EMBEDDED SYSTEMS LABORATORY

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

1. Follow the instructions on the allotted exercises given in Lab Manual

2. Show the program and results to the instructors on completion of experiments

3. On receiving approval from the instructor, copy the program and results in the Lab record

4. Prescribed textbooks and class notes can be kept ready for reference if required

LAB NO: 1
Title: 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 u Vision 4 software for assembly language.
- Write, build and execute assembly language programs in Keil u Vision 4.
- Use different data transfer instructions of ARM processor

## I. Running an assembly language program in Keil u Vision 4

Step 1:

- Create a directory with section followed by roll number (to be unique); e.g. A21

- Start up uVision4 by clicking on the icon from the desktop or from the "Start" menu or "All Programs". The following screen appears.
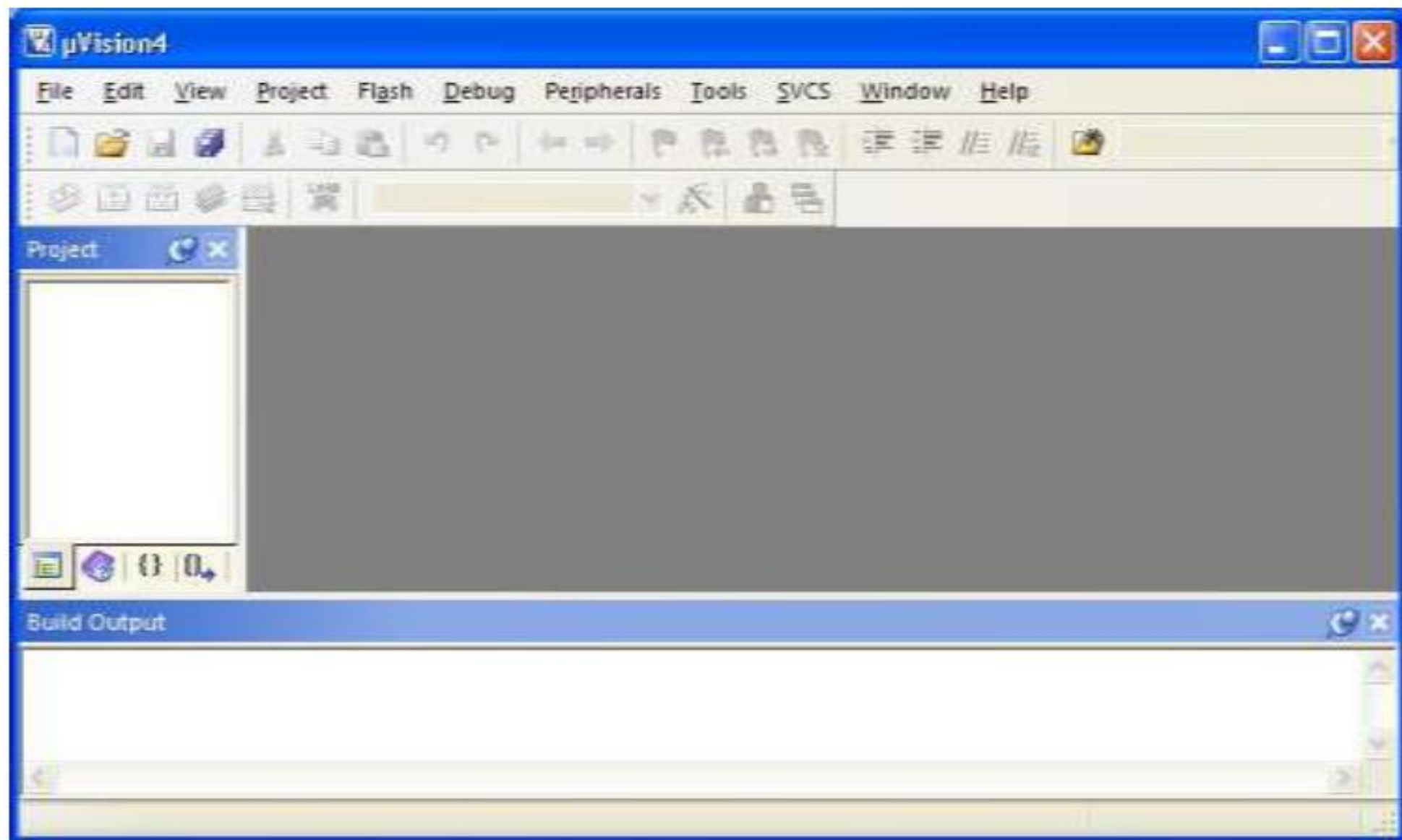
Step 2: Create a project

To create a project, click on the "Project" menu from the uVision4 screen and select "New uVision Project". Then, select the folder you have created already, give project name and save.
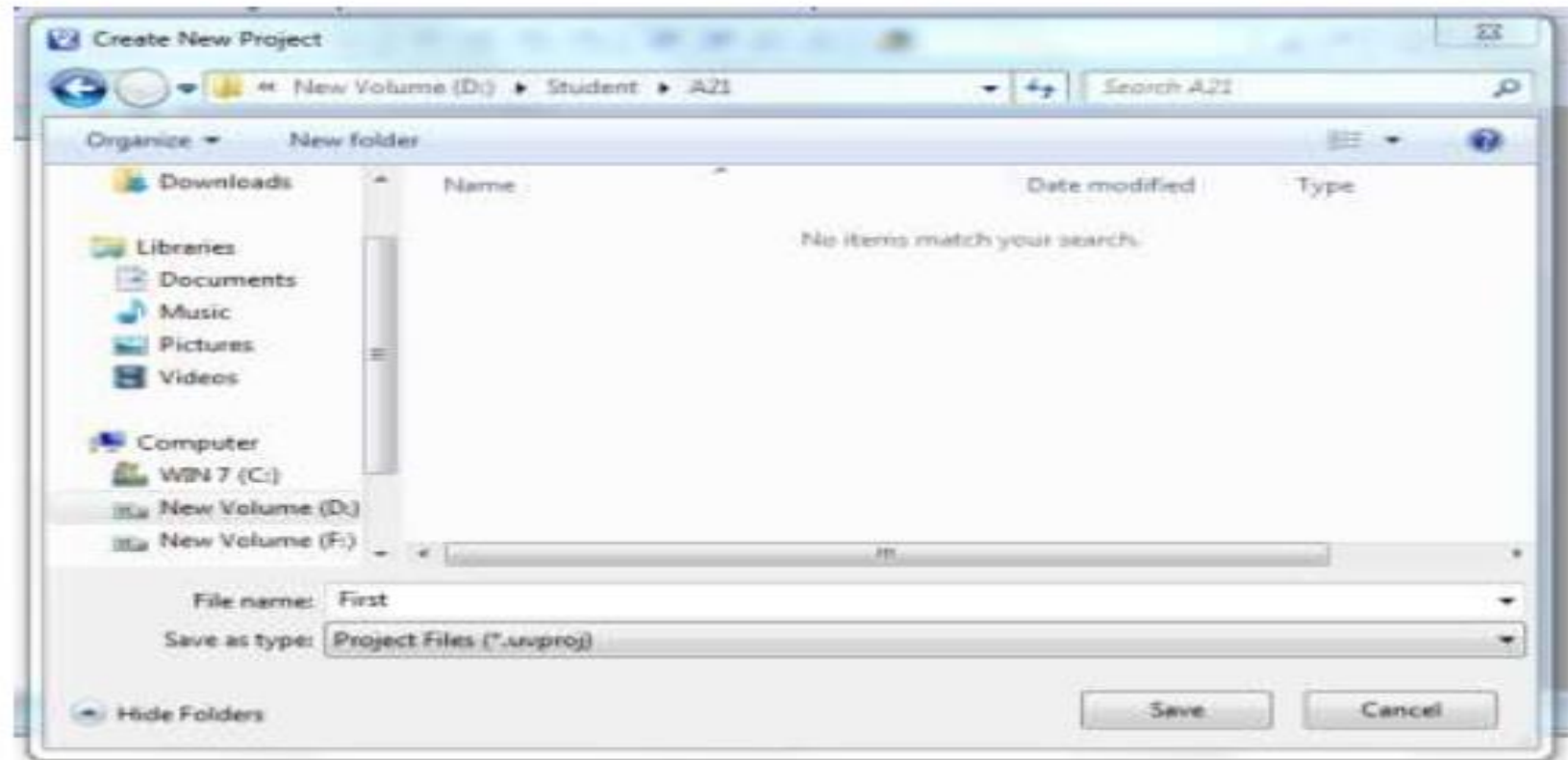
From the "Select Device for Target Target 1..." window, select "NXP" as the vendor. In that select LPC1768 ARM controller, then click on OK button. Some general information of the chip is shown in the description box.
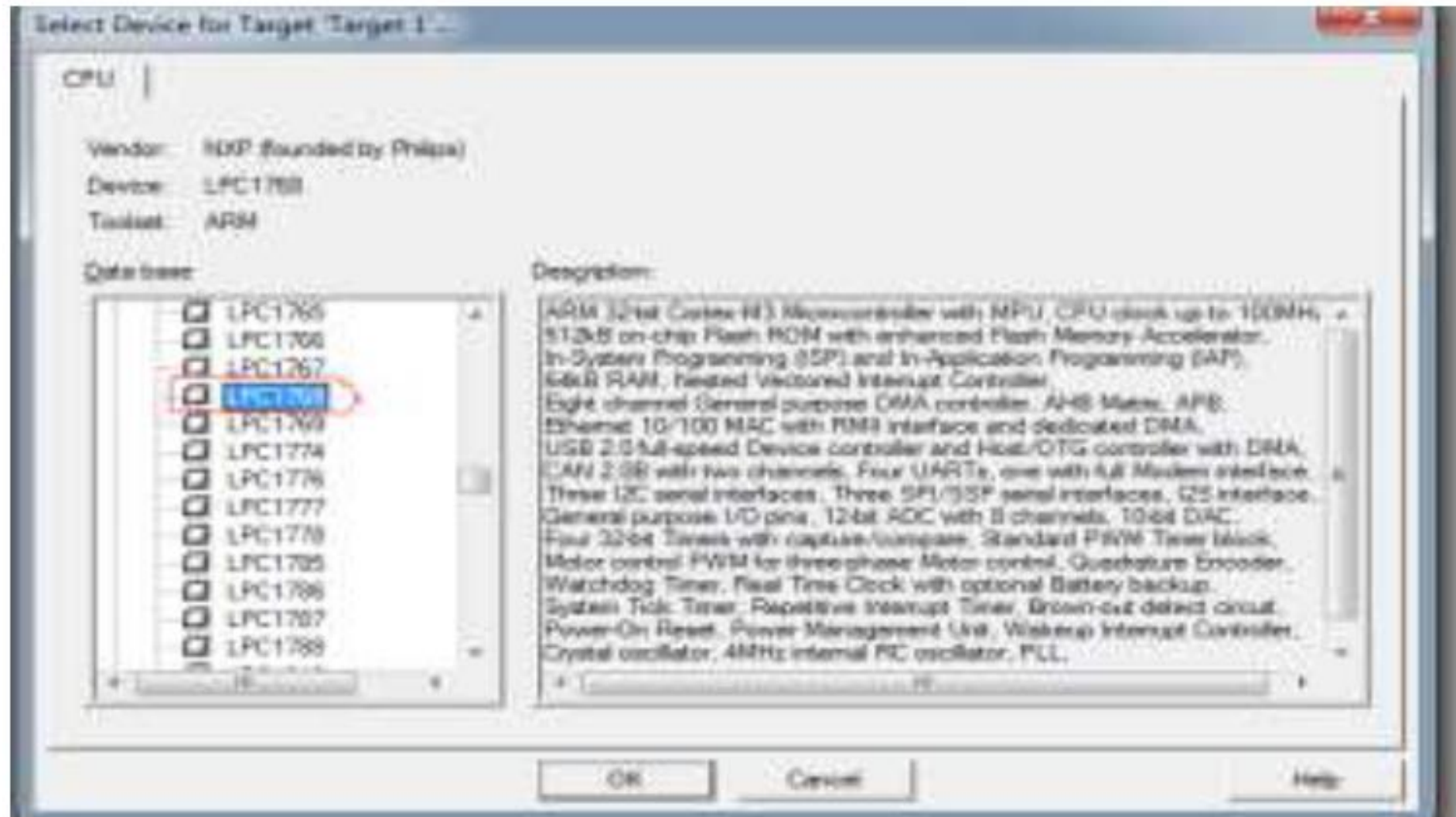
Make sure you click on "NO" for the following pop up window.
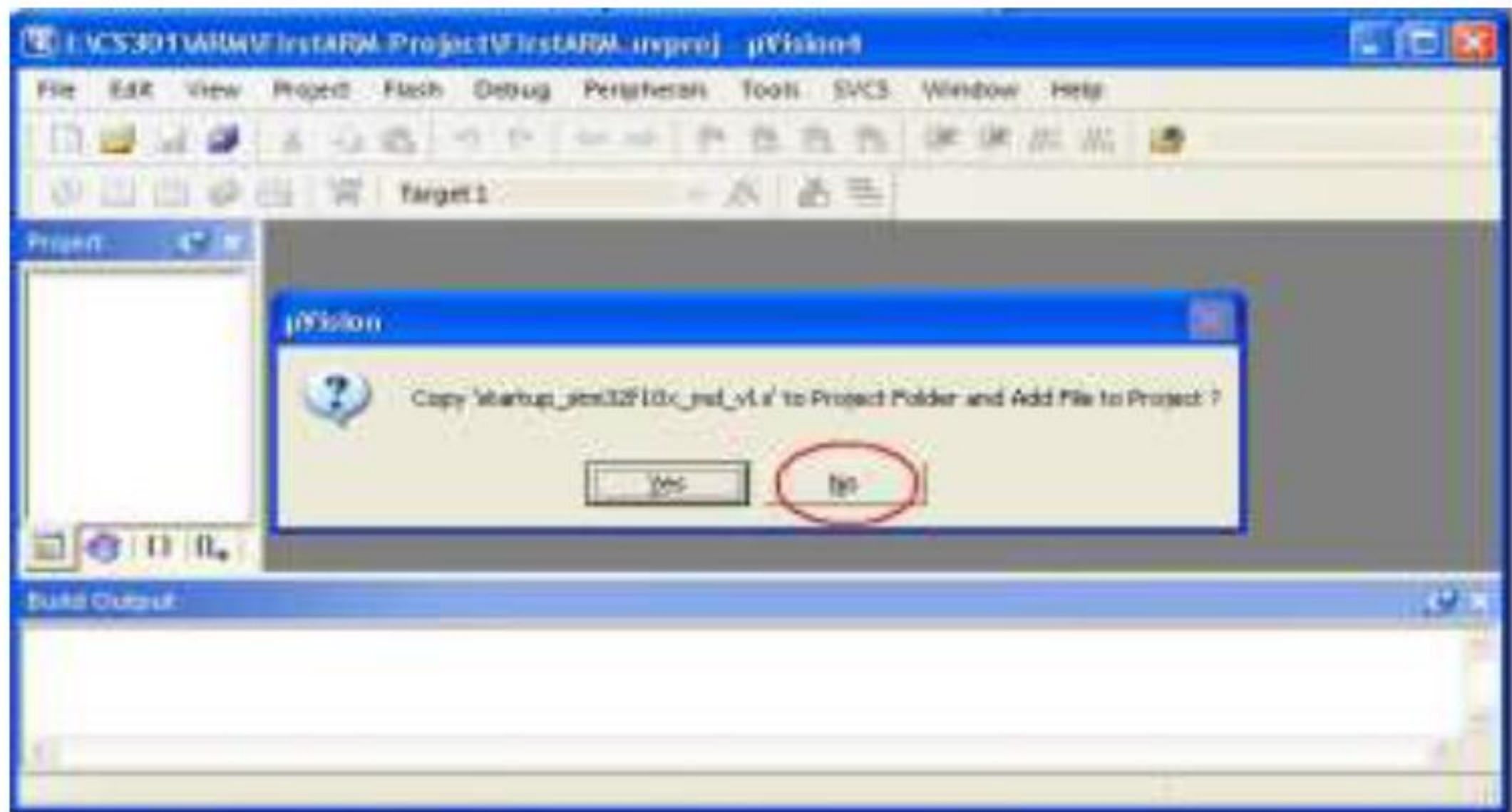
Step 3: Create Source File

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.

File   Edit   View   Project   Flash   Debug   Peripherals   Tools   SVCS   Window   Help

Target 1

Project

Addition.s

Target 1

```
 1  AREA       RESET, DATA, READONLY
 2      EXPORT __Vectors
 3  __Vectors
 4      DCD 0X10001000
 5      DCD Reset_Handler
 6      ALIGN
 7      AREA       mycode, CODE, READONLY
 8      ENTRY
 9      EXPORT Reset_Handler
10  Reset_Handler
11      MOV R0,#3
12      MOV R1, #10
13      ADD R0, R0, R1
14      END
```

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.


```
D:\Student\A21\First.uvproj - µVision4
File    Edit    View    Project    Flash    Debug    Peripherals    Tools    SVCS    Window    Help

Target 1

Project                                    Addition.s
Target 1                              1        AREA        RESET,  DATA,  READONLY
  Source Group 1                      2        EXPORT  __Vectors
    Addition.s                        3  __Vectors
                                      4        DCD  0X10001000
                                      5        DCD  Reset_Handler
                                      6        ALIGN
                                      7        AREA        mycode,  CODE,  READONLY
                                      8        ENTRY
                                      9        EXPORT  Reset_Handler
                                     10  Reset_Handler
                                     11        MOV  R0,#10
                                     12        MOV  R1,#3
                                     13        ADD  R0,R1
                                     14        END

Build Output
Build target 'Target 1'
linking...
Program Size: Code=12 RO-data=8 RW-data=0 ZI-data=0
".\First.axf" - 0 Error(s), 0 Warning(s).
```
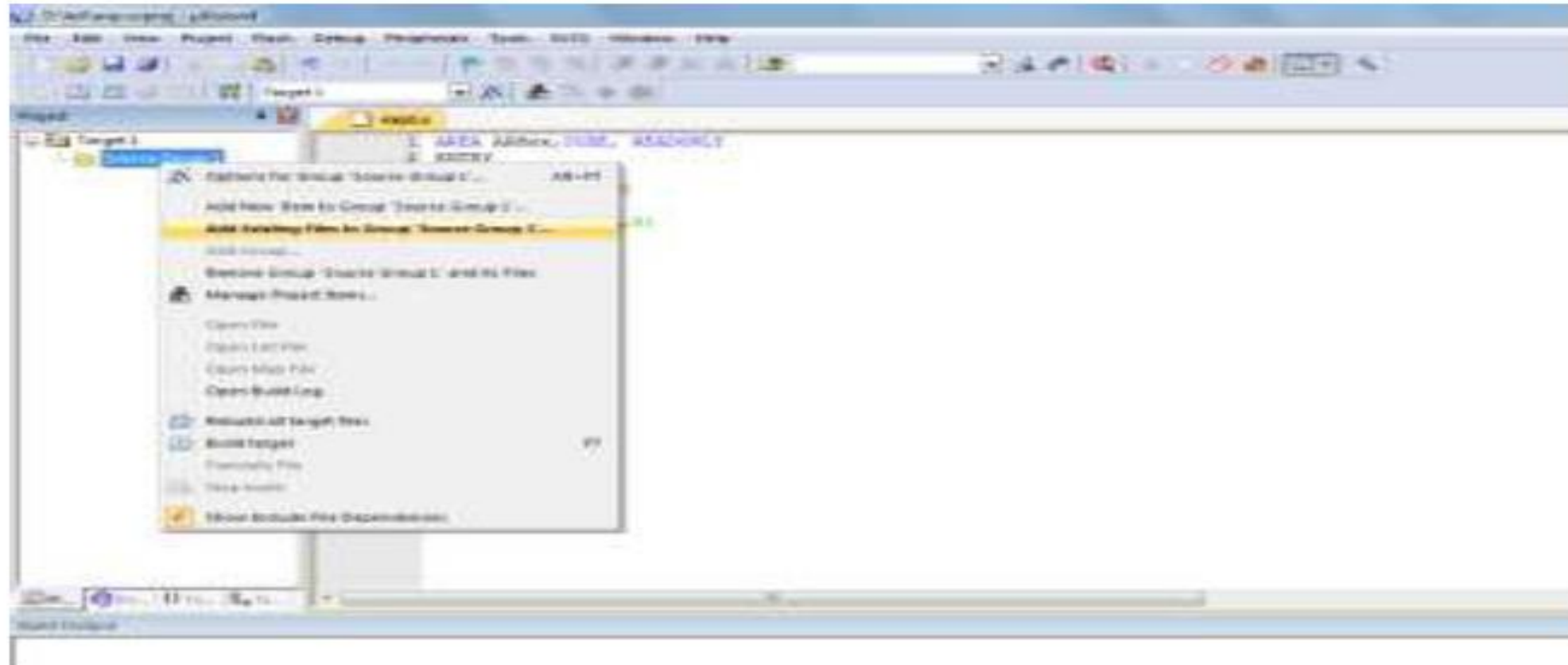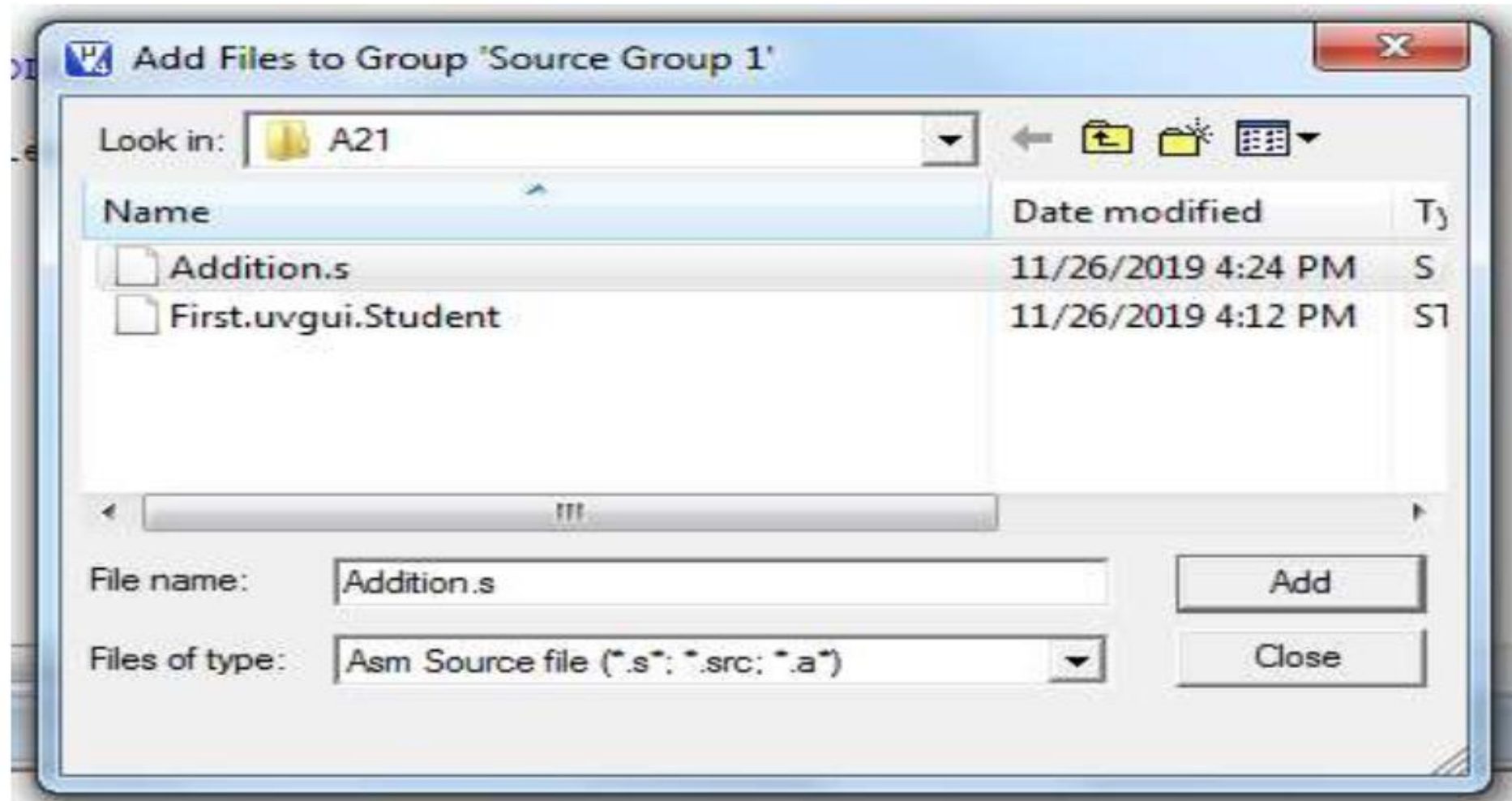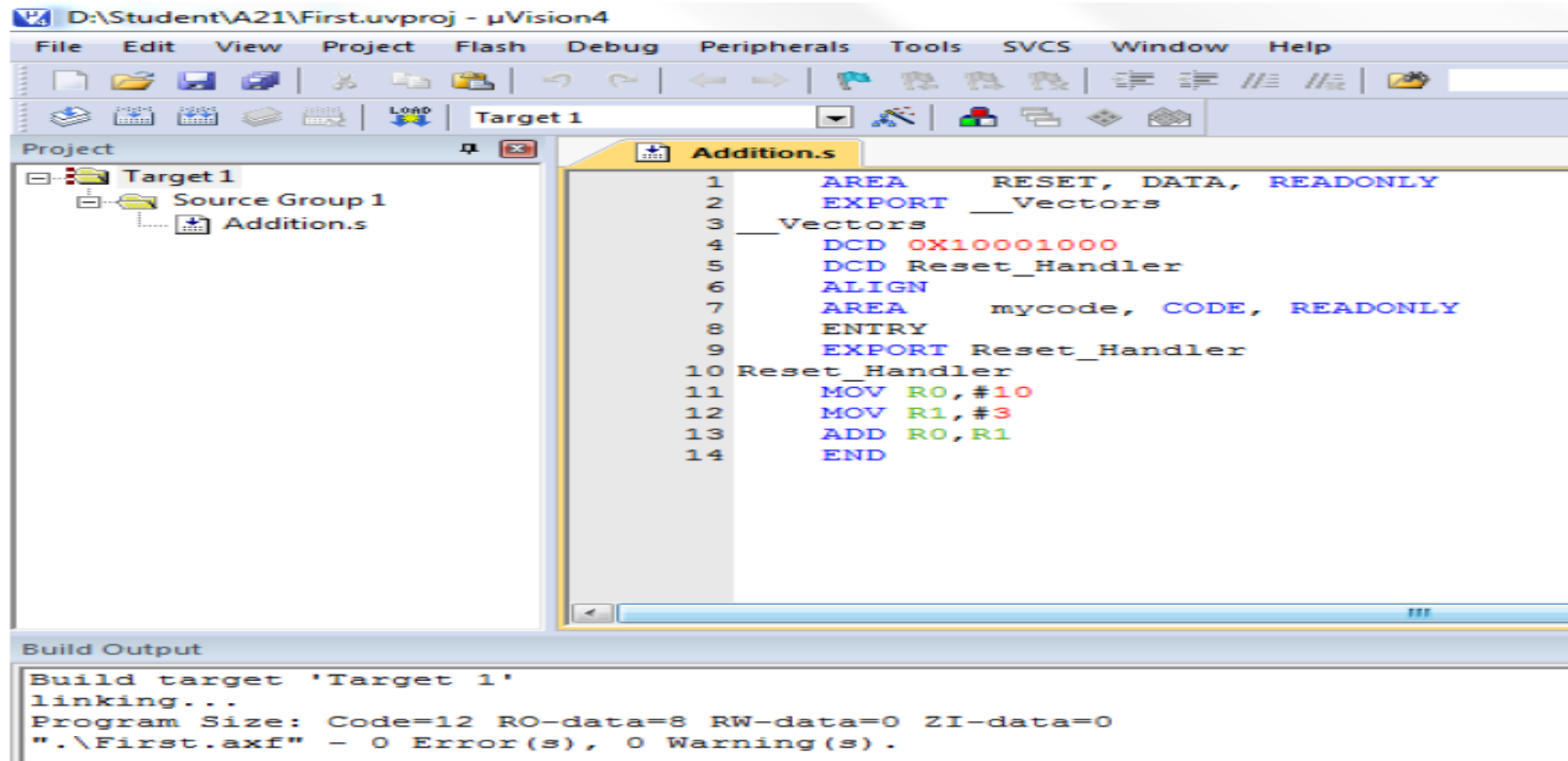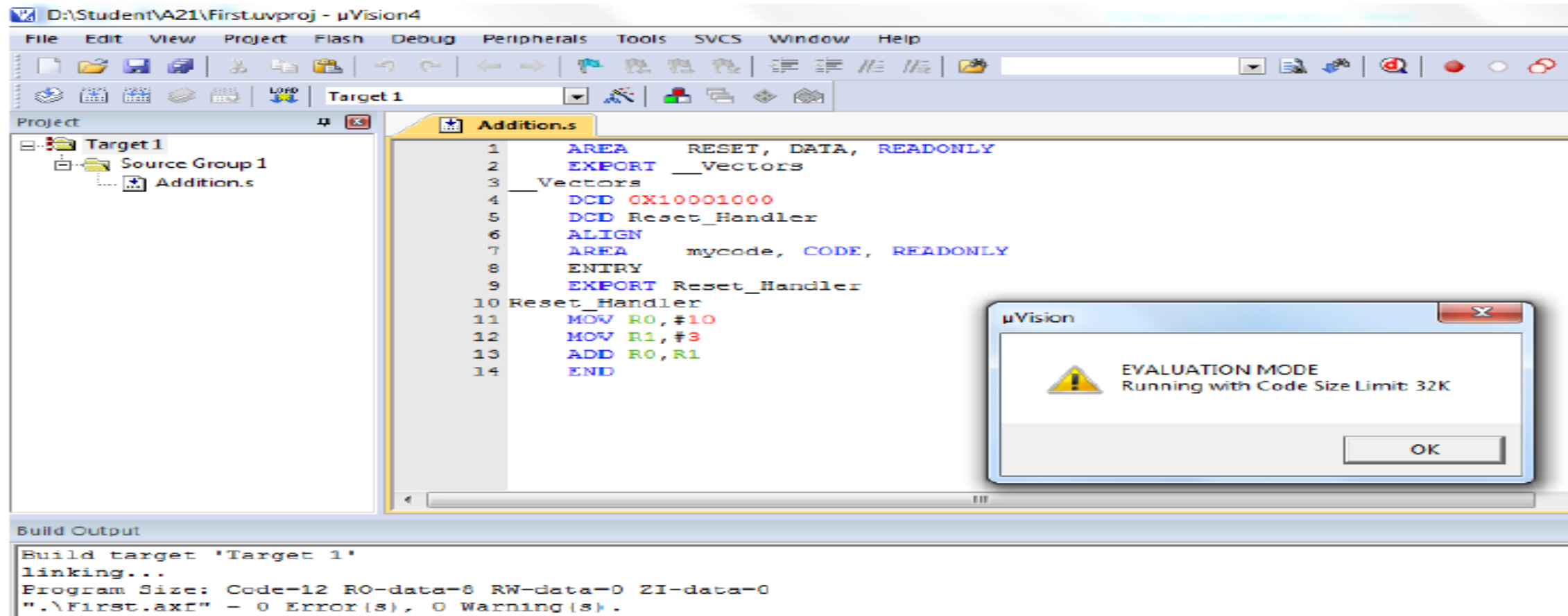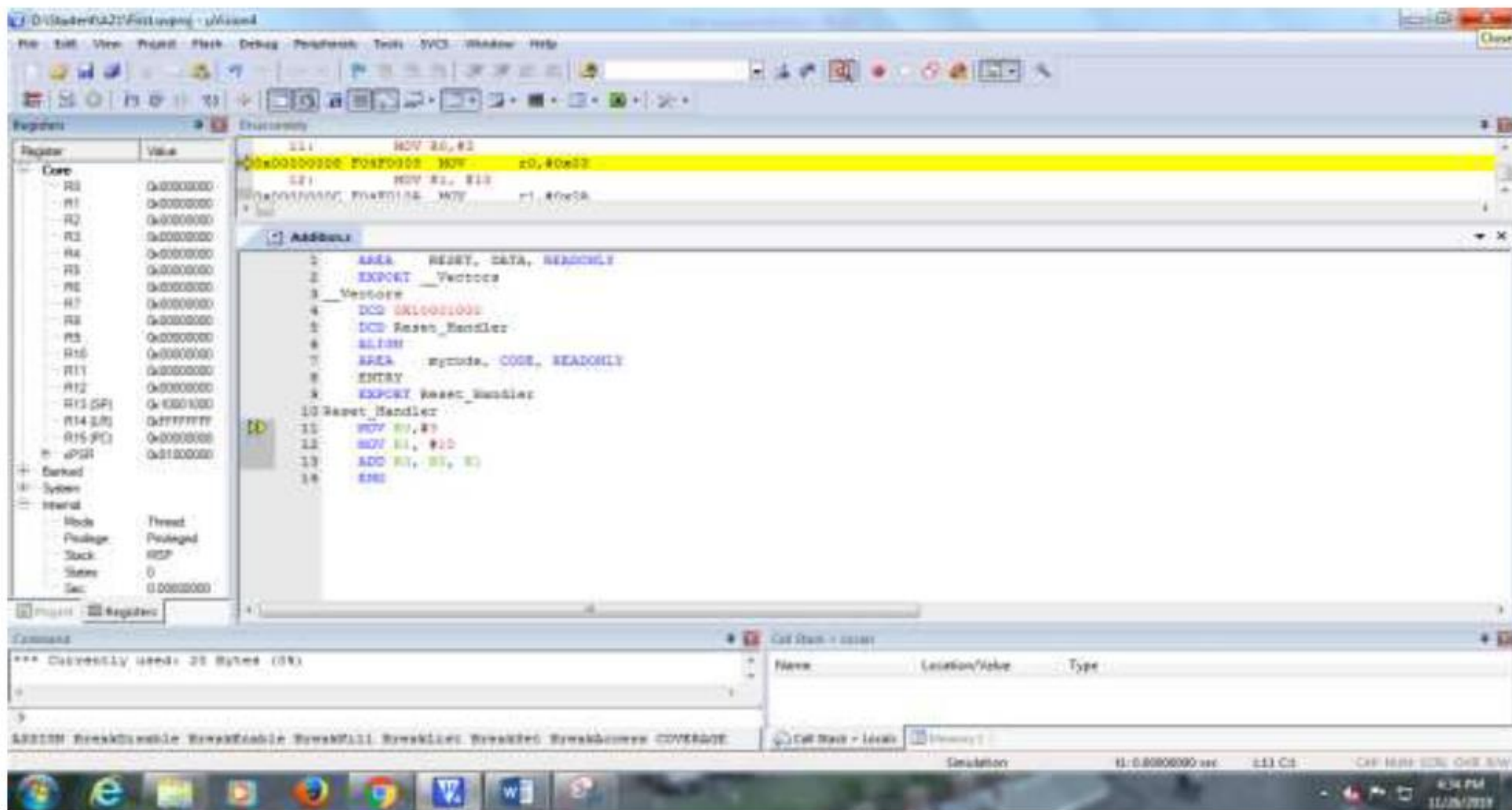
Step 6: Run the program
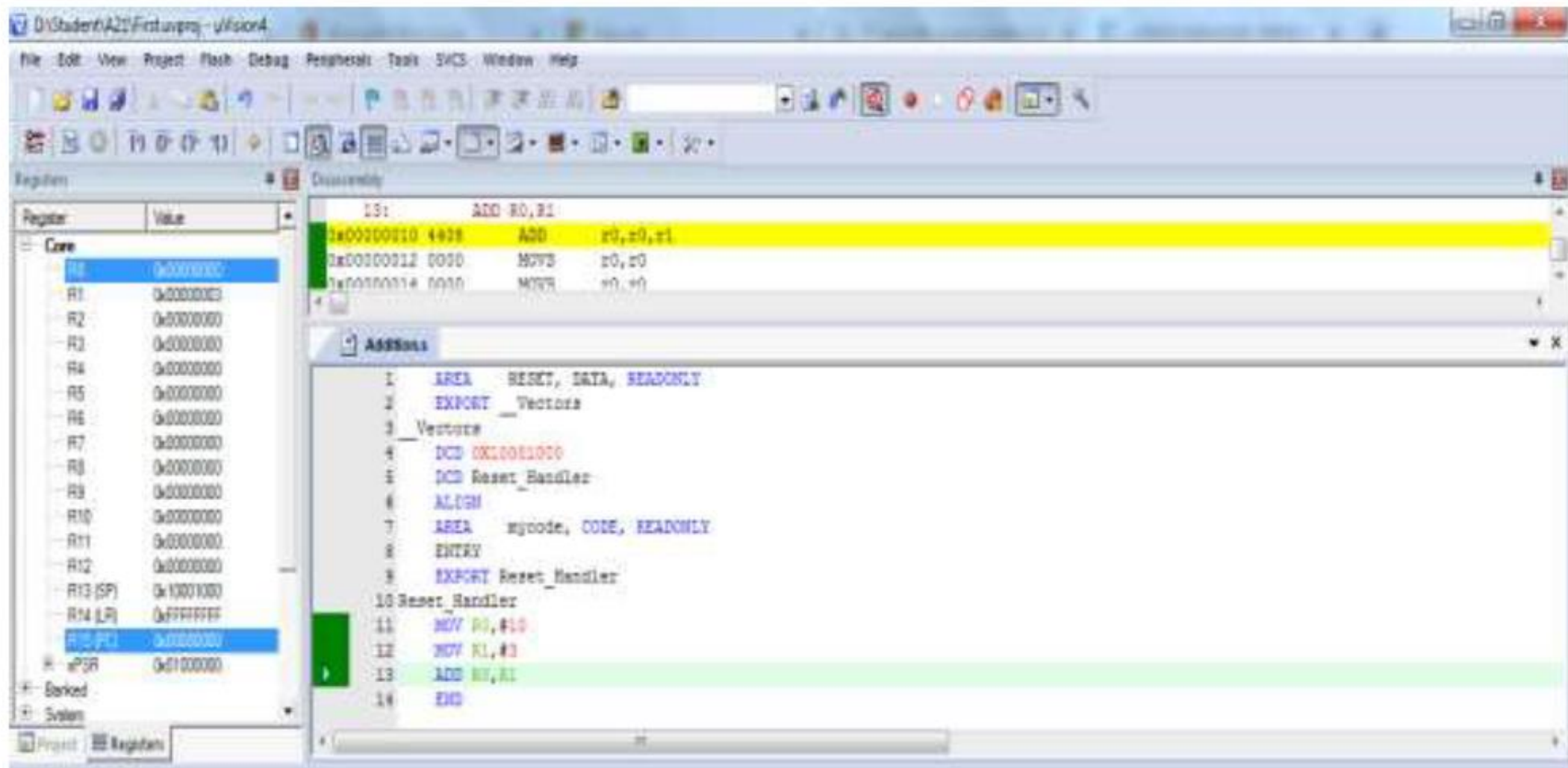
Run the program through "Debug" menu.

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

Open uVision4 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 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 PC is executing the function and want 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

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 REASDONLY 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 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.

ENTRTY:

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
        END    ;End of the program
```

Add two immediate values in the registers and store the result in 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:

```
R0          0x0000000A
R1          0x00000003
R2          0x0000000D
```