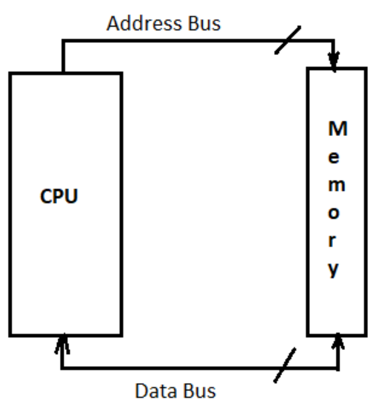
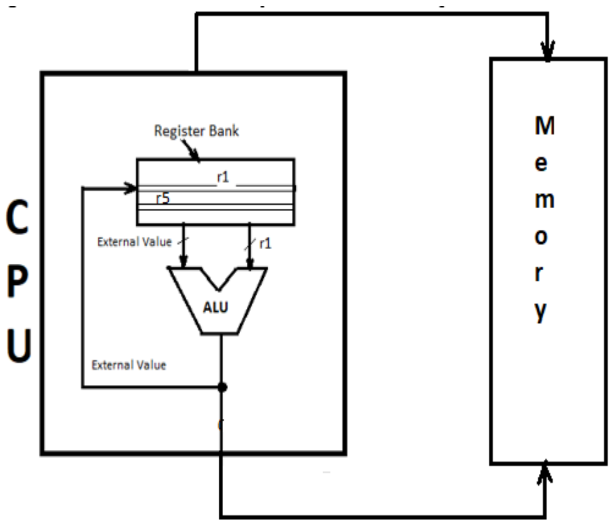
Department of Engineering Technology and Industrial Distribution

ESET 349 Microcontroller Architecture

Lab 1: Understanding the STM32F4 architecture – at CPU register and memory level using debugging tools



(a)



(b)

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

Objective

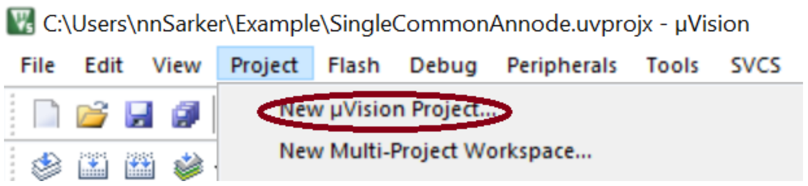
The objective of this lab is to understand:

a) Addition and subtraction of numbers at register level
b) Storing (moving) data from register to a specific memory location.
c) 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 STM32F401RE board.

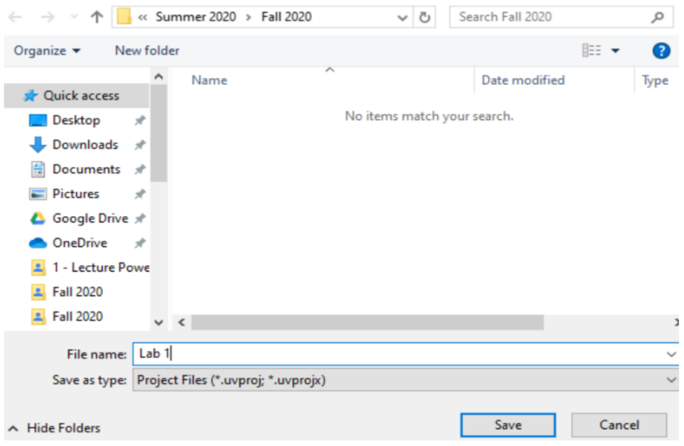
Keil Project Setup

Step 1: Create a new project.


C:\Users\nnSarker\Example\SingleCommonAnnode.uvprojx - µVision
File
Edit
View
Project
Flash
Debug
Peripherals
Tools
SVCS
New µVision Project...
New Multi-Project Workspace...

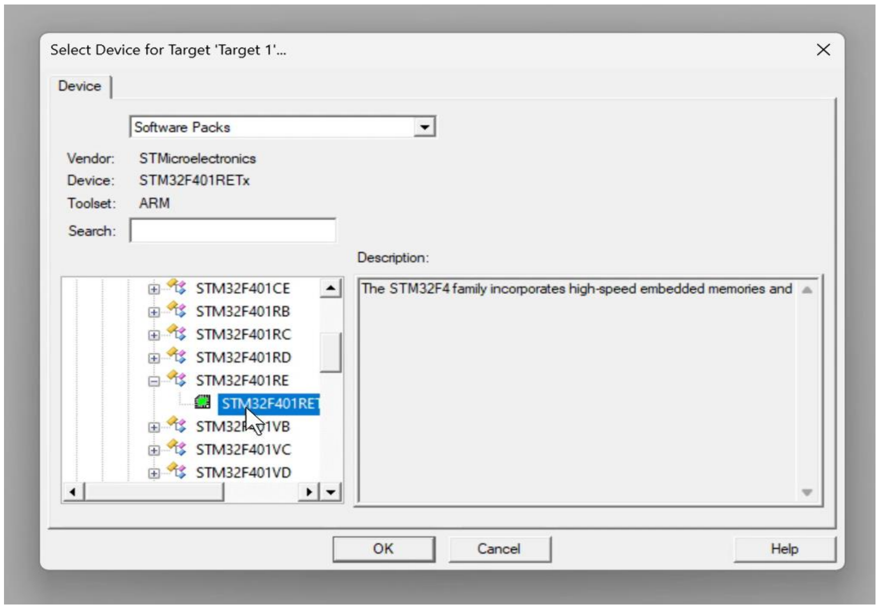
Note: Create project for Lab1. Name the assembly language file as "lab1.s".

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



File name: Lab 1
Save as type: Project Files (*.uvproj; *.uvprojx)

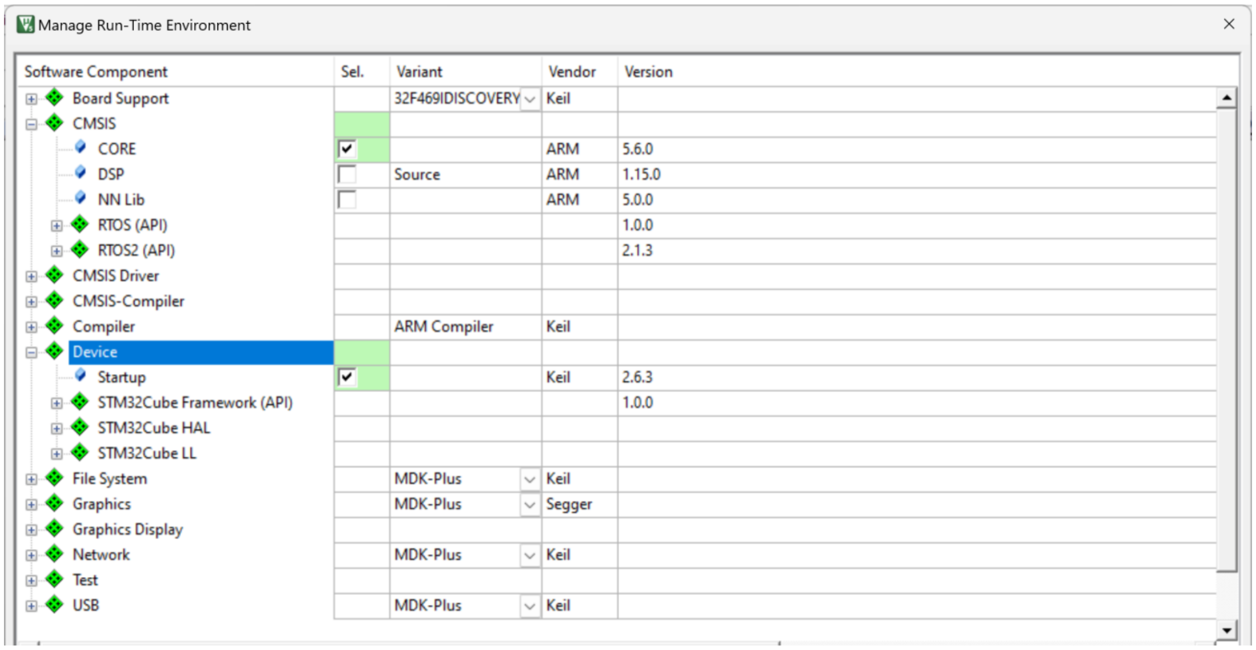
Select STM32F401RE as the target device.



Vendor: STMicroelectronics
Device: STM32F401RETx
Toolset: ARM
Search:

The STM32F4 family incorporates high-speed embedded memories and

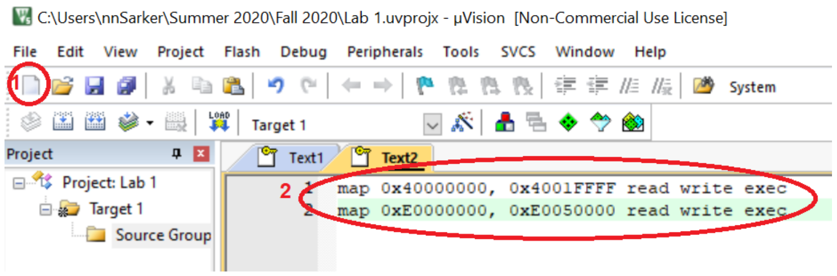
Select the following options shown below.



Software Component	Sel.	Variant	Vendor	Version
Board Support		32F469IDISCOVERY	Keil	
CMSIS				
CORE	✓		ARM	5.6.0
DSP	☐	Source	ARM	1.15.0
NN Lib	☐		ARM	5.0.0
RTOS (API)				1.0.0
RTOS2 (API)				2.1.3
CMSIS Driver				
CMSIS-Compiler				
Compiler		ARM Compiler	Keil	
Device				
Startup	✓		Keil	2.6.3
STM32Cube Framework (API)				1.0.0
STM32Cube HAL				
STM32Cube LL				
File System		MDK-Plus	Keil	
Graphics		MDK-Plus	Segger	
Graphics Display				
Network		MDK-Plus	Keil	
Test				
USB		MDK-Plus	Keil	

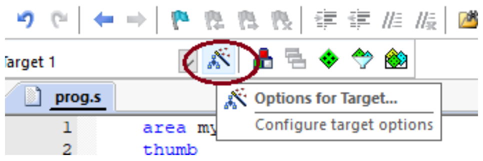
Step 2
Create a new file (icon shown in 1 in the snapshot below).

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

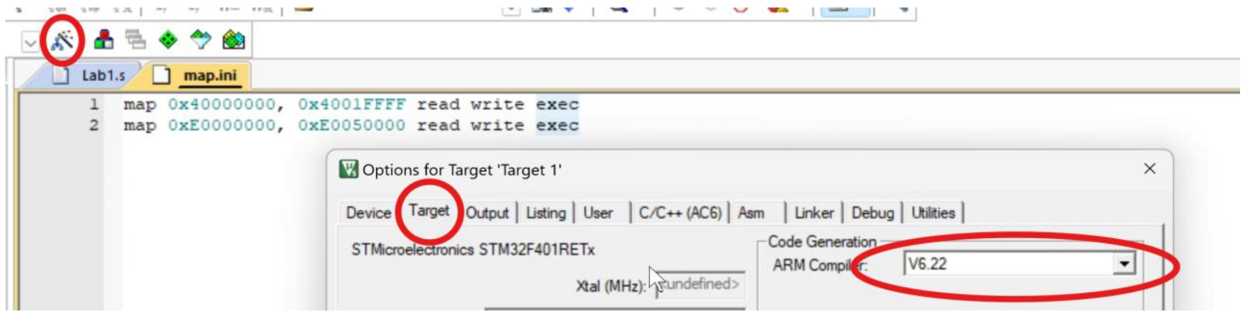

C:\Users\nnSarker\Summer 2020\Fall 2020\Lab 1.uvprojx - µVision [Non-Commercial Use License]
File Edit View Project Flash Debug Peripherals Tools SVCS Window Help
System
Target 1
Project
Project: Lab 1
Target 1
Source Group
Text1
Text2
1
2

```
map 0x40000000, 0x4001FFFF read write exec
map 0xE0000000, 0xE0050000 read write exec
```

Step 3:
Click on the Icon for “Options for Target” as shown below and follow the screenshots.



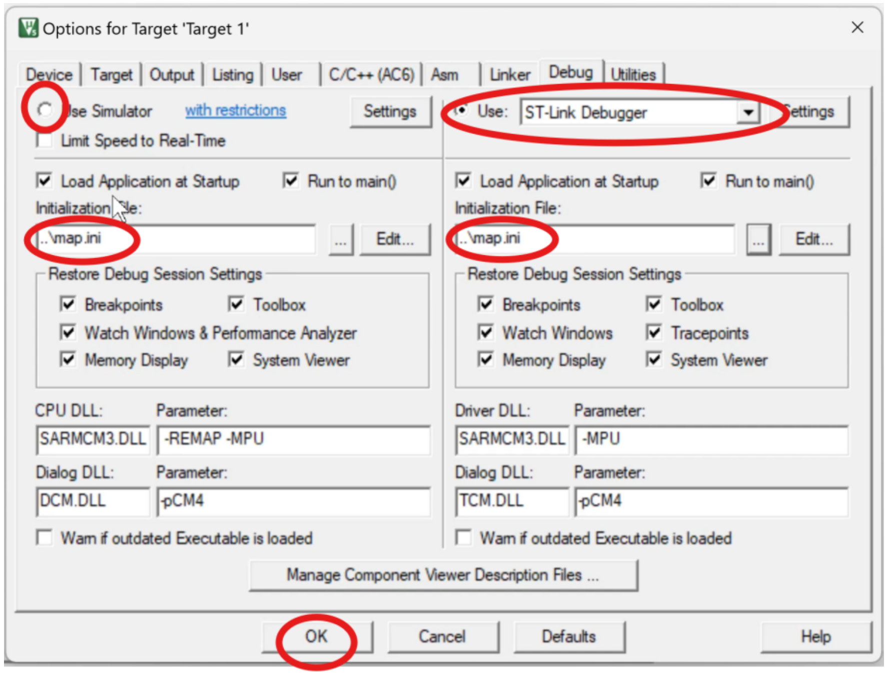
In the “Target” tab, choose latest compiler version.



```
map 0x40000000, 0x4001FFFF read write exec
map 0xE0000000, 0xE0050000 read write exec
```

Options for Target 'Target 1'
Device | Target | Output | Listing | User | C/C++ (AC6) | Asm | Linker | Debug | Utilities
STMicroelectronics STM32F401RETx, Xtal (MHz): <undefined>, Code Generation: ARM Compiler: V6.22

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 ST-Link Debugger on the right instead to load code onto the STM32F401RE**.

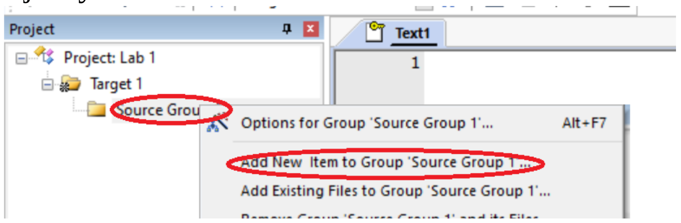


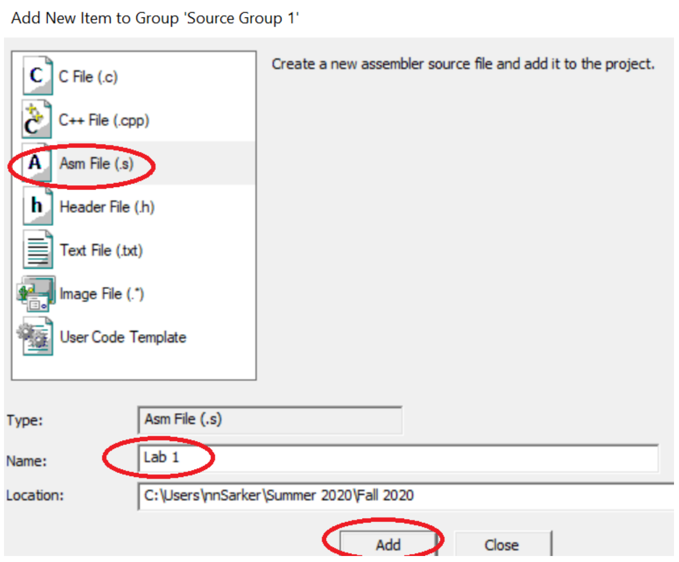
Options for Target 'Target 1' – Debug

Use Simulator (with restrictions)	Use: ST-Link Debugger
Limit Speed to Real-Time	
✓ Load Application at Startup, ✓ Run to main()	✓ Load Application at Startup, ✓ Run to main()
Initialization File: ..\map.ini	Initialization File: ..\map.ini
Restore Debug Session Settings: ✓ Breakpoints, ✓ Toolbox, ✓ Watch Windows & Performance Analyzer, ✓ Memory Display, ✓ System Viewer	Restore Debug Session Settings: ✓ Breakpoints, ✓ Toolbox, ✓ Watch Windows, ✓ Tracepoints, ✓ Memory Display, ✓ System Viewer
CPU DLL: SARMCM3.DLL, Parameter: -REMAP -MPU	Driver DLL: SARMCM3.DLL, Parameter: -MPU
Dialog DLL: DCM.DLL, Parameter: -pCM4	Dialog DLL: TCM.DLL, Parameter: -pCM4
Warn if outdated Executable is loaded	Warn if outdated Executable is loaded

Step 4:

Add a ‘.s’ file to the project you created.


Project
Project: Lab 1
Target 1
Source Grou
Text1
Options for Group 'Source Group 1'... Alt+F7
Add New Item to Group 'Source Group 1'...
Add Existing Files to Group 'Source Group 1'...



Create a new assembler source file and add it to the project.

Type: Asm File (.s)

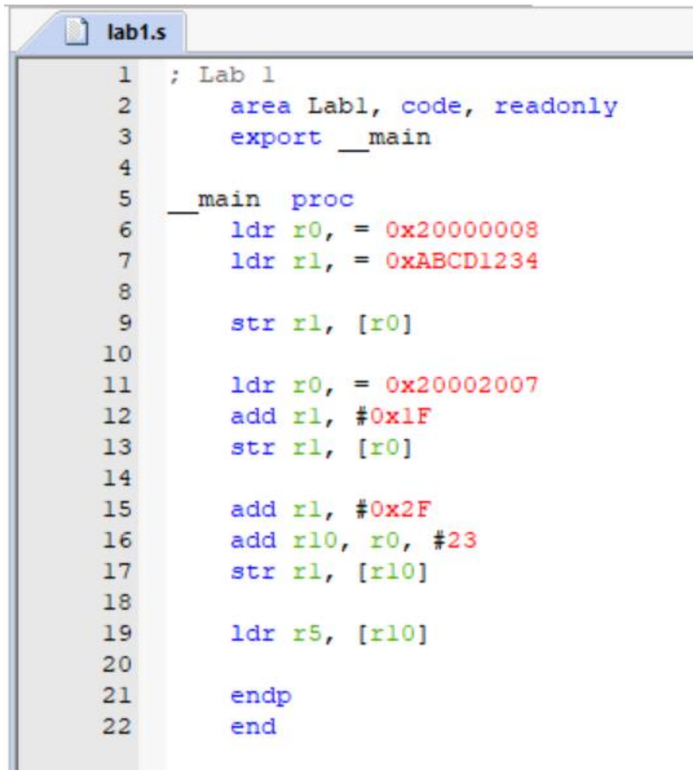
Name: Lab 1

Location: C:\Users\nnSarker\Summer 2020\Fall 2020

Lab Program

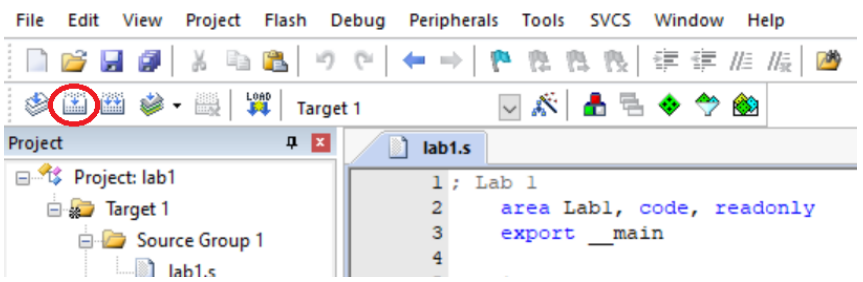
Step 1:

Use the following program in your Lab 1.



```
; Lab 1
    area Lab1, code, readonly
    export __main

__main  proc
    ldr r0, = 0x20000008
    ldr r1, = 0xABCD1234

    str r1, [r0]

    ldr r0, = 0x20002007
    add r1, #0x1F
    str r1, [r0]

    add r1, #0x2F
    add r10, r0, #23
    str r1, [r10]

    ldr r5, [r10]

    endp
    end
```

Step 2:
Build the program using the button shown by ellipse.


File Edit View Project Flash Debug Peripherals Tools SVCS Window Help
Target 1
Project
Project: lab1
Target 1
Source Group 1
lab1.s
lab1.s

```
; Lab 1
    area Lab1, code, readonly
    export __main
```

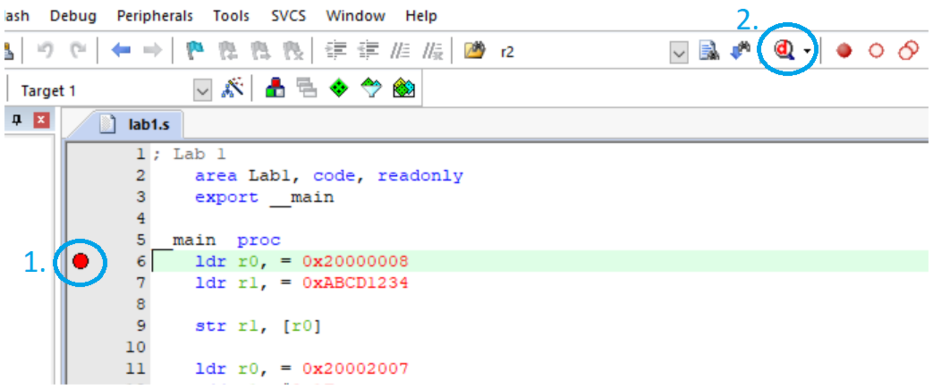
<u>Debugging</u>

Now it is time to debug the program <u>**one instruction**</u> 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.

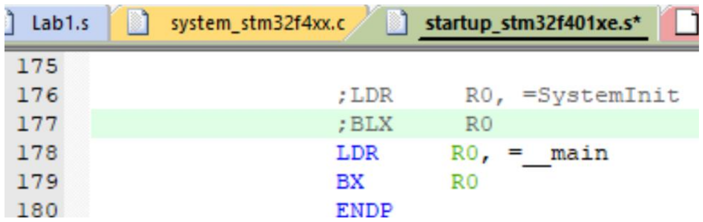
Put your cursor in these lines one at a time and click on the solid red ball. Then to start debugging, click on the button marked with ‘d’ inside. Click ‘OK’ in the “EVALUATION MODE” popup.

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.)



```
; Lab 1
    area Lab1, code, readonly
    export __main

__main proc
    ldr r0, = 0x20000008
    ldr r1, = 0xABCD1234

    str r1, [r0]

    ldr r0, = 0x20002007
```

Speeding up debugging (recommended)



```
175
176                 ;LDR     R0, =SystemInit
177                 ;BLX     R0
178                 LDR     R0, =__main
179                 BX      R0
180                 ENDP
```

Comment out the lines shown above (as 176 and 177) in the startup_stm32f401xe.s file. Repeat the steps for all projects using the Keil debugger but **<u>make sure to uncomment the lines while uploading programs to STM32F401RE in the later labs</u>**.

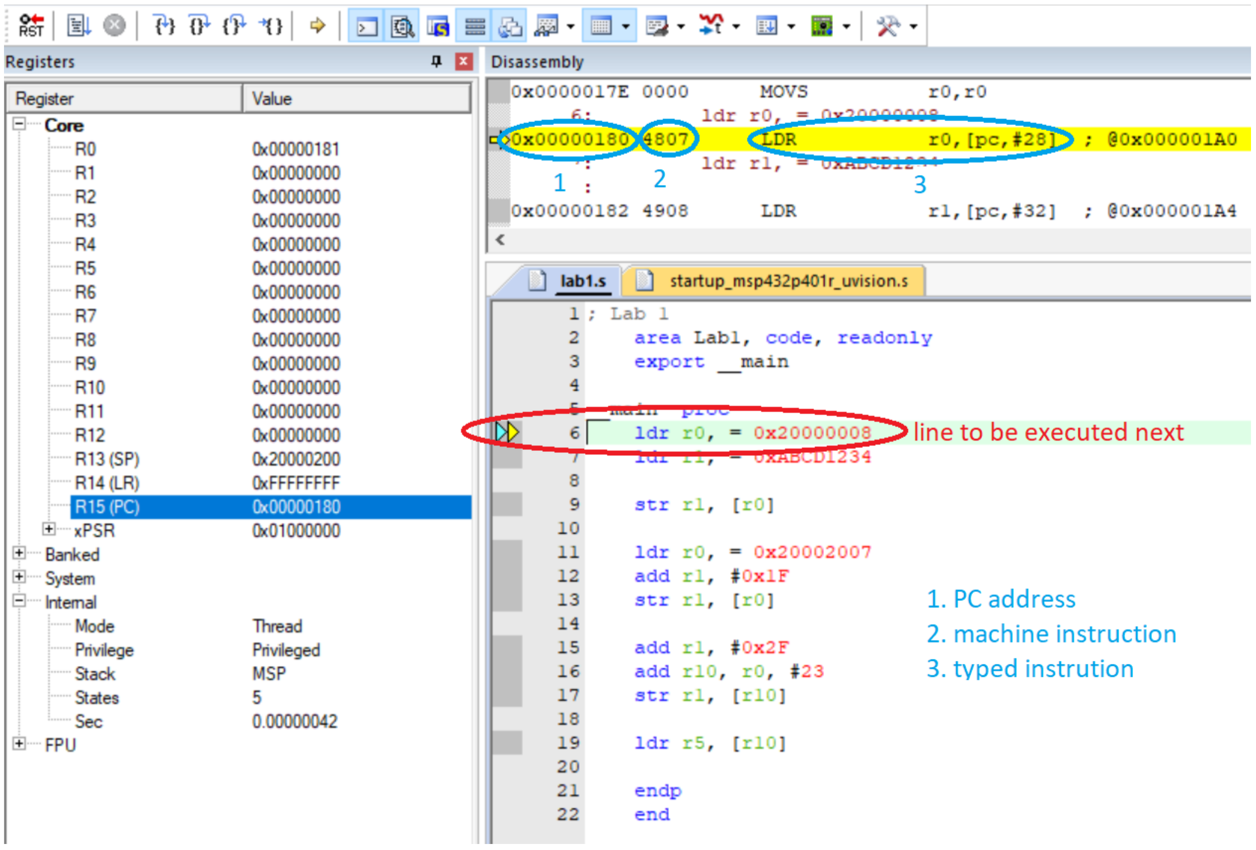
Complete executing startup code

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.



Registers

Register	Value
Core	
R0	0x00000181
R1	0x00000000
R2	0x00000000
R3	0x00000000
R4	0x00000000
R5	0x00000000
R6	0x00000000
R7	0x00000000
R8	0x00000000
R9	0x00000000
R10	0x00000000
R11	0x00000000
R12	0x00000000
R13 (SP)	0x20000200
R14 (LR)	0xFFFFFFFF
R15 (PC)	0x00000180
xPSR	0x01000000
Banked	
System	
Internal	
Mode	Thread
Privilege	Privileged
Stack	MSP
States	5
Sec	0.00000042
FPU	

Disassembly

```
0x0000017E 0000      MOVS      r0,r0
     6:        ldr r0, = 0x20000008
0x00000180 4807      LDR       r0,[pc,#28]  ; @0x000001A0
     7:        ldr r1, = 0xABCD1234
0x00000182 4908      LDR       r1,[pc,#32]  ; @0x000001A4
```

lab1.s   startup_msp432p401r_uvision.s

```
; Lab 1
    area Lab1, code, readonly
    export __main

__main proc
    ldr r0, = 0x20000008
    ldr r1, = 0xABCD1234

    str r1, [r0]

    ldr r0, = 0x20002007
    add r1, #0x1F
    str r1, [r0]

    add r1, #0x2F
    add r10, r0, #23
    str r1, [r10]

    ldr r5, [r10]

    endp
    end
```

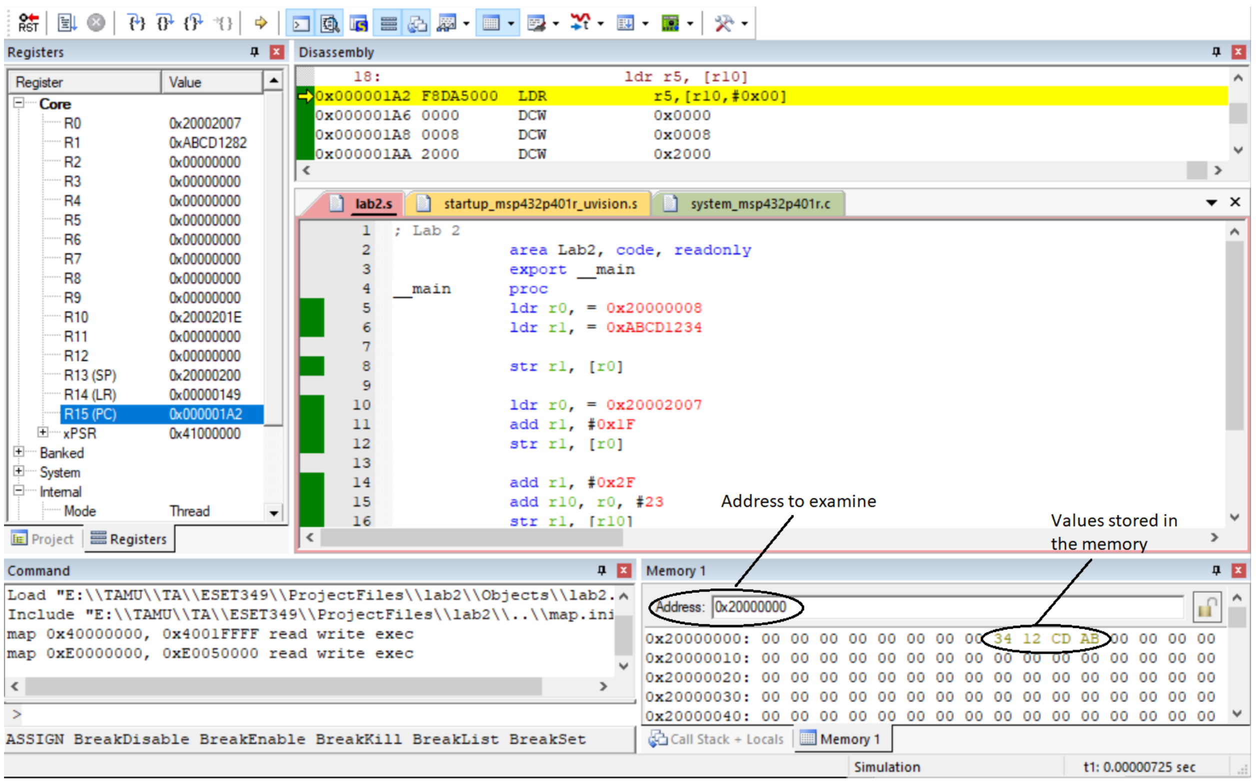
Figure 2. Breakdown of disassembly

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.



Register	Value
Core	
R0	0x20002007
R1	0xABCD1282
R2	0x00000000
R3	0x00000000
R4	0x00000000
R5	0x00000000
R6	0x00000000
R7	0x00000000
R8	0x00000000
R9	0x00000000
R10	0x2000201E
R11	0x00000000
R12	0x00000000
R13 (SP)	0x20000200
R14 (LR)	0x00000149
R15 (PC)	0x000001A2
xPSR	0x41000000
Banked	
System	
Internal	
Mode	Thread

```
    18:                               ldr r5, [r10]
0x000001A2 F8DA5000  LDR                  r5,[r10,#0x00]
0x000001A6 0000      DCW                  0x0000
0x000001A8 0008      DCW                  0x0008
0x000001AA 2000      DCW                  0x2000
```

```
; Lab 2
                area Lab2, code, readonly
                export __main
__main          proc
                ldr r0, = 0x20000008
                ldr r1, = 0xABCD1234

                str r1, [r0]

                ldr r0, = 0x20002007
                add r1, #0x1F
                str r1, [r0]

                add r1, #0x2F
                add r10, r0, #23
                str r1, [r10]
```

```
Load "E:\\TAMU\\TA\\ESET349\\ProjectFiles\\lab2\\Objects\\lab2.
Include "E:\\TAMU\\TA\\ESET349\\ProjectFiles\\lab2\\..\\map.ini
map 0x40000000, 0x4001FFFF read write exec
map 0xE0000000, 0xE0050000 read write exec
>
ASSIGN BreakDisable BreakEnable BreakKill BreakList BreakSet
```

```
Address: 0x20000000
0x20000000: 00 00 00 00 00 00 00 00 34 12 CD AB 00 00 00 00
0x20000010: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x20000020: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x20000030: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x20000040: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```

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

The lab requires you to **<u>complete Tables 1 and 2</u>** 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. <u>If you have a situation that the last statement, `ldr r5, [r10]`, does not get executed with F11</u>, 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.