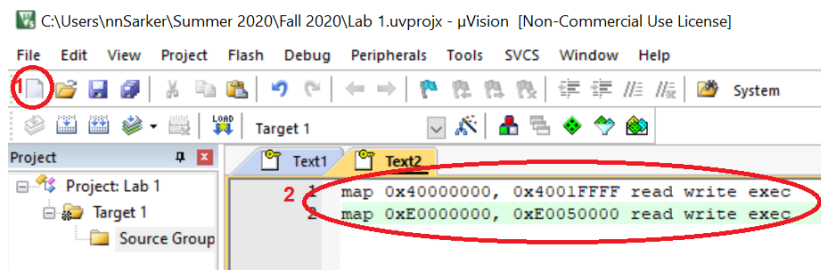
Select the following options shown below.



Step 2

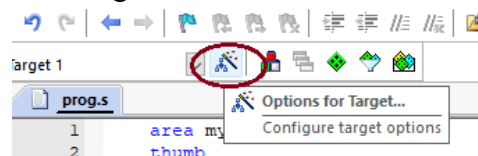
Create a new file (icon shown in 1 in the snapshot below).

Add the two lines (shown in 2) and save the file as 'map.ini'.

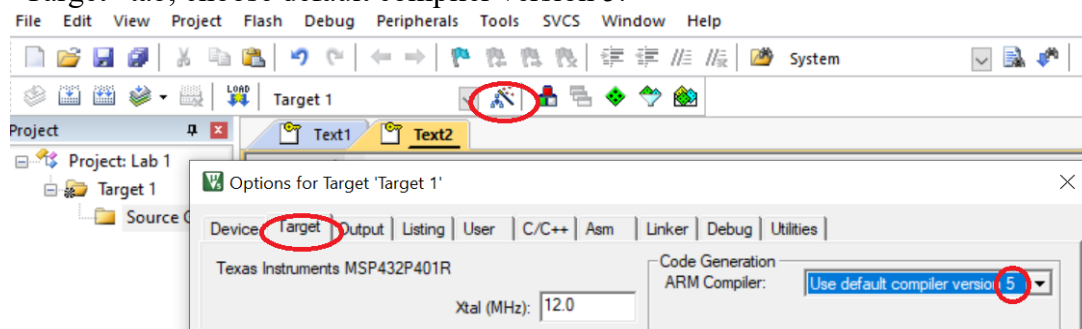


Step 3:

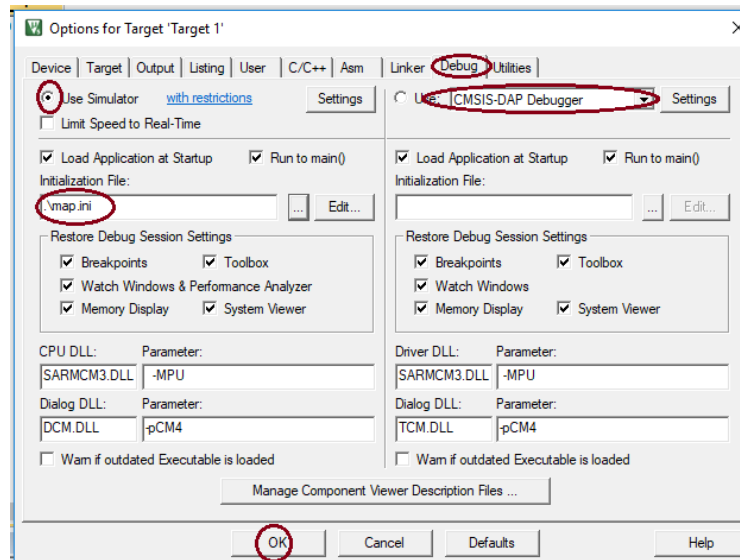
Click on the Icon for "Options for Target" as shown below and follow the screenshots.



In the "Target" tab, choose default compiler version 5.

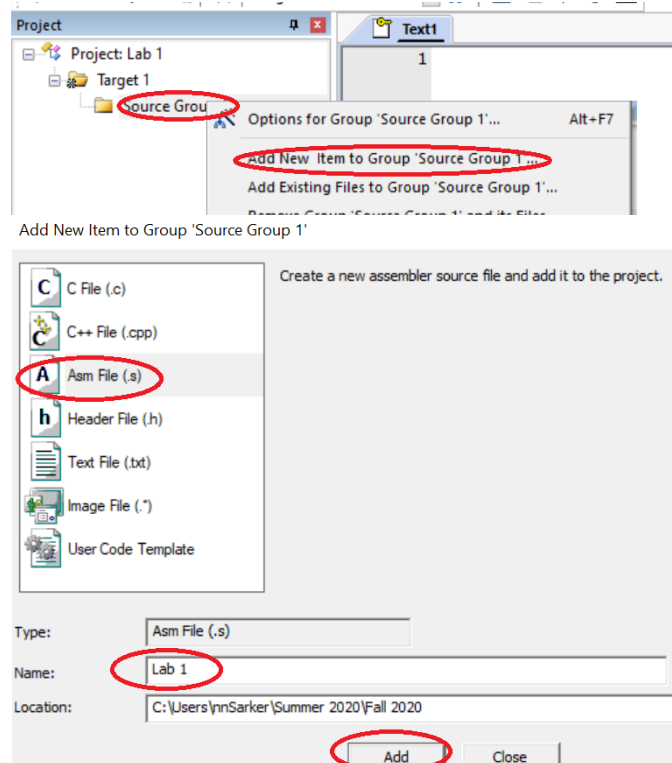


Select the “Debug” tab first and select the boxes as shown with ellipses. For the '\map.ini' file, click on the three-dot button at the left of the Edit button and select the **map** file previously created in Step 2 using the file explorer pop-up. Finally click the OK button. **In later labs, use the CMSIS-DAP Debugger on the right instead to load code onto the MSP432.**



Step 4:

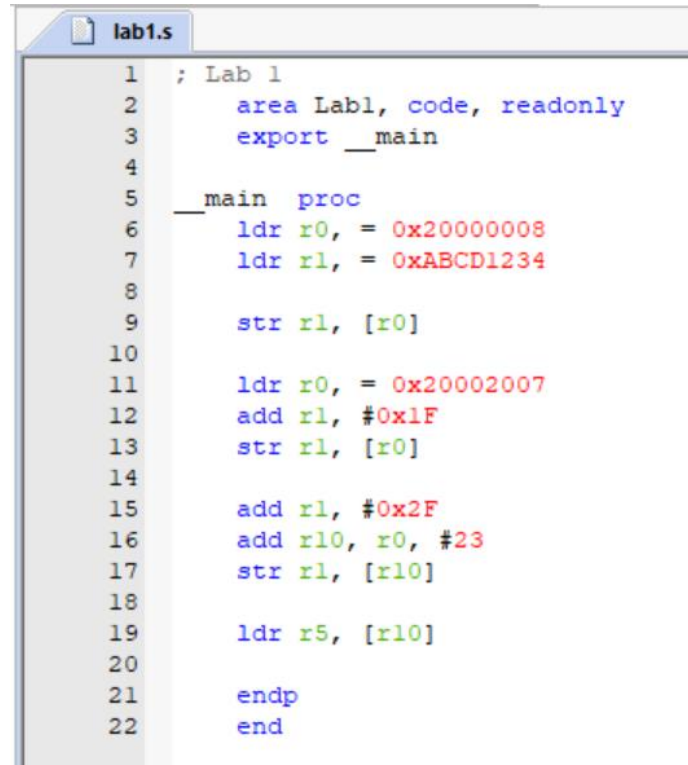
Add a .s file to the project you created.



Lab Program

Step 1:

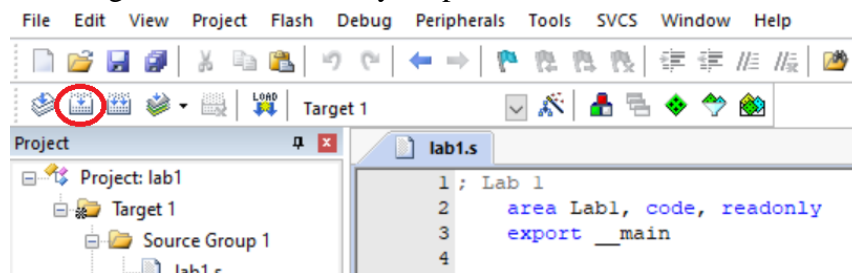
Use the following program in your Lab 1.



```
lab1.s
1 ; Lab 1
2     area Lab1, code, readonly
3     export __main
4
5     __main proc
6         ldr r0, = 0x20000008
7         ldr r1, = 0xABCD1234
8
9         str r1, [r0]
10
11        ldr r0, = 0x20002007
12        add r1, #0x1F
13        str r1, [r0]
14
15        add r1, #0x2F
16        add r10, r0, #23
17        str r1, [r10]
18
19        ldr r5, [r10]
20
21    endp
22    end
```

Step 2:

Build the program using the button shown by ellipse.



Debugging

Now it is time to debug the program one instruction at a time.

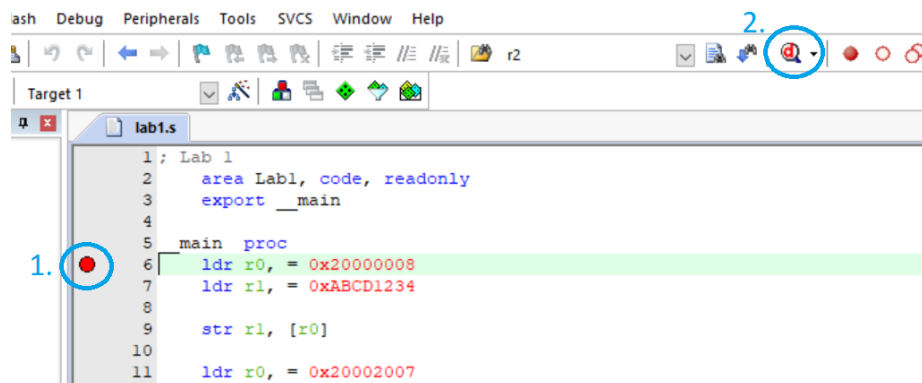
Setting Breakpoints

Notice the red dot labeled as 1 in the snapshot below. These are called breakpoints. Breakpoints allow you to pause the execution of a program and inspect the status of the environment, including registers and memory. If no breakpoint was present, the debugger would continue executing the program without a pause for inspection.

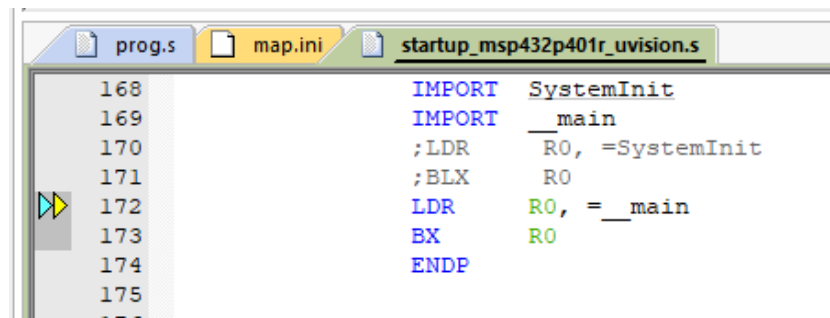
Put your cursor in these lines one at a time and click on the solid red ball. Then to start debugging, click on the button marked with 'd' inside. Click 'OK' in the "EVALUATION MODE" popup.

Set a breakpoint in front of the first "LDR" statement (left click in the greyed region shown in the snapshot below).

Press "Ctrl + F5" or the "Start/Stop Debug Session" (the red 'd' button.)



Speeding up debugging (recommended)



Comment out the lines shown above (as 170 and 171) in the startup_msp432p401r_uvision.s file. Repeat the steps for all projects using the Keil debugger but **make sure to uncomment the lines while uploading programs to MSP432 in the later labs.**

Complete executing startup code

Press “F11” until the startup instructions execute, and the debugger reaches the program file above. In the program pane, you will see the lab1.s file in context and the “LDR” statement will be highlighted in yellow.

Execute instructions

Line 6 (you may have a different line number) shows the first statement of your code. The yellow line in the disassembly pane shows the detailed information about your statement. It has three sections. The first one is the 32-bit (shown in hex) memory address of the instruction, the second one is the actual instruction for the machine (machine instruction) in 32-bit (also shown in hex) and the last one is the instruction you have typed. These sections are labeled in Figure 2 below.

The line of code to be executed next is pointed by the yellow arrow in the disassembly pane as well as the yellow triangle in the program pane.

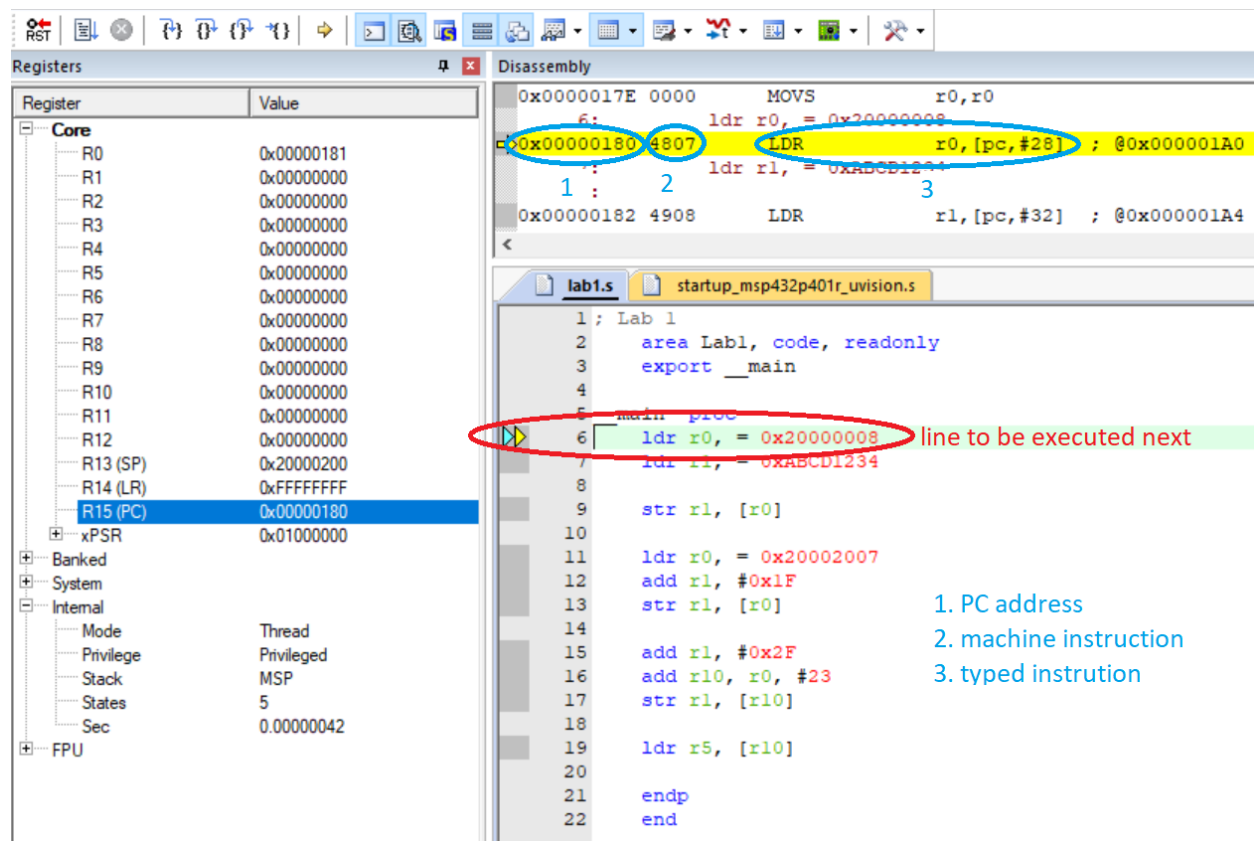


Figure 2. Breakdown of dissassembly

Notice the left-hand pane called “Registers”. The left column shows the core register names and the right column shows values in hex number format. Right now, R13, R14, R15 and xPSR have some values.

Press ‘F11’ and observe the outcomes of the execution of the current instruction; and note the changes in the register values for every press of F11.

Continue Debugging and record data to complete the table below. As you continue, look for the values in the registers and in the memory. Open the “Memory 1” pane. Type 0x20000000 in the Address box.

Look at Figure 3 below to understand the data movement to memory.

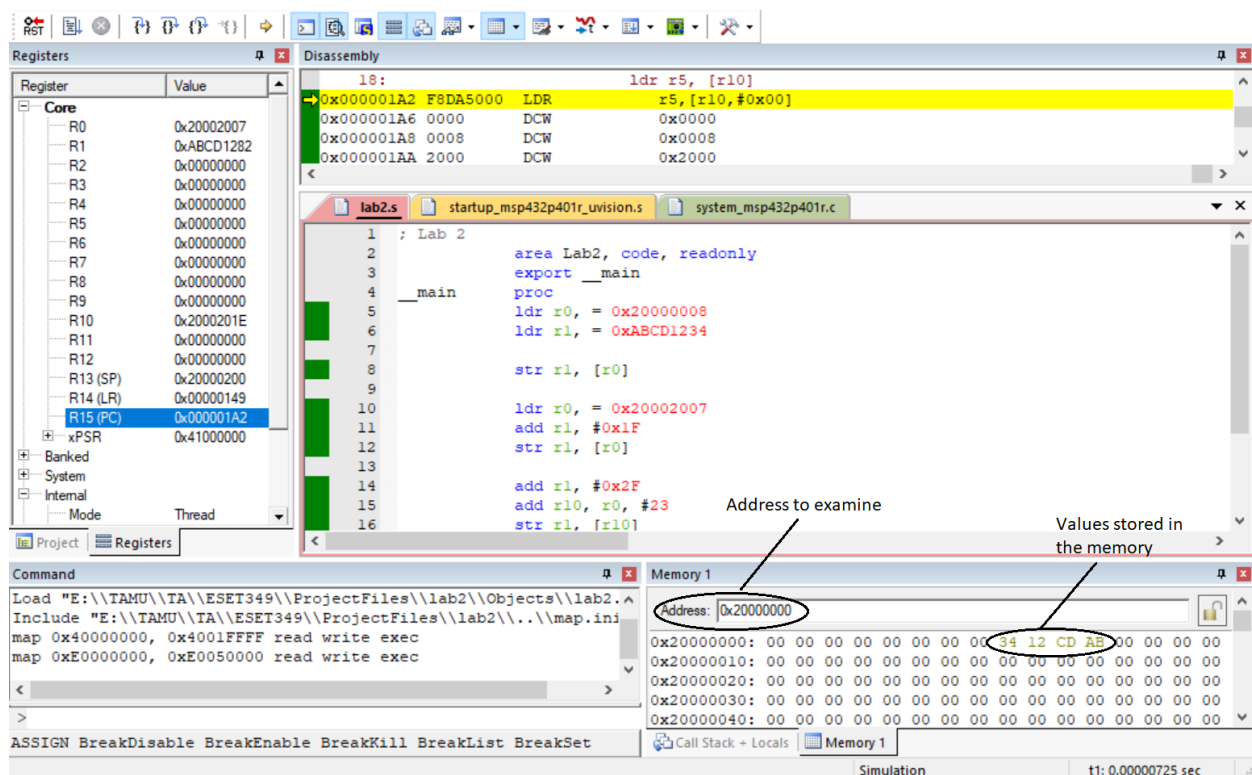


Figure 3. Examining the values stored in memory using the Keil debugger.

The lab requires you to **complete Tables 1 and 2** as shown below.

Complete the following Table 1.

Table 1

F11 count	R15 (PC)	To be executed			Register values in hex after each F11			
		My statement	Instruction memory address, hex	Instruction, hex	R0	R1	R5	R10
Not pressed								
1								

Add lines as needed in your table. Make sure to write the appropriate register names as given at the beginning of this lab. If you have a situation that the last statement, `ldr r5, [r10]`, does not get executed with F11, add a dummy statement, such as `mov r7, #0`, right before the `endp` command.

Find out the difference between the first, second, and the third memory locations used in this program.

Complete the following Table 2.

Table 2

- a) Show a memory map for the first location. Each location has room for one byte

Memory address	Bytes in memory
0x20000008	0x34

b) Show a memory map for the second location. Each location has room for one byte

Memory address	Bytes in memory

c) Show a memory map for the third location. Each location has room for one byte

Memory address	Bytes in memory

Note: At checkout, show all your activities to your TA.