MANIPAL INSTITUTE OF TECHNOLOGY
MANIPAL
(A constituent unit of MAHE, Manipal)

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

INSPIRED BY LIFE

CERTIFICATE

This is to certify that Ms./Mr. …………………...……………………………………

Reg. No. …..…………………… Section: ……………… Roll No: ………………... has satisfactorily completed the lab exercises prescribed for Embedded systems lab [CSE 2263] of Second Year B. Tech. in Computer Science and Engineering Degree at MIT, Manipal, in the academic year 2020-2021.

Date: ……...................................

Signature
Faculty in Charge

CONTENTS

LAB NO	TITLE	PAGENO.
	COURSE OBJECTIVES AND OUTCOMES	1
	EVALUATION PLAN	1
	INSTRUCTIONS TO THE STUDENTS	2-3
	SAMPLE LAB OBSERVATION NOTE PREPARATION	4
1	INTRODUCTION TO KEIL MVISION-4 AND PROGRAMS ON DATA TRANSFER INSTRUCTIONS	5
2	PROGRAMS ON ARITHMETIC INSTRUCTIONS	21
3	PROGRAMS ON ARITHMETIC AND LOGICAL INSTRUCTIONS	23
4	BRANCHING AND LOOPING	25
5	SORTING, SEARCHING PROGRAMS	27
6	INTERFACING LED TO ARM MICROCONTROLLER.	30
7	PROGRAMS ON MULTIPLEXED SEVEN SEGMENT DISPLAY	36
8	LIQUID CRYSTAL DISPLAY (LCD) AND KEYBOARD INTERFACING	42
9	ANALOG TO DIGITAL CONVERTOR PROGRAM	52

10	PROGRAM ON DIGITAL TO ANALOG CONVERTOR (DAC)	55
11	PROGRAM ON PULSE WIDTH MODULATION (PWM)	57
12	PROGRAM ON STEPPER MOTOR	60
	APPENDIX A	64
	APPENDIX B	68
	APPENDIX C	72

Course Objectives

- To gain knowledge about assembly language and Embedded C programming
- To implement the programs using ARM instruction set
- To understand various interfacing circuits necessary for various applications and programming using ARM.

Course Outcomes

On the completion of this laboratory course, the students will be able to:

- Gain knowledge about simulators for an embedded system and to execute simple programs.
- Comprehend the software development for ARM cortex-M microcontroller using assembly language.
- Develop embedded C program for ARM cortex-M microcontroller by interfacing various modules to ARM kit

Evaluation plan

- Internal Assessment Marks : 60%

    ✓ Continuous evaluation component (for each experiment):10 marks

    ✓ The assessment will depend on punctuality, program execution, maintaining the observation note and answering the questions in viva voce

    ✓ Total marks of the 12 experiments reduced to marks out of 60

- End semester assessment of 2 hour duration: 40 %

INSTRUCTIONS TO THE STUDENTS

Pre- Lab Session Instructions

1. Students should carry the Class notes, Lab Manual and the required stationery to every lab session
2. Be in time and follow the Instructions from Lab Instructors
3. Must Sign in the log register provided
4. Make sure to occupy the allotted seat and answer the attendance
5. Adhere to the rules and maintain the decorum

In- Lab Session Instructions

- Follow the instructions on the allotted exercises given in Lab Manual
- Show the program and results to the instructors on completion of experiments
- On receiving approval from the instructor, copy the program and results in the Lab record
- Prescribed textbooks and class notes can be kept ready for reference if required

General Instructions for the exercises in Lab

- The programs should meet the following criteria:
    - Programs should be interactive with appropriate prompt messages, error messages if any, and descriptive messages for outputs.
    - Use meaningful names for variables and procedures.
- Plagiarism (copying from others) is strictly prohibited and would invite severe penalty during evaluation.
- The exercises for each week are divided under three sets:
    - Solved exercise
    - Lab exercises - to be completed during lab hours
    - Additional Exercises - to be completed outside the lab or in the lab to enhance the skill
- In case a student misses a lab class, he/ she must ensure that the experiment is completed at students end or in a repetition class (if available) with the permission of the faculty concerned but credit will be given only to one day’s experiment(s).

- Questions for lab tests and examination are not necessarily limited to the questions in the manual, but may involve some variations and / or combinations of the questions.

- A sample note preparation is given later in the manual as a model for observation.

LAB NO 1

Sample lab observation note preparation

LAB NO: **Date:**

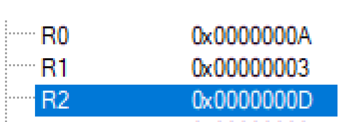
Title: INTRODUCTION TO KEIL μVISION-4 AND PROGRAMS ON DATA TRANSFER INSTRUCTIONS

Add two immediate values in the registers and store the result in the third register.

Program:

```
AREA RESET, DATA, READONLY
        EXPORT __Vectors
__Vectors
                DCD 0X10001000
                DCD Reset_Handler
                ALIGN
                AREA mycode, CODE, READONLY
                ENTRY
                EXPORT Reset_Handler
Reset_Handler
        MOV R0, #10
        MOV R1, #3
        ADD R2, R0, R1
        END
```

Sample output:



LAB NO: 1

INTRODUCTION TO KEIL µVISION-4 AND PROGRAMS ON DATA TRANSFER INSTRUCTIONS

Objectives:

In this lab, students will be able to

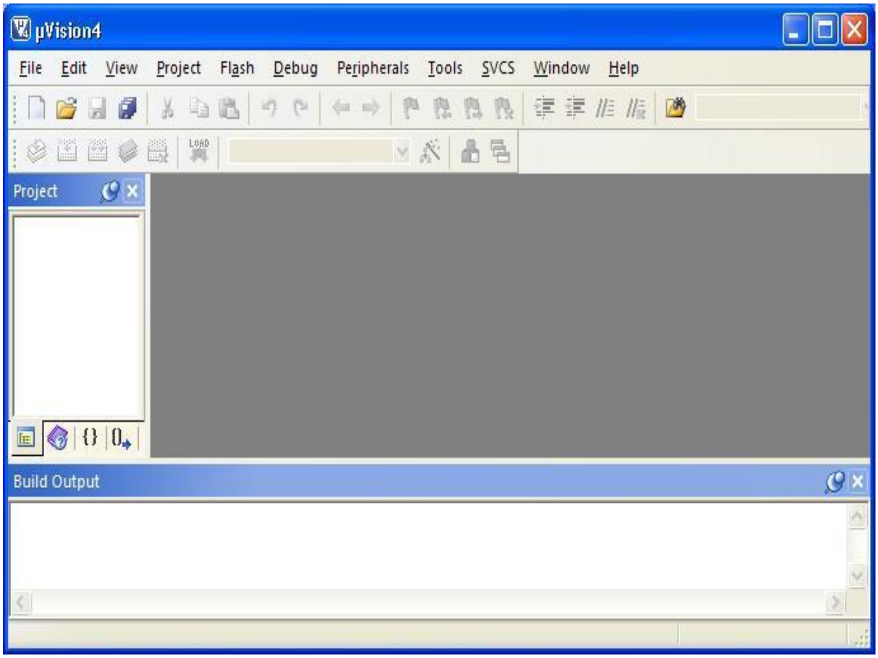
- Understand the usage of Keil µVision 4 software for assembly language.
- Write, build and execute assembly language programs in Keil µVision 4.
- Use different data transfer instructions of ARM processor.

I. Running an assembly language program in Keil µVision 4

Step 1:

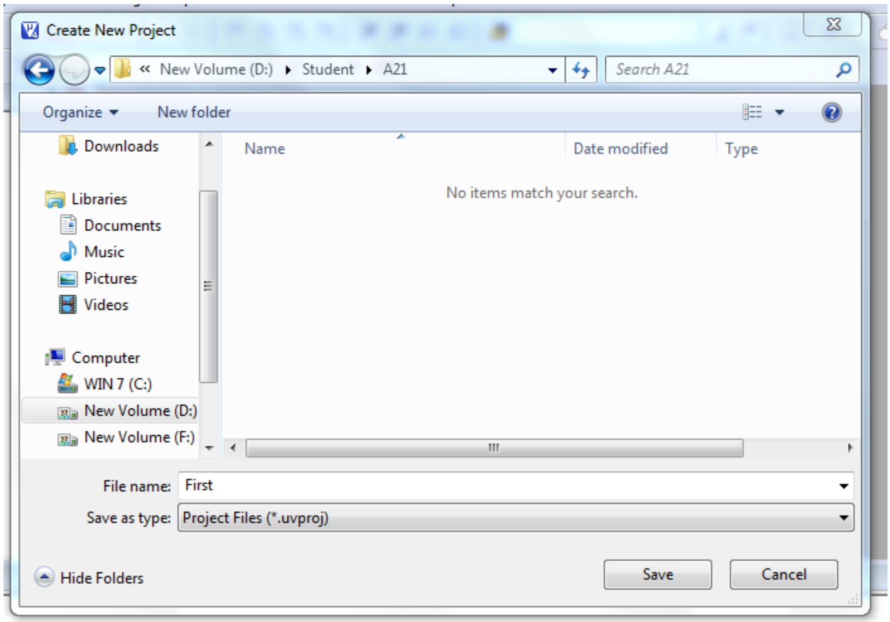
- Create a directory with section followed by roll number (to be unique); e.g. A21
- Start up µVision-4 by clicking on the icon 

from the desktop or from the "Start" menu or "All Programs". The following screen appears.

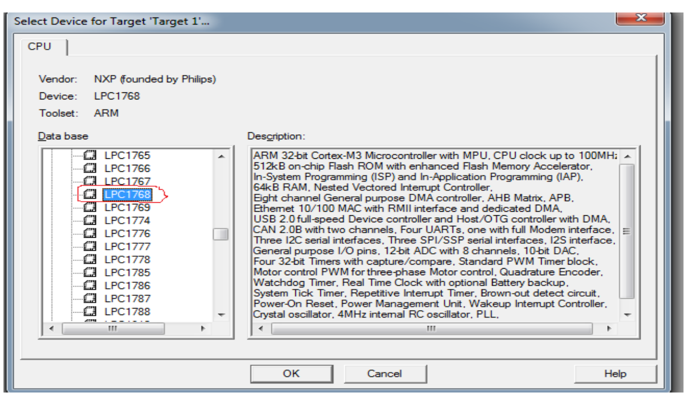


Step 2: Create a project

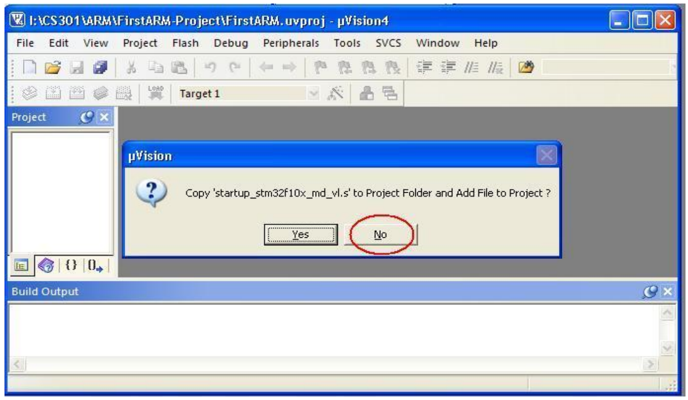
To create a project, click on the "Project" menu from the µVision-4 screen and select "New µVision Project". Then, select the folder you have created already, give project name and save.



From the "Select Device for Target Target 1..." window, select "NXP" as the vendor. In that, select LPC1768 ARM controller, and then click on OK button. Some general information of the chip is shown in the description box.

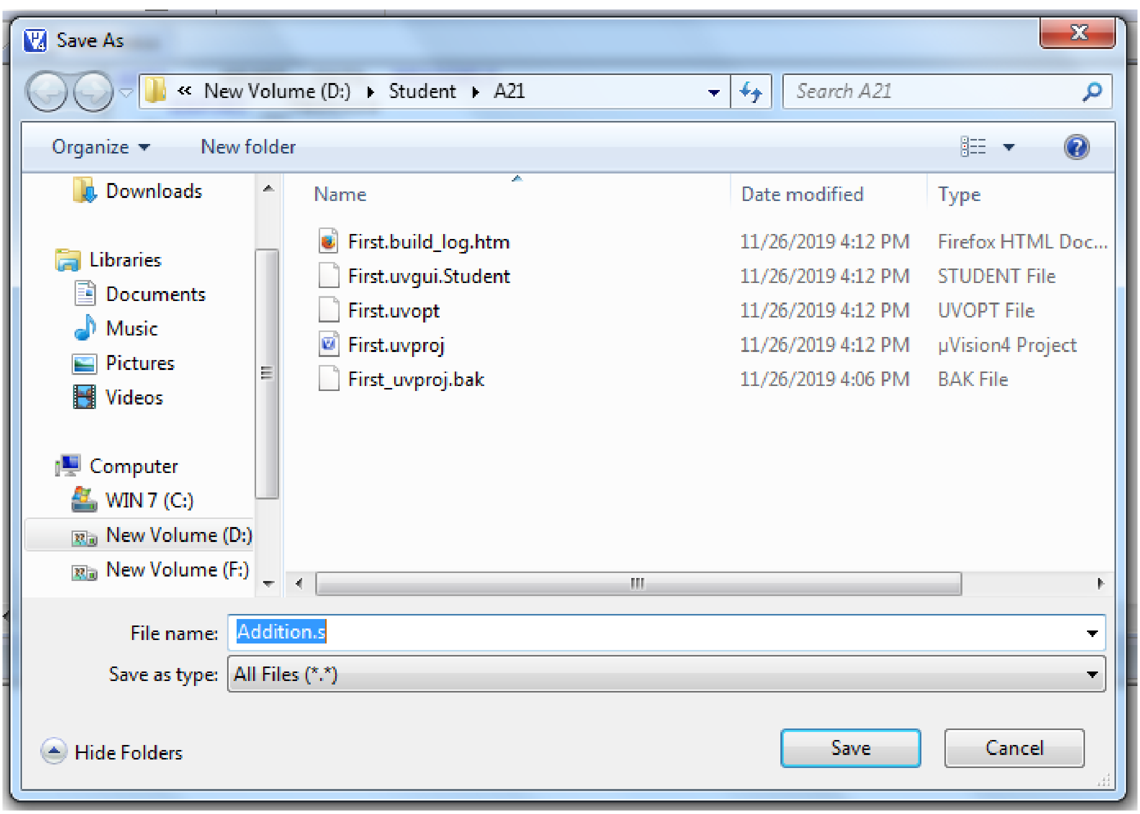


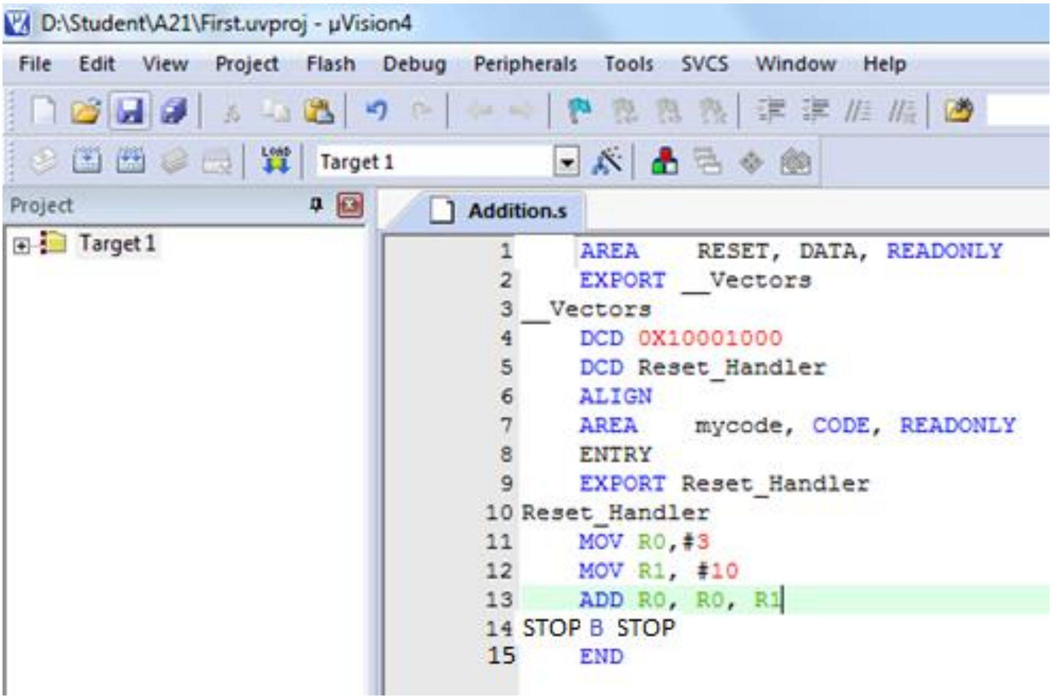
Make sure you click on "NO" for the following pop up window.



Step 3: Create Source File

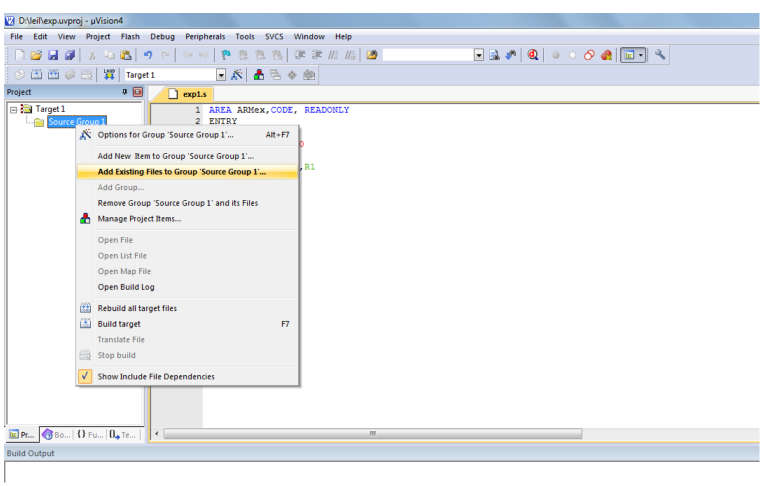
From the "File" menu, select "New", to get the editor window. Type the program here. (Note: give a tab space at the beginning). Save the program with .s extension in the directory.



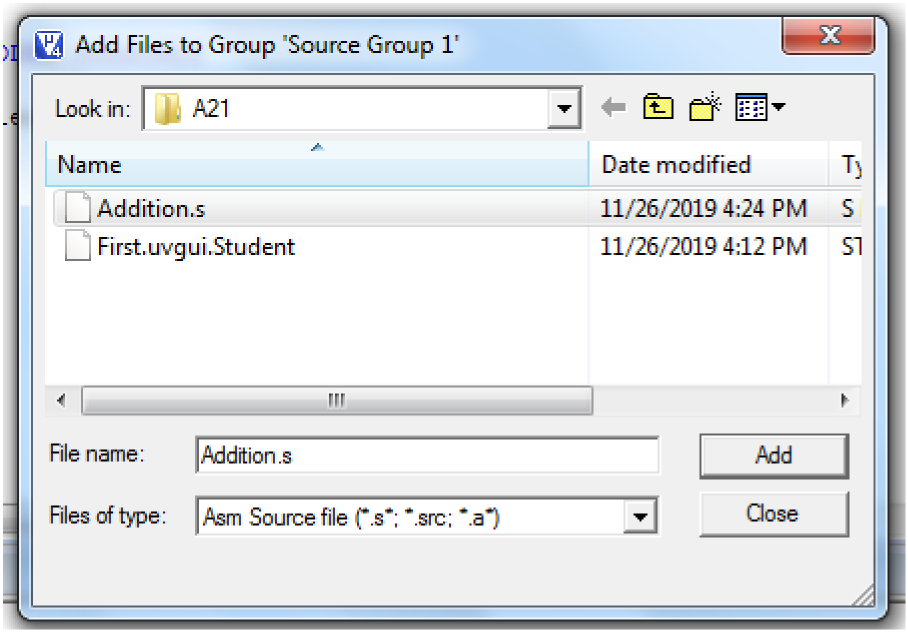


Step 4: Add Source File to the Project

Click on the + symbol near the Target 1 in the top left corner of the window. Right click on the "Source Group 1", select "Add Existing Files to Group 'Source Group 1'".

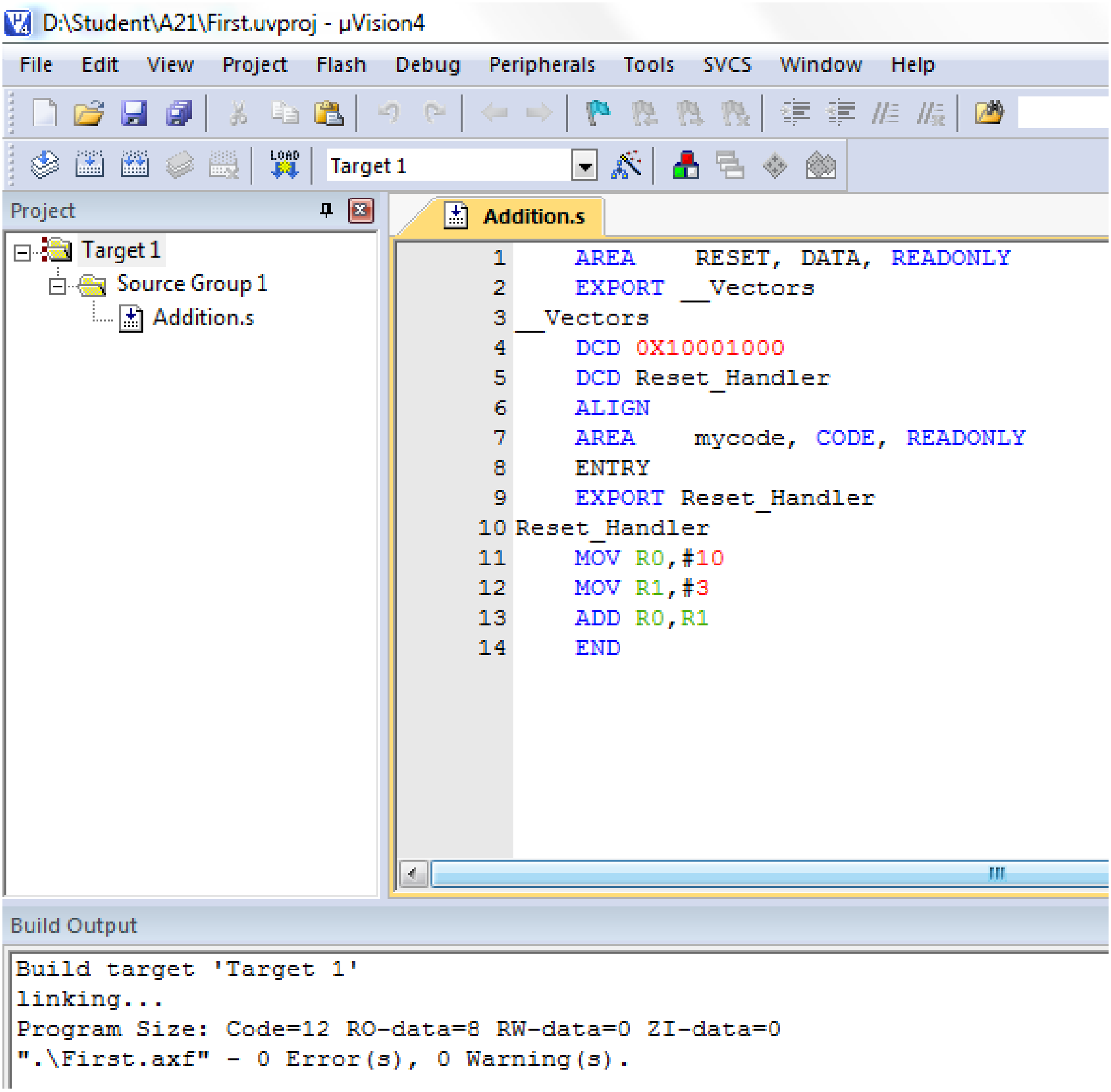


Select "Files of type" as "asm Source file (*.s*;*.src*;*.a*), then select the file. Click on "Add", and then click on "Close".



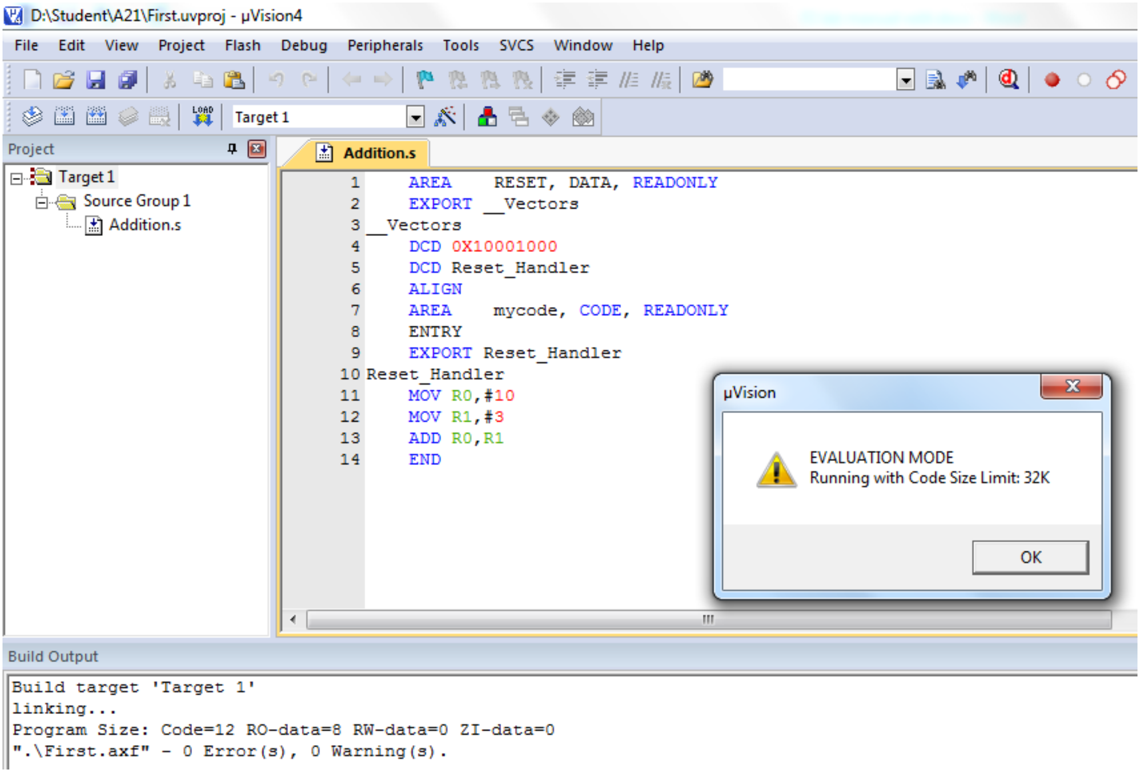
Step 5: Build your project

Click on the "+" beside the "Source Group 1", you will see the program " Addition.s" Click on the "Build" button or from the "Project" menu, you will see the following screen.

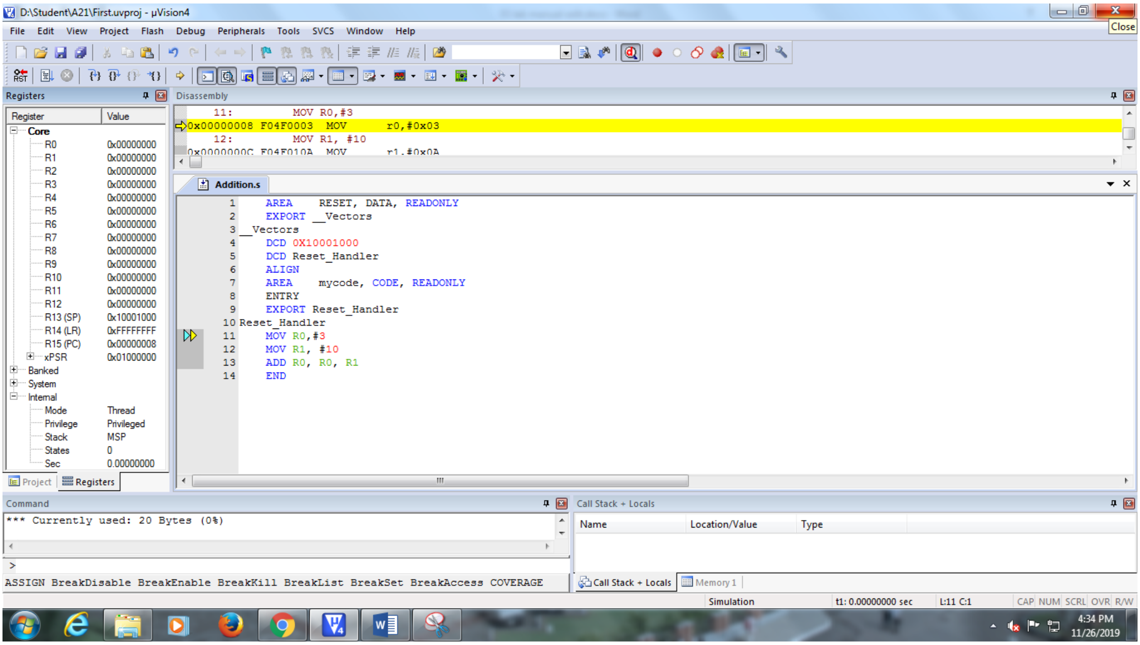


Step 6: Run the program

Run the program through the "Debug" menu.

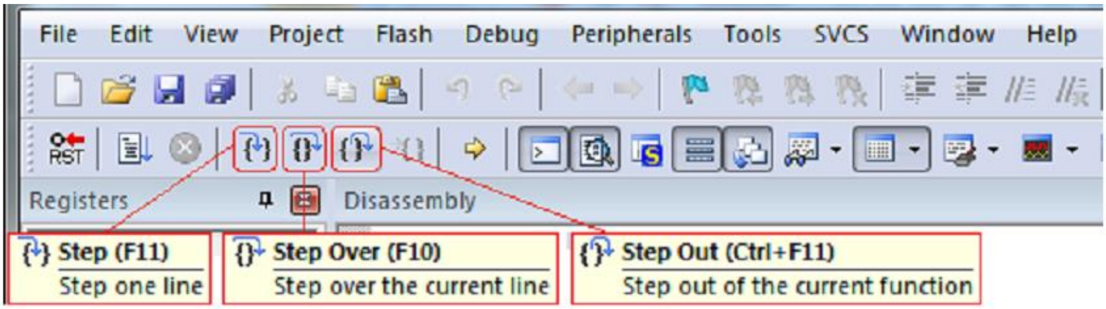


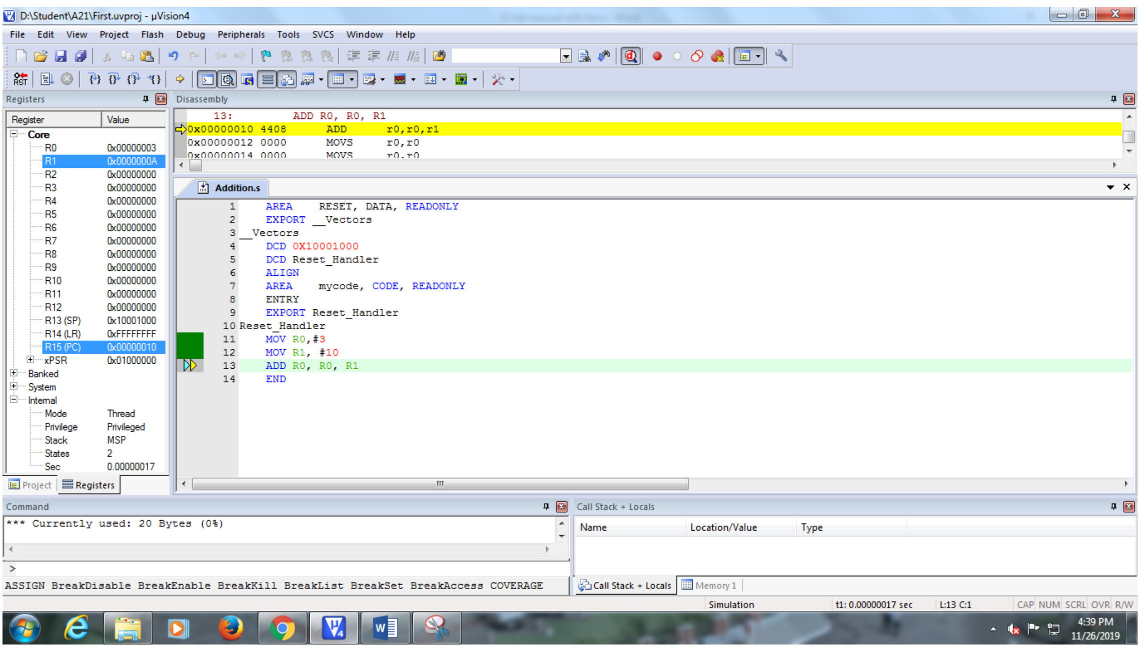
Click on "OK" for the pop up window showing "EVALUATION MODE, Running with Code Size Limit: 32K". You will see the following window.

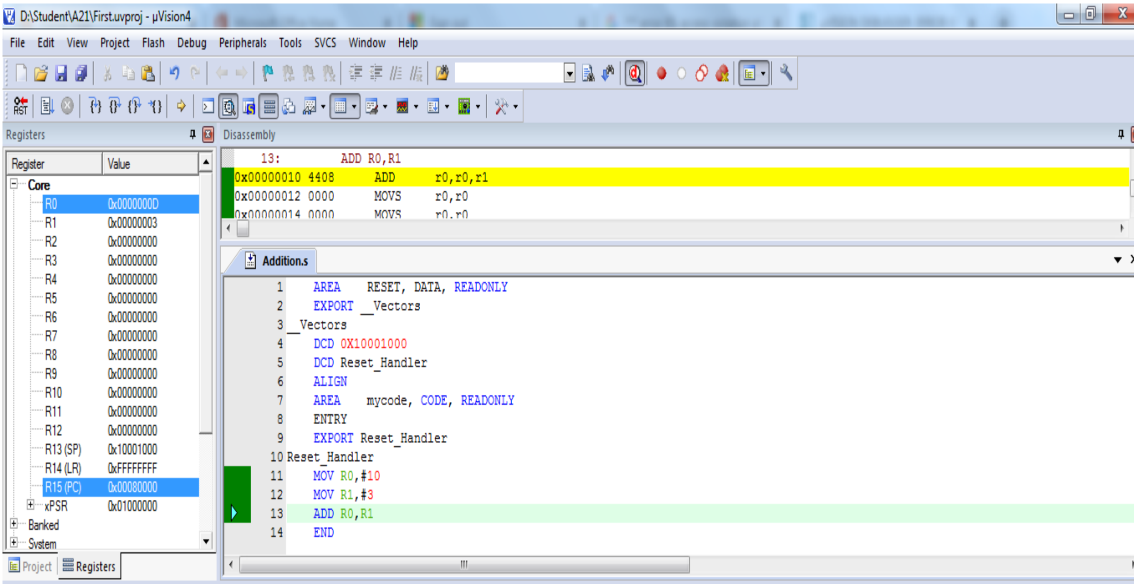


Open µVision4 to full screen to have a better and complete view. The left hand side window shows the registers and the right side window shows the program code. There are some other windows open. Adjust the size of them to have a better view. Run the program step by step; observe the change of the values in the registers.

Run the program using the **Step Over** button or click on **Step Over** from the Debug menu. It executes the instructions of the program one after another. To trace the program one can use the **Step** button, as well. The difference between the **Step Over** and **Step** is in executing functions. While **Step** goes into the function and executes its instructions one by one, **Step Over** executes the function completely and goes to the instruction next to the function. To see the difference between them, trace the program once with **Step Over** and then with **Step**. When the PC is executing the function and wants the function to be executed completely one can use **Step Out**. In this case, the instructions of the function will be executed, it returns from the function, and goes to the instruction which is next to the function call.


File Edit View Project Flash Debug Peripherals Tools SVCS Window Help
Registers
Disassembly
Step (F11)
Step one line
Step Over (F10)
Step over the current line
Step Out (Ctrl+F11)
Step out of the current function


D:\Student\A21\First.uvproj - µVision4
AREA RESET, DATA, READONLY
EXPORT __Vectors
__Vectors
DCD 0X10001000
DCD Reset_Handler
ALIGN
AREA mycode, CODE, READONLY
ENTRY
EXPORT Reset_Handler
Reset_Handler
MOV R0,#3
MOV R1, #10
ADD R0, R0, R1
END



Click on the "**Start/Stop Debug** Session" again to stop execution of the program.

II. ARM assembly language module

An ARM assembly language module has several constituent parts.

These are:

- Extensible Linking Format (ELF) sections (defined by the AREA directive).
- Application entry (defined by the ENTRY directive).
- Program end (defined by the END directive).

Assembler Directives

- Assembler directives are the commands to the assembler that direct the assembly process.
- They do not generate any machine code i.e. they do not contribute to the final size of machine code and they are assembler specific

AREA:

The AREA directive tells the assembler to define a new section of memory. The memory can be code (instructions) or data and can have attributes such as READONLY, READWRITE and so on. This is used to define one or more blocks of indivisible memory for code or data to be used by the linker. The following is the format:

AREA sectionname attribute, attribute, …

The following line defines a new area named mycode which has CODE and READONLY attributes:

AREA mycode, CODE, READONLY

Commonly used attributes are CODE, DATA, READONLY, READWRITE, ALIGN and END.

READONLY:

It is an attribute given to an area of memory which can only be read from. It is by default for CODE. This area is used to write the instructions.

READWRITE:

It is an attribute given to an area of memory which can be read from and written to. It is by default for DATA.

CODE:

It is an attribute given to an area of memory used for executable machine instructions. It is by default READONLY memory.

DATA:

It is an attribute given to an area of memory used for data and no instructions can be placed in this area. It is by default READWRITE memory.

ALIGN:

It is an attribute given to an area of memory to indicate how memory should be allocated according to the addresses. When the ALIGN is used for CODE and READONLY, it is aligned in 4-bytes address boundary by default since the ARM instructions are 32 bit word. If it is written as ALIGN = 3, it indicates that the information should be placed in memory with addresses of 2^3, that is for example 0x50000, 0x50008, 0x50010, 0x50018 and so on.

EXPORT:

The EXPORT directive declares a symbol that can be used by the linker to resolve symbol references in separate object and library files.

DCD (Define constant word):

Allocates a word size memory and initializes the values. Allocates one or more words of memory, aligned on 4-byte boundaries and defines initial run time contents of the memory.

ENTRY:

The ENTRY directive declares an entry point to the program. It marks the first instruction to be executed. In applications using the C library, an entry point is also contained within the C library initialization code. Initialization code and exception handlers also contain entry points

END:

It indicates to the assembler the end of the source code. The END directive is the last line of the ARM assembly program and anything after the END directive in the source file is ignored by the assembler.

Example:

```
        AREA RESET, DATA, READONLY
        EXPORT __Vectors
__Vectors
```

```
        DCD 0X10001000    ;stack pointer value when stack is empty
                          ;The processor uses a full descending stack.
                          ;This means the stack pointer holds the address of the last
                     ;stacked item in memory. When the processor pushes a new item
                          ;onto the stack, it decrements the stack pointer and then
                          ;writes the item to the new memory location.
        DCD Reset_Handler   ; reset vector. The program linker requires Reset_Handler

        ALIGN

        AREA mycode, CODE, READONLY

        ENTRY

        EXPORT Reset_Handler

Reset_Handler

;;;;;;;;;;User Code Starts from the next line;;;;;;;;;;;;

        MOV R0, #10

        MOV R1, #3

        ADD R0, R1

STOP    B  STOP

        END    ;End of the program
```

III. Introduction to ARM addressing modes

Data can be transferred into and out of the ARM controller using different addressing modes. There are different ways to specify the address of the operands for any given operations such as load, add or branch. The different ways of determining the address of the operands are called addressing modes. Different addressing modes used in ARM are listed in Appendix A.

Solved Exercise:

Write an ARM assembly language program to copy 32 bit data from code memory to data memory.

Source: SRC= 0X00000008 at location pointed by R0

Destination: DST = 0X00000008 at location pointed by R1 after the execution

Program:

```
        AREA    RESET, DATA, READONLY
        EXPORT  __Vectors

   __Vectors
       DCD  0x10001000     ; stack pointer value when stack is empty
       DCD  Reset_Handler  ; reset vector
       ALIGN
       AREA mycode, CODE, READONLY
       ENTRY
       EXPORT Reset_Handler

   Reset_Handler
        LDR R0, =SRC       ; Load address of SRC into R0
        LDR R1, =DST       ; Load the address of DST onto R1
        LDR R3, [R0]       ; Load data pointed by R0 into R3
        STR R3,[R1]        ; Store data from R3 into the address pointed by R1
   STOP
        B STOP         ; Be there
   SRC DCD 8         ;SRC  location in code memory
        AREA mydata, DATA, READWRITE
   DST DCD 0                  ;DST location in data memory
        END
```

Observations to be made

1. **Data storage into the memory:** Click on Memory window and go to Memory1 option. Type address pointed by R0 in address space and observe how the data are stored into the memory.
2. **Data movement from one memory to another memory:** Click on Memory window and go to Memory2 option. Type address pointed by R1 in address

space and observe data movement to another location before execution and after execution.

Lab Exercises:

1. Write an ARM assembly language program to store data into general purpose registers.
2. Write an ARM assembly language program to transfer a 32 bit number from one location in the data memory to another location in the data memory.
3. Write an ARM assembly language program to transfer block of ten 32 bit numbers from code memory to data memory when the source and destination blocks are non-overlapping.
4. Reverse an array of ten 32 bit numbers in the memory.

Additional Exercises:

1. Repeat Q3 above using pre indexing mode.
2. Repeat Q3 above when the source and destination blocks are overlapping

LAB NO: 2 **Date:**

PROGRAMS ON ARITHMETIC INSTRUCTIONS

Objectives:

In this lab, students will be able to

- Identify and use the instructions required to perform addition and subtraction
- Debug and trace the programs.

Refer Appendix A for instruction details.

Solved Exercise:

Write a program to add two 32 bit numbers available in the code memory. Store the result in the data memory

```
        AREA    RESET, DATA, READONLY
        EXPORT  __Vectors

  __Vectors
        DCD  0x40001000     ; stack pointer value when stack is empty
        DCD  Reset_Handler  ; reset vector

        ALIGN
        AREA mycode, CODE, READONLY
        ENTRY
        EXPORT Reset_Handler
Reset_Handler
        LDR R0, =VALUE1   ;pointer to the first value1
        LDR R1, [R0]      ;load the first value into R1
        LDR R0, =VALUE2   ;pointer to the second value
        LDR R3, [R0]      ;load  second number into r3
        ADDS R6, R1, R3   ;add two numbers and store the result in r6
        LDR R2, =RESULT
        STR R6, [R2]
  STOP
```

```
        B STOP
VALUE1 DCD 0X12345678       ; First 32 bit number
VALUE2 DCD 0XABCDEF12       ; Second 32 bit number
        AREA data, DATA, READWRITE
RESULT DCD 0
        END
```

Lab Exercises:

1. Write a program to add ten 32 bit numbers available in code memory and store the result in data memory.
2. Write a program to add two 128 bit numbers available in code memory and store the result in data memory.
   Hint: Use indexed addressing mode.
3. Write a program to subtract two 32 bit numbers available in the code memory and store the result in the data memory.
4. Write a program to subtract two 128 bit numbers available in the code memory and store the result in the data memory.

Additional Exercises:

1. Write a program to find the 2’s complement of 64 bit data in R0 and R1 registers. The R0 holds the lower 32 bit.
2. Add and subtract two 32 bit numbers and check all the flags. Take appropriate data to check all the flags.