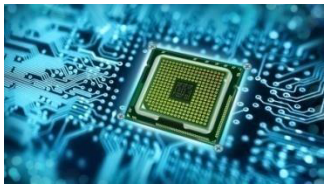


WELCOME TO THE LAB ON

BECE204P

Microprocessors and Microcontrollers Lab

by

Dr Abdul Majeed K. K.

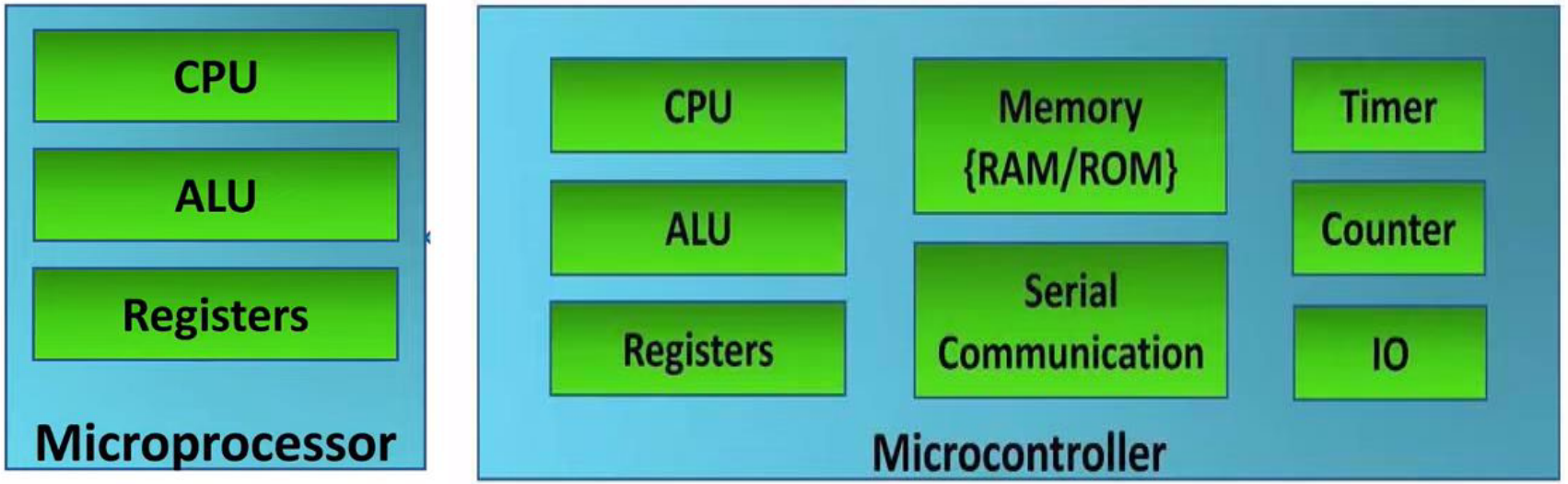
Associate Professor

School of SENSE

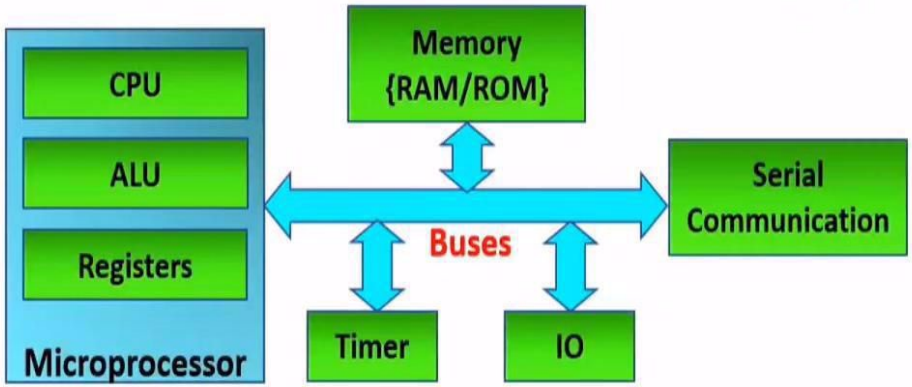
Department of Micro and Nanoelectronics

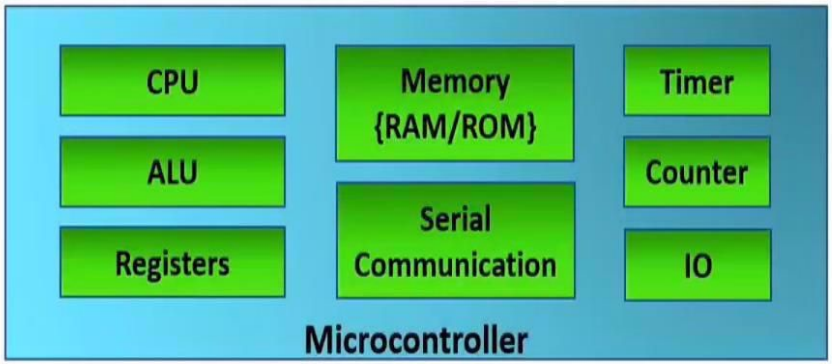
Vellore Institute of Technology, Vellore

Difference between Microprocessor and Microcontroller



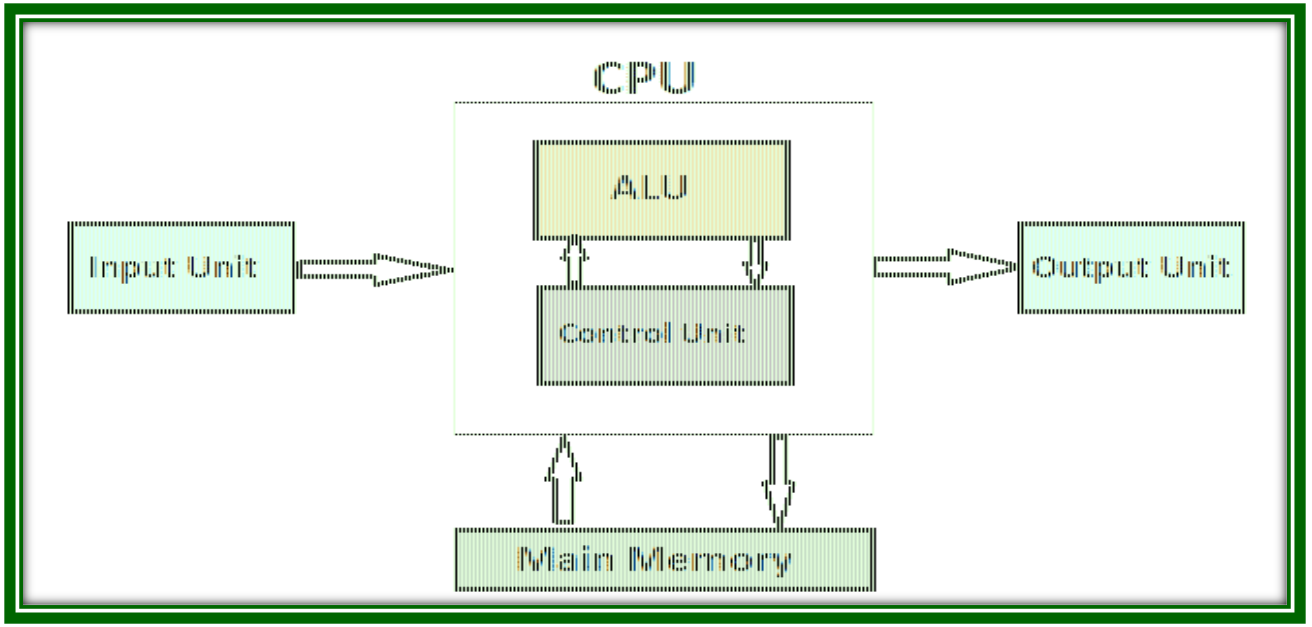
Difference between Microprocessor and Microcontroller





Microprocessor	Microcontroller
❖ It contains only CPU	❖ CPU, Memory, IO, Timer are on single chip.
❖ Designer decides size of ROM, RAM & IO handling.	❖ Fixed size of ROM, RAM & IO handling.
❖ Architecture : Von numen [Mostly].	❖ Architecture : Harvard [Mostly].
❖ It is better in Multi-Tasking.	❖ Relatively weak in Multi-Tasking.
❖ General Purpose. {Like our Main Computer}	❖ Application Specific Purpose. {Embedded System}
❖ High speed and High Cost.	❖ Relatively Low speed and low cost.
❖ It requires more hardware to be interfaced.	❖ It requires less hardware to be interfaced.
❖ High Power Consumption	❖ Low Power Consumption
❖ Does not support bit addressability [Mostly].	❖ Support bit addressability [Mostly].
❖ Examples : 8085, 8086, Core i3, Core i5, Corei7, AMD Processor.	❖ Examples : 8051, AVR, PIC, ARM

Digital Computer Architecture

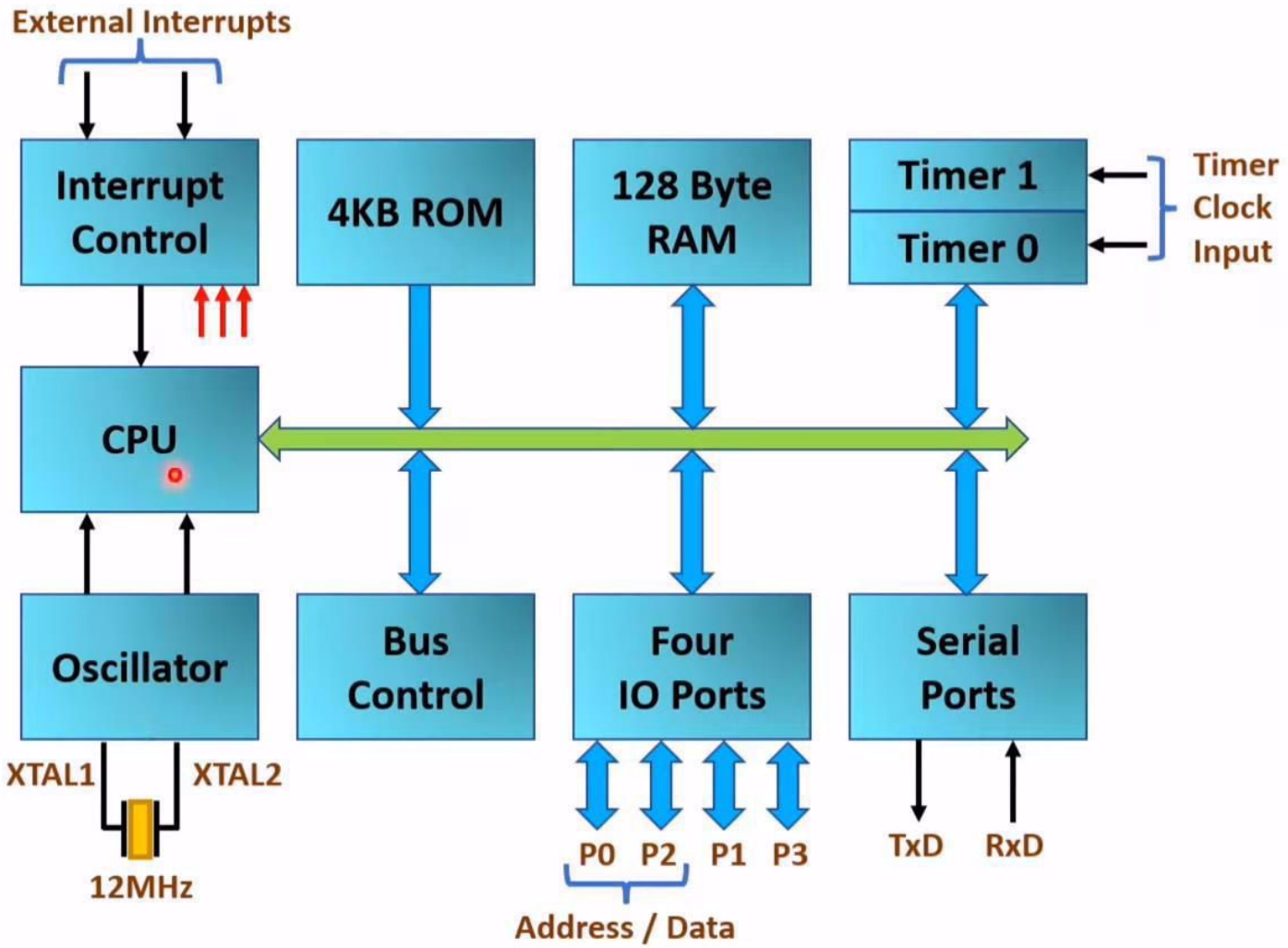


Micro Processor (µp)=Central Processing Unit

"*The Microprocessor is a programmable integrated device that has computing and decision making capability similar to that of the central processing unit (CPU) of a computer*"

- **Is a brain of microcomputer**
- **It is a single chip which can process data.**
- **It controls all components**
- **It execute sequence of instructions**
- **Microprocessor fetch, decode and execute the instructions**

Block Diagram of 8051 μC



Course Code	Course Title	L	T	P	C
BECE204P	Microprocessors and Microcontrollers Lab	0	0	2	1
Pre-requisite	BECE102L	Syllabus version			
		1.0			

Course Objectives

1. To familiarize the students with assembly language programming using microprocessor and microcontroller.
2. To familiarize the students with Embedded C language programming using microcontroller.
3. To interface peripherals and I/O devices with the microcontroller and microprocessor.

Course Outcome

Student will be able to

1. Showcase the skill, knowledge and ability of programming microcontroller and microprocessor using its instruction set.
2. Expertise with microcontroller and interfaces including general purpose input/ output, timers, serial communication, LCD, keypad and ADC.

Indicative Experiments [Experiments using 8086/8051/ARM]		
1	Assembly language programming of Arithmetic/logical operations.	**6 hours**
2	Assembly language programming of memory operations.	**4 hours**
3	Assembly language programming/ Embedded C programming for interfacing the peripherals: General purpose input/ output, timers, serial communication, LCD, keypad and ADC.	**10 hours**
4	Hardware implementation of peripheral interfacing: General purpose input/ output, timers, serial communication, LCD, keypad and ADC.	**10 hours**
	Total Laboratory Hours	**30 hours**

Evaluation Scheme

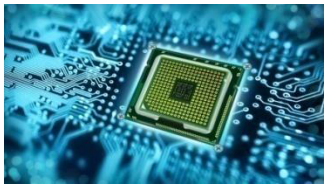
For Lab

5 tasks, each task is evaluated for 12 Marks
(Assembly code with simulation in Keil Software-5 Marks, Output - 3 Marks, Report with plagiarism and output shown before due date -2 Marks)

Lab tasks (Indicative)

1. **ALP**
2. **Ports- Digital**
3. **Timer and Counter**
4. **Serial communication and interrupt**
5. **Hardware implementation of peripheral interfacing**

THANK YOU



WELCOME TO THE LAB ON

BECE204P

Microprocessors and Microcontrollers

Lab2

by

Dr Abdul Majeed K. K.

Associate Professor

School of SENSE

Department of Micro and Nanoelectronics

Vellore Institute of Technology, Vellore

Instructions to use KEIL SIMULATOR

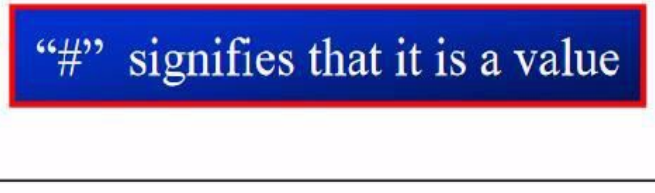
```
ORG 000h              ; Origin directive, setting the starting address of the program to 0000h

MOV A, #50h           ; Move immediate value 50h into register A
MOV B, #40h           ; Move immediate value 40h into register B
ADD A, B              ; Add the contents of registers A and B, result stored in A

HALT:  SJMP HALT      ; Label for the HALT instruction
                      ; Unconditional jump (infinite loop), causing the program to halt
END                   ; End directive, indicating the end of the program
```

MOV Instruction

```
MOV     Destination,    Source                  ;copy source to dest
```

The instruction tells the CPU to move (in reality, COPY) the source operand to the destination operand

MOV Instruction




```
MOV   A,#55H     ;load value 55H into reg. A
MOV   R0,A       ;copy contents of A into R0
                 ;(now A=R0=55H)
MOV   R1,A       ;copy contents of A into R1
                 ;(now A=R0=R1=55H)
MOV   R2,A       ;copy contents of A into R2
                 ;(now A=R0=R1=R2=55H)
MOV   R3,#95H    ;load value 95H into R3
                 ;(now R3=95H)
MOV   A,R3       ;copy contents of R3 into A
                 ;now A=R3=95H
```

MOV Instruction

Notes on programming

- `MOV A, #23H`
- `MOV R5, #0F9H`

If it's not preceded with #, it means to load from a memory location

Add a 0 to indicate that F is a hex number and not a letter

MOV Instruction

Notes on programming

- "MOV A, #5", the result will be A=05; i.e., A = 00000101 in binary

ADD Instruction

```
ADD A, source      ;ADD the source operand
                   ;to the  accumulator
```

Source operand can be either a register or immediate data, but the destination must always be register A

ADD Instruction: Example

```
MOV A, #25H      ;load 25H into A
MOV R2, #34H     ;load 34H into R2
ADD A, R2 ;add R2 to Accumulator
                 ;(A = A + R2)
```

ADD Instruction: Example

```
MOV A, #25H       ;load 25H into A
MOV R2, #34H      ;load 34H into R2
ADD A, R2 ;add R2 to Accumulator
                  ;(A = A + R2)
```

```
MOV A, #25H       ;load one operand
          ;into A (A=25H)
ADD A, #34H       ;add the second
                  ;operand 34H to A
```

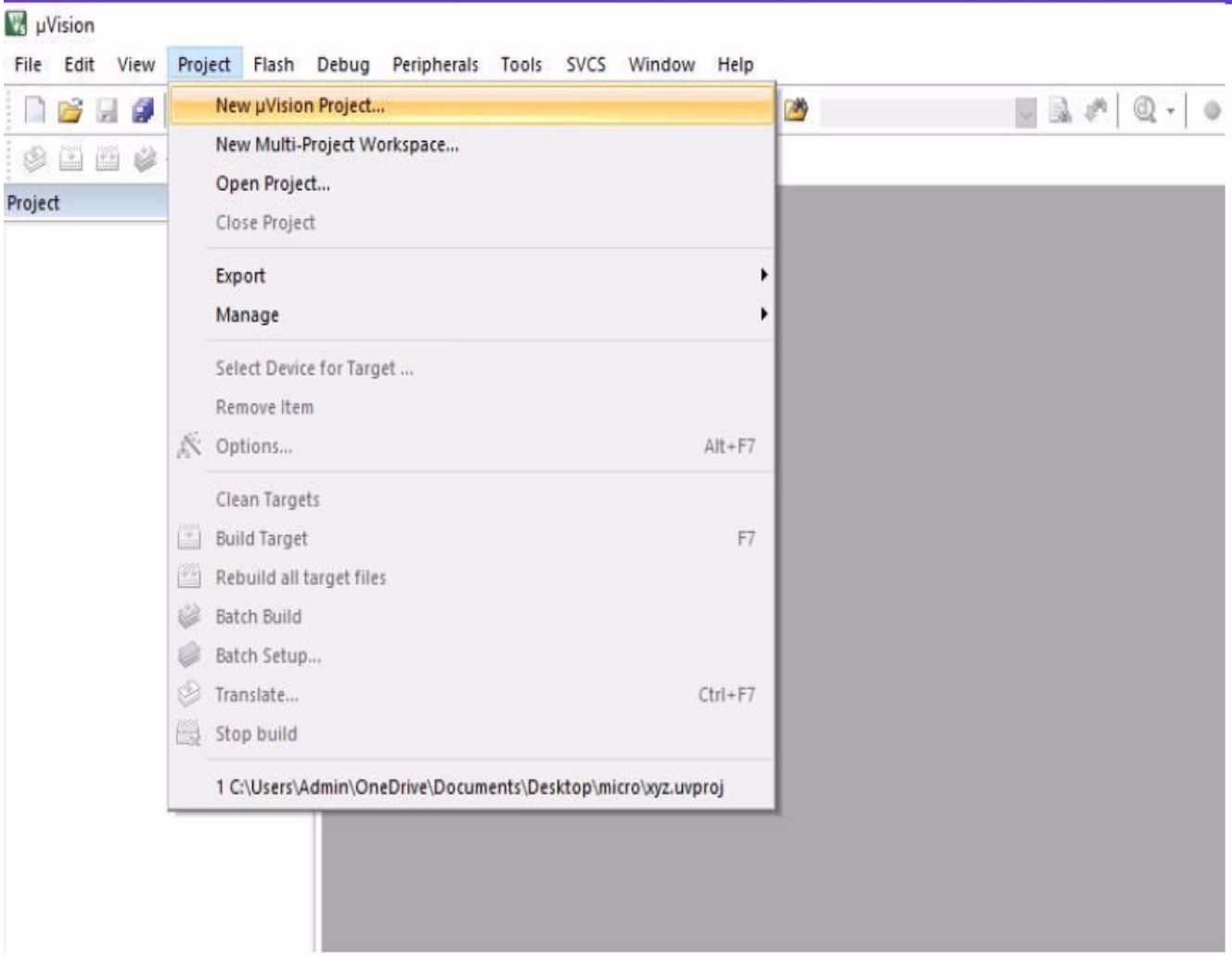
There are always many ways to write the same program, depending on the registers used

Download KEIL Simulator

https://www.keil.com/demo/eval/c51.htm#/DOWNLOAD

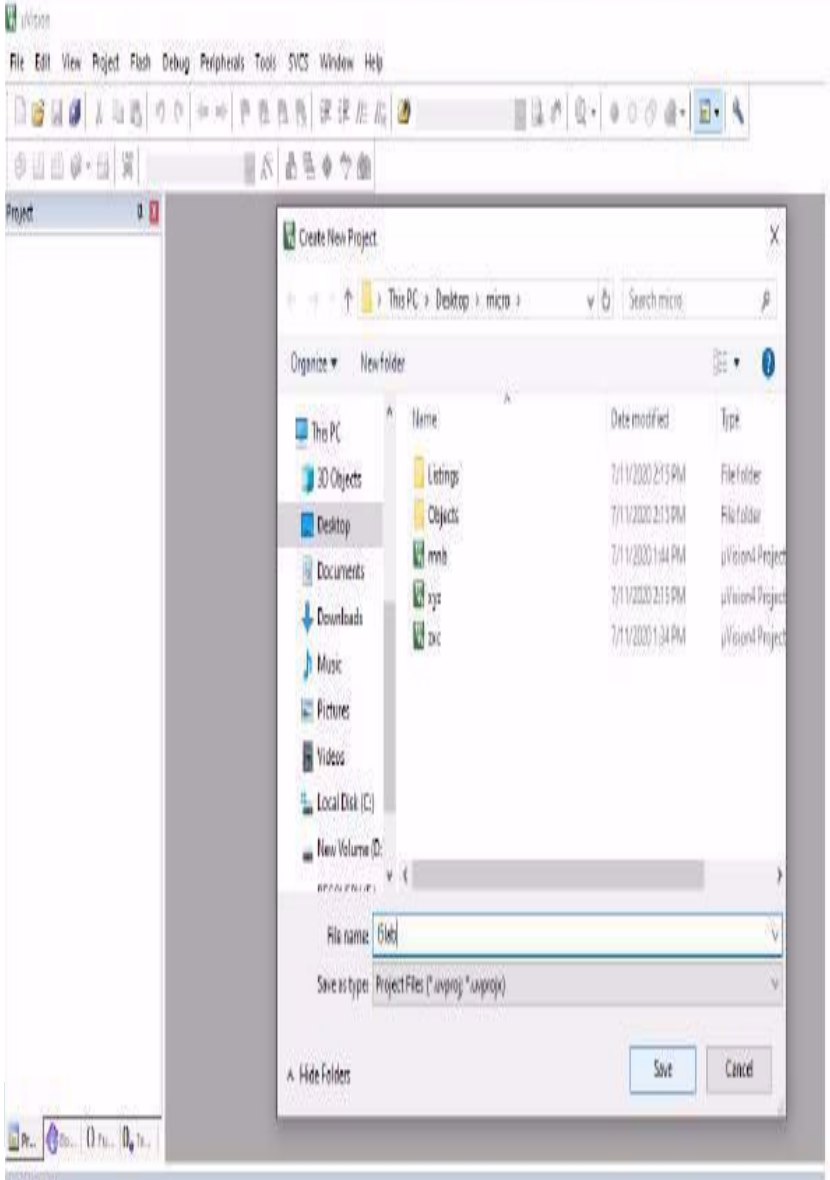
Instructions to use KEIL SIMULATOR

Instructions to use KEIL SIMULATOR

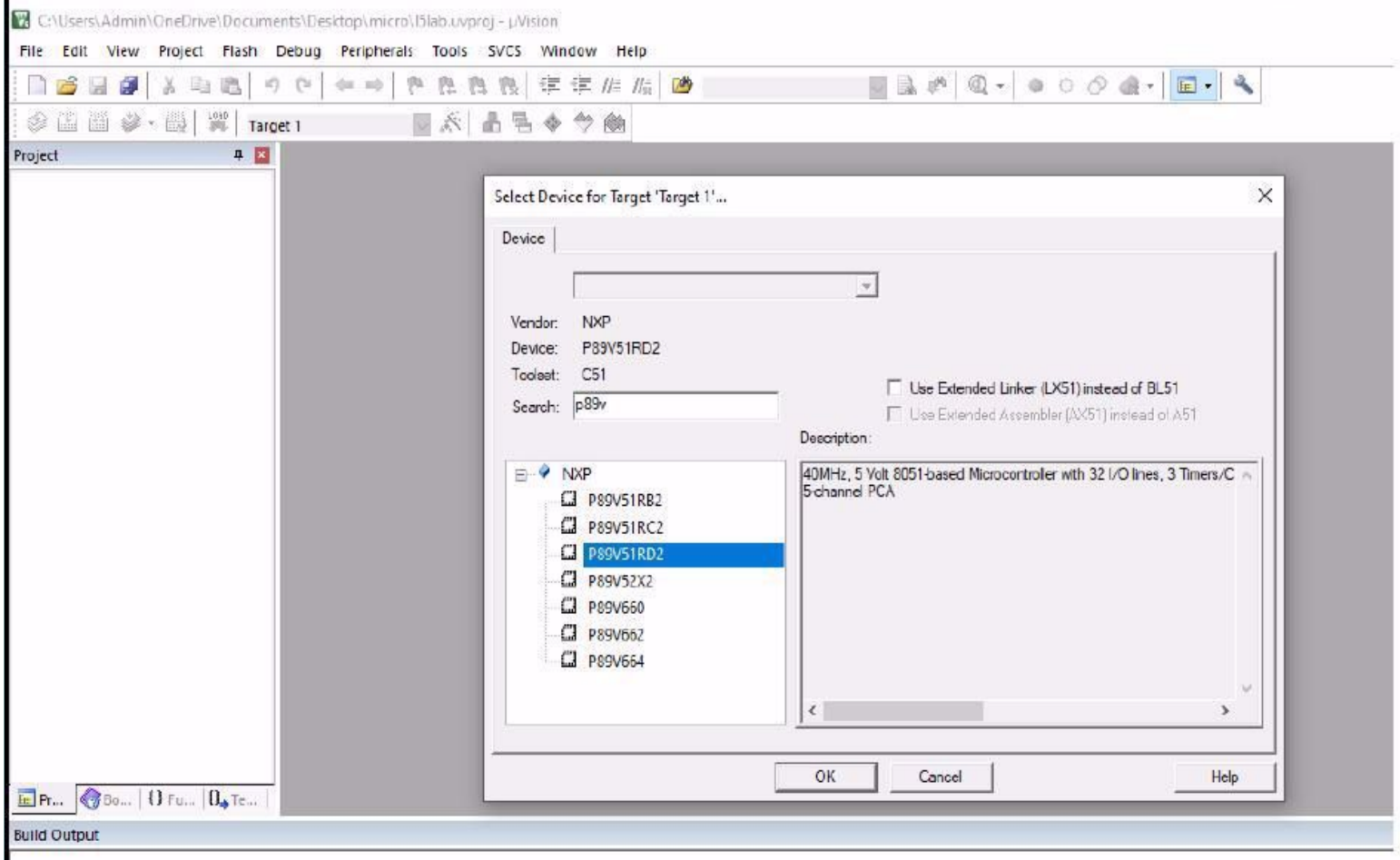


STEP2: Start a new project (File>New>µvision project

Instructions to use KEIL SIMULATOR

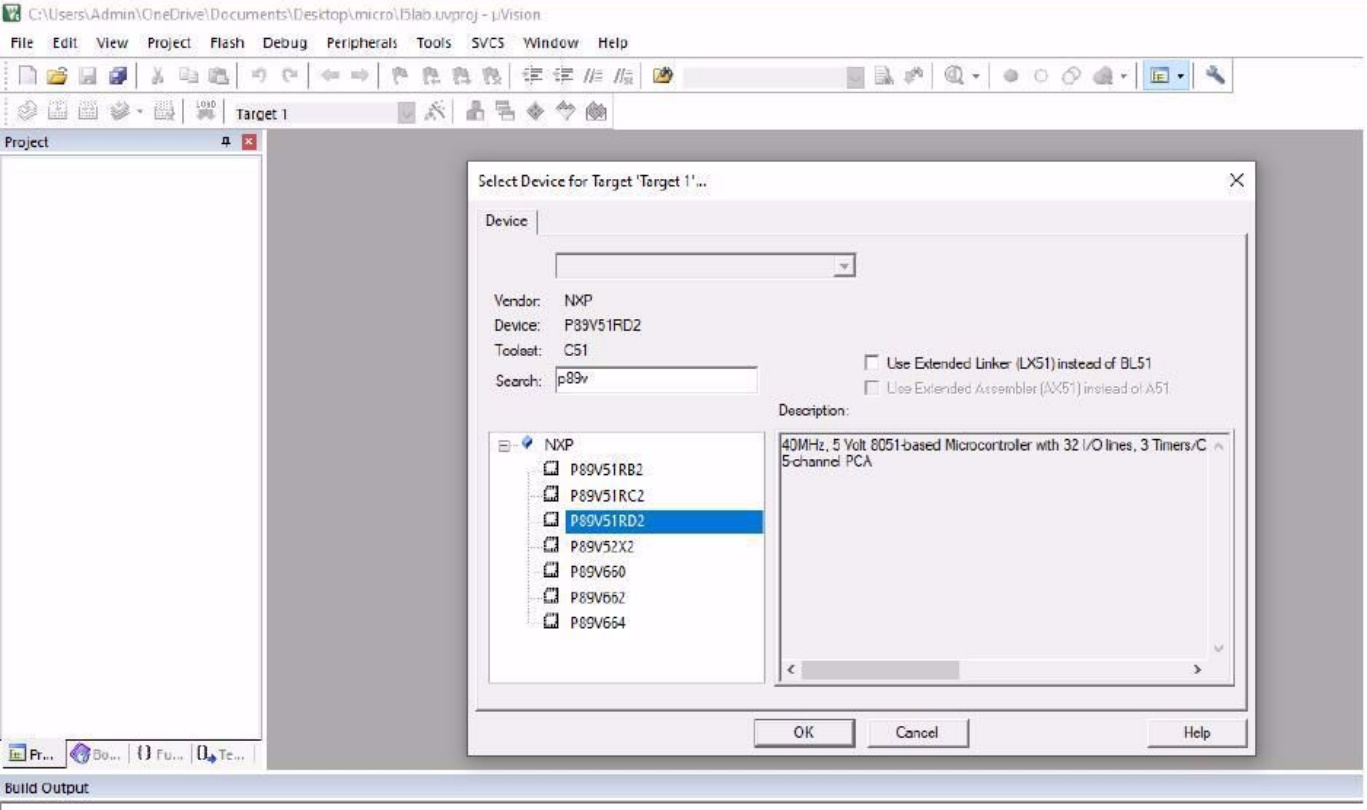


Introduction to Computing



STEP 4: Select Nxp >P89V51RD2

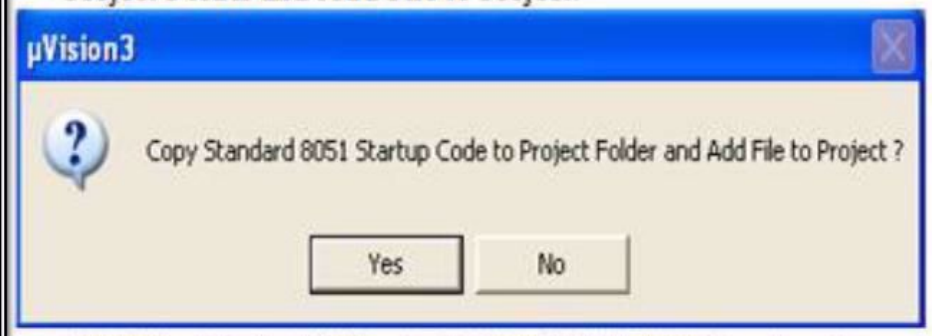
Instructions to use KEIL SIMULATOR



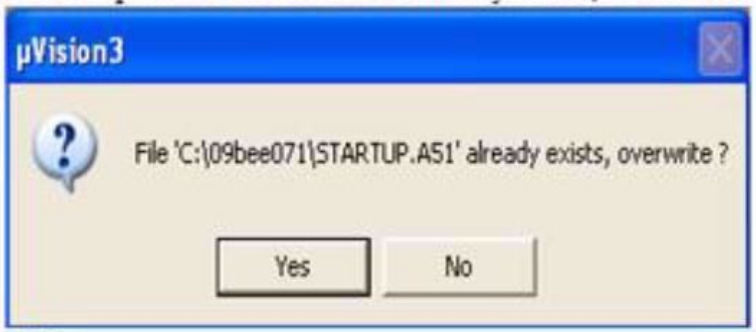
STEP 4: Select Nxp >P89V51RD2

Instructions to use KEIL SIMULATOR

- Click Yes when asked whether to "Copy Standard 8051 Startup Code to Project Folder and Add File to Project."

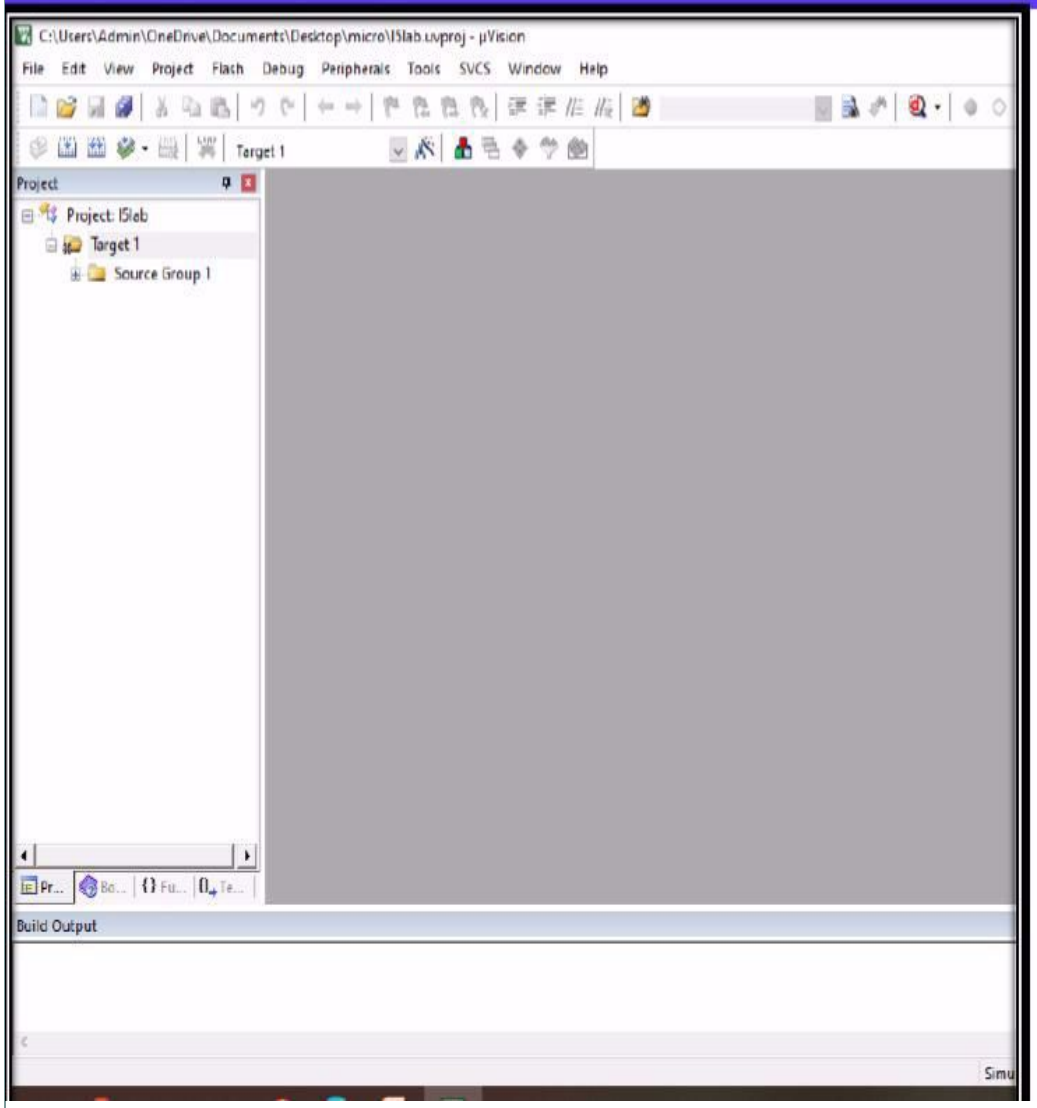


- Click Yes if asked "File <path>STARTUP.A51 already exists, overwrite?



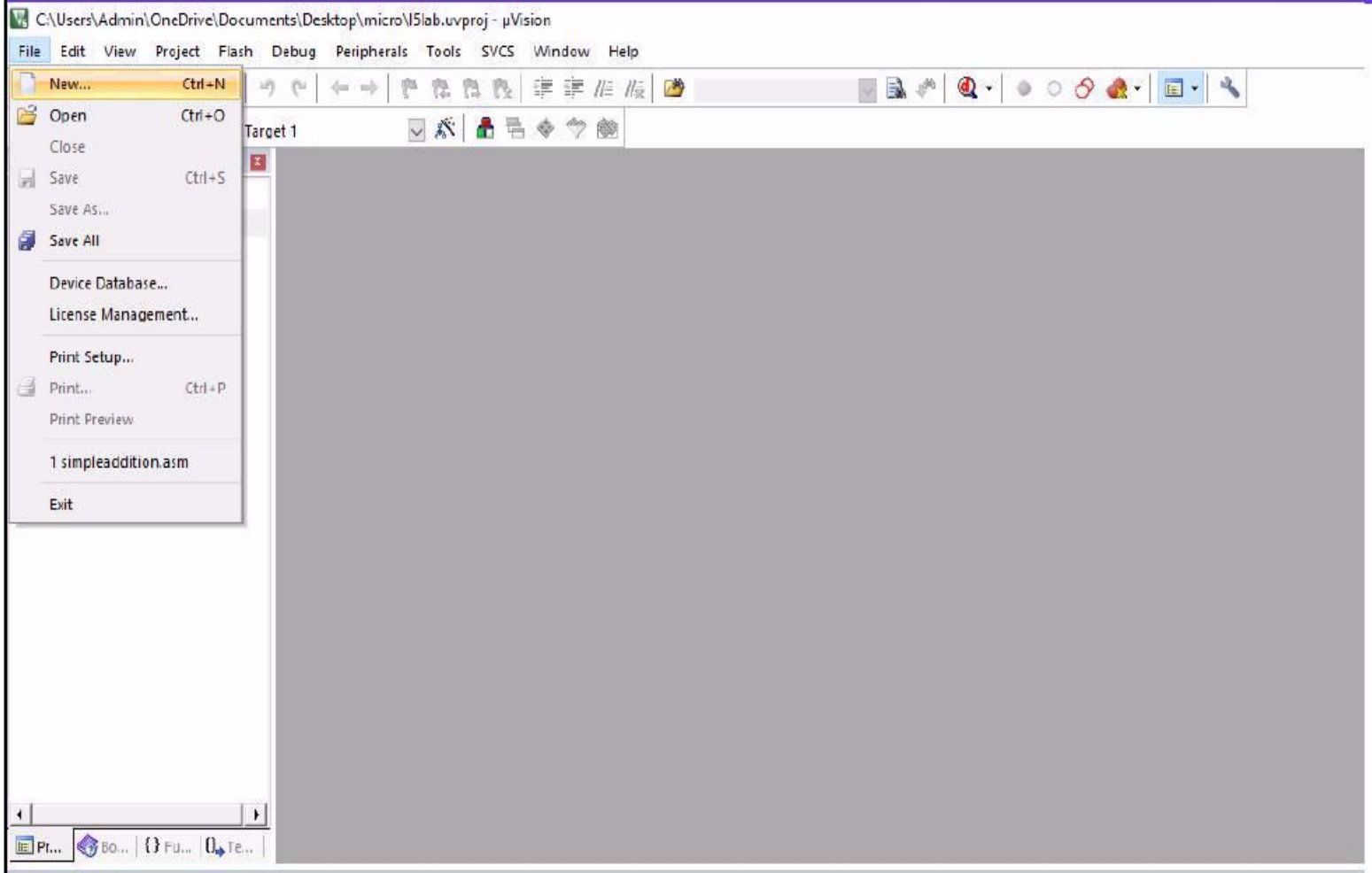
STEP 5 (select :yes)

Instructions to use KEIL SIMULATOR



Blank space will appear

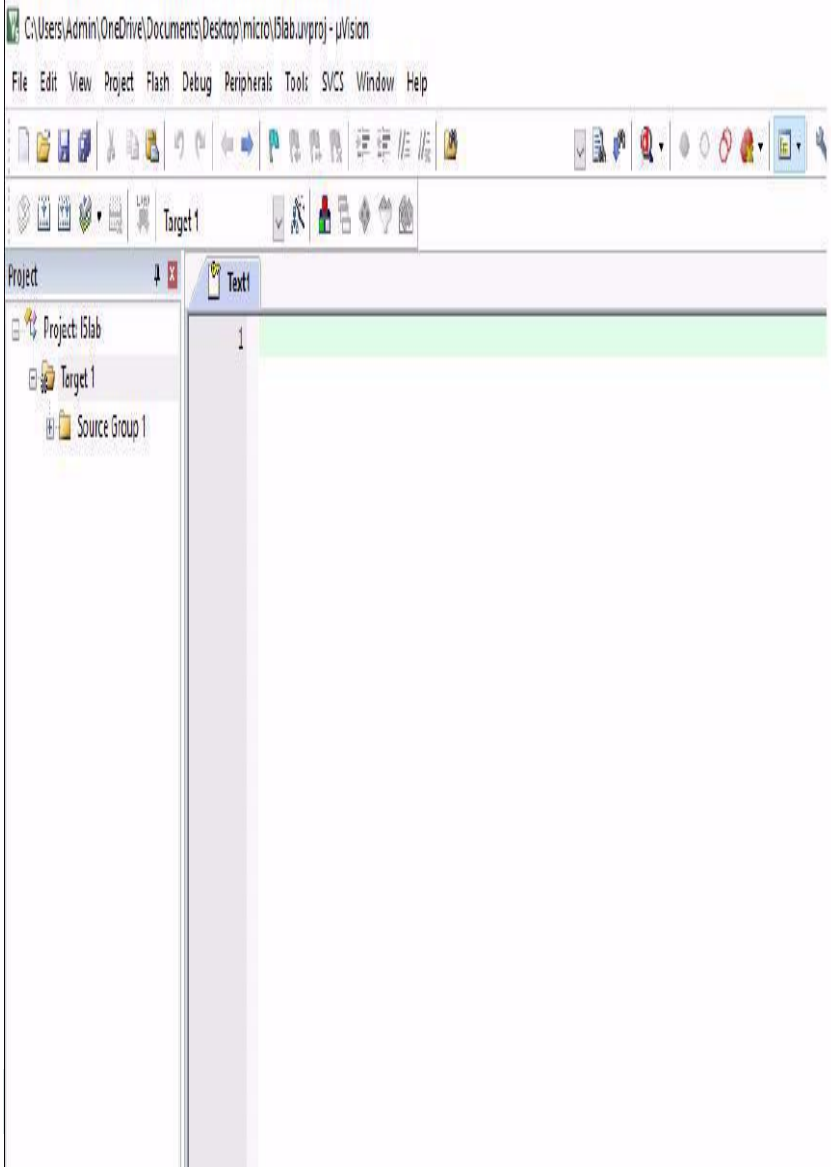
Instructions to use KEIL SIMULATOR



STEP 6: To create source file

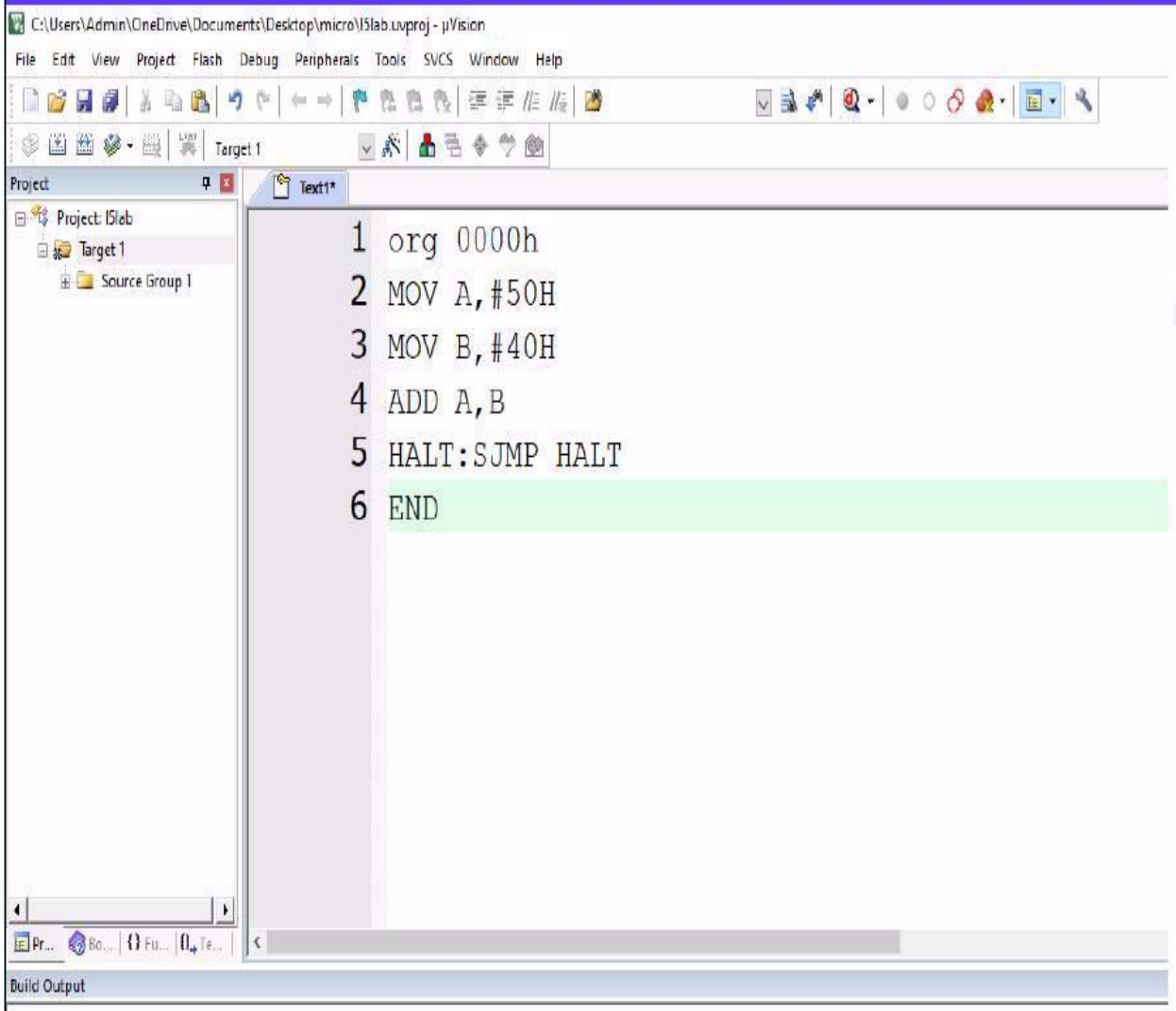
Open a new file (File:New File)

Instructions to use KEIL SIMULATOR



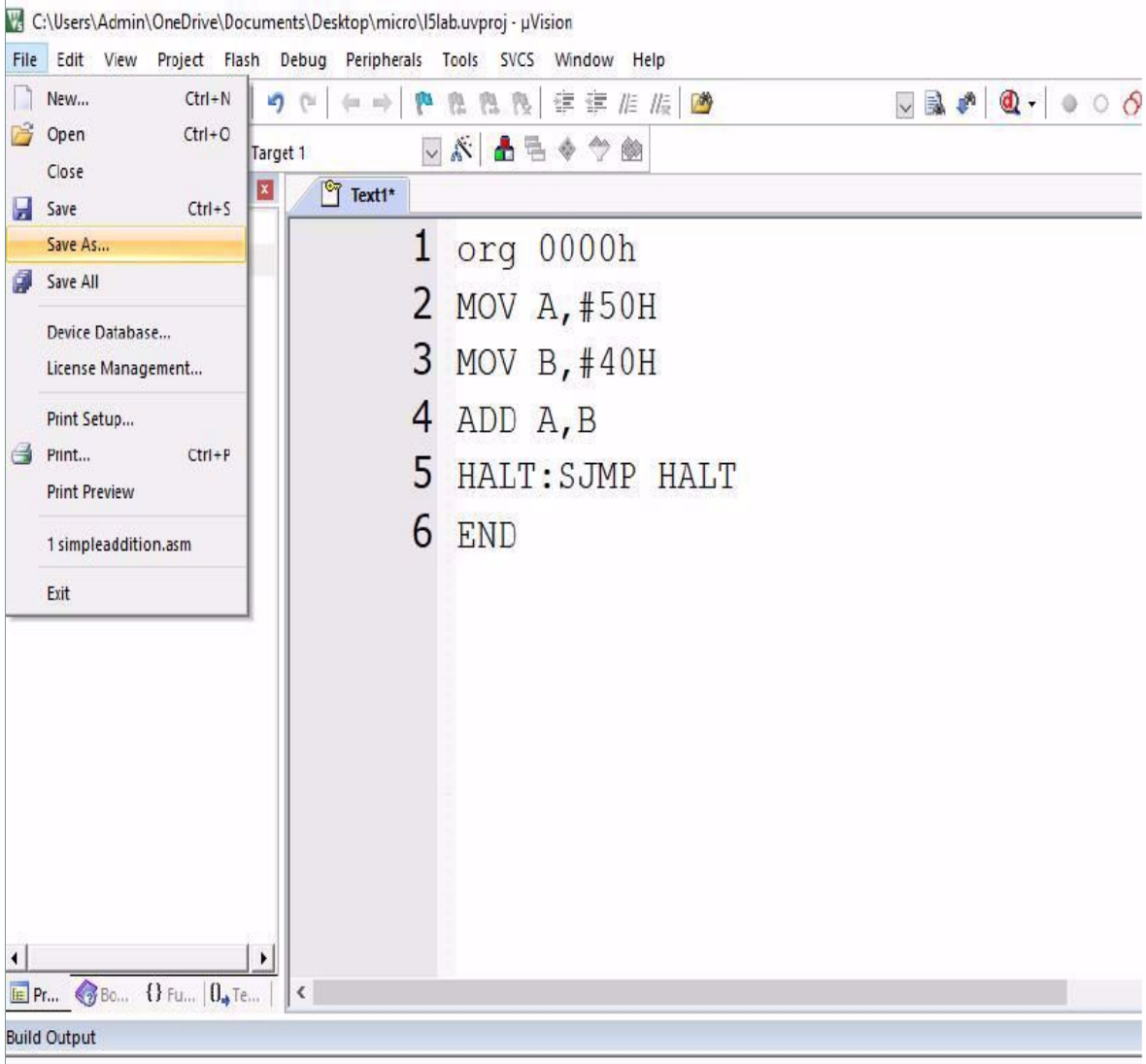
STEP 6 (Contd): A new window will open with blank space to type the code

Instructions to use KEIL SIMULATOR



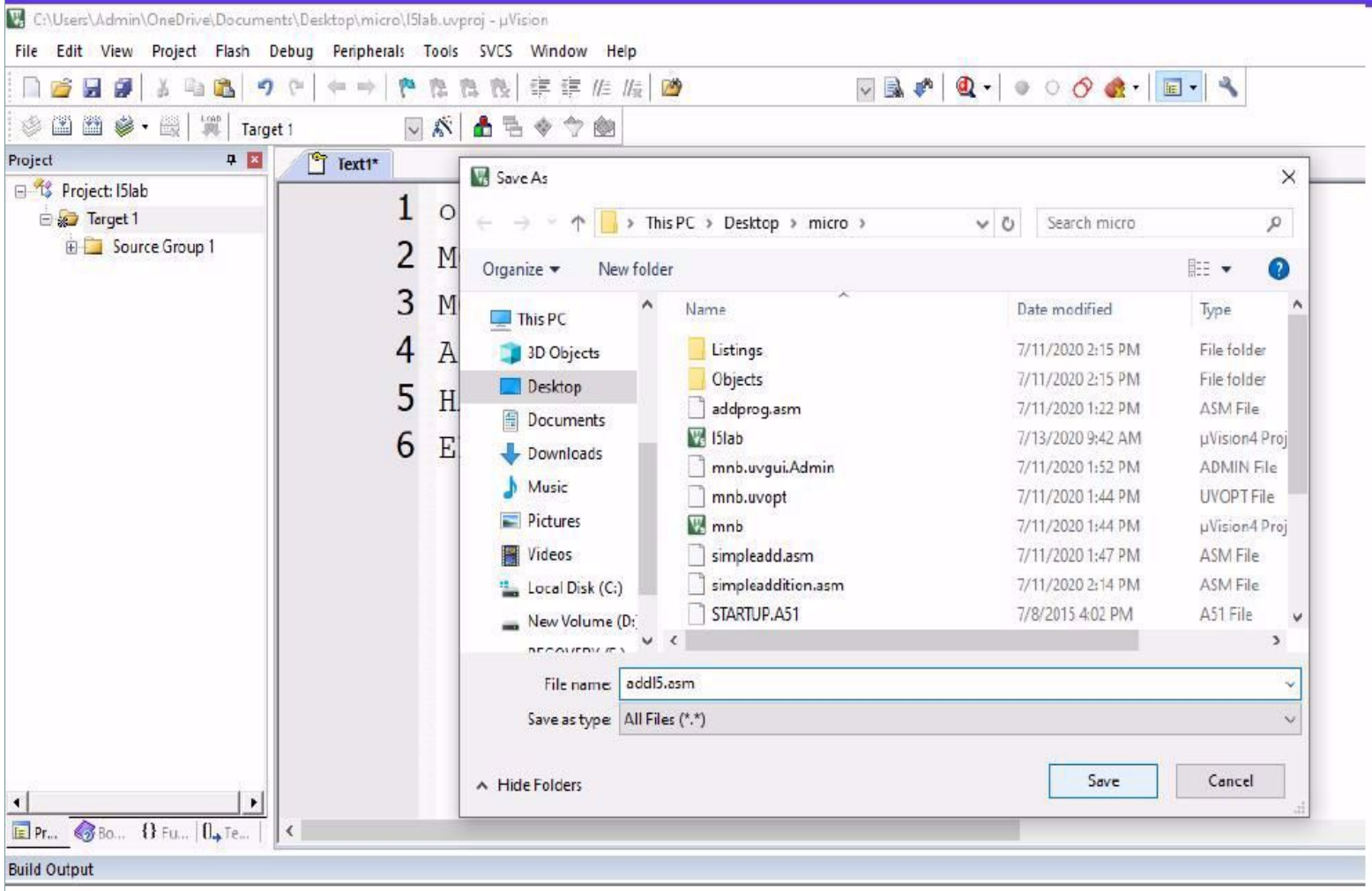
STEP 7: Type the code

Instructions to use KEIL SIMULATOR



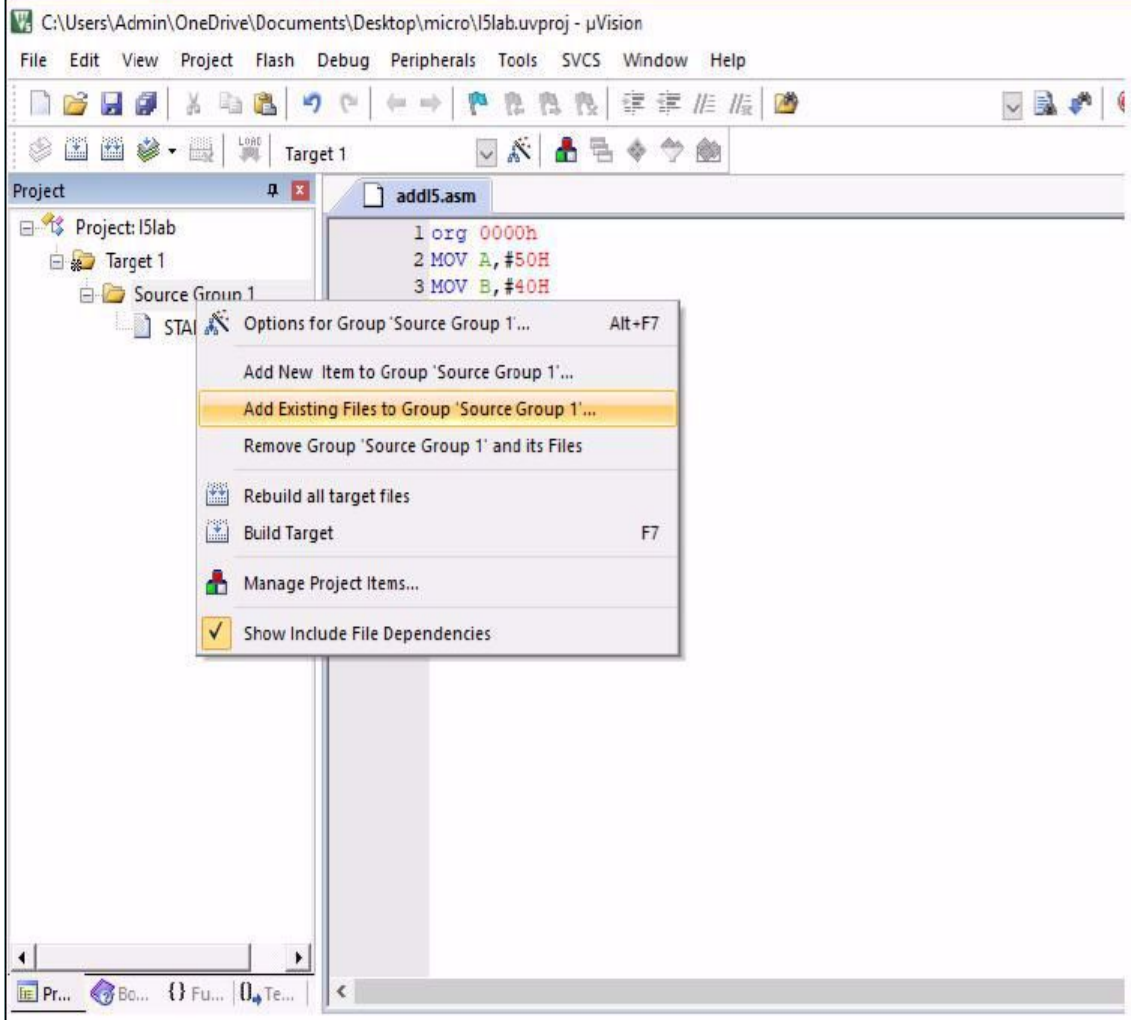
STEP 8: Click file menu and save

Instructions to use KEIL SIMULATOR



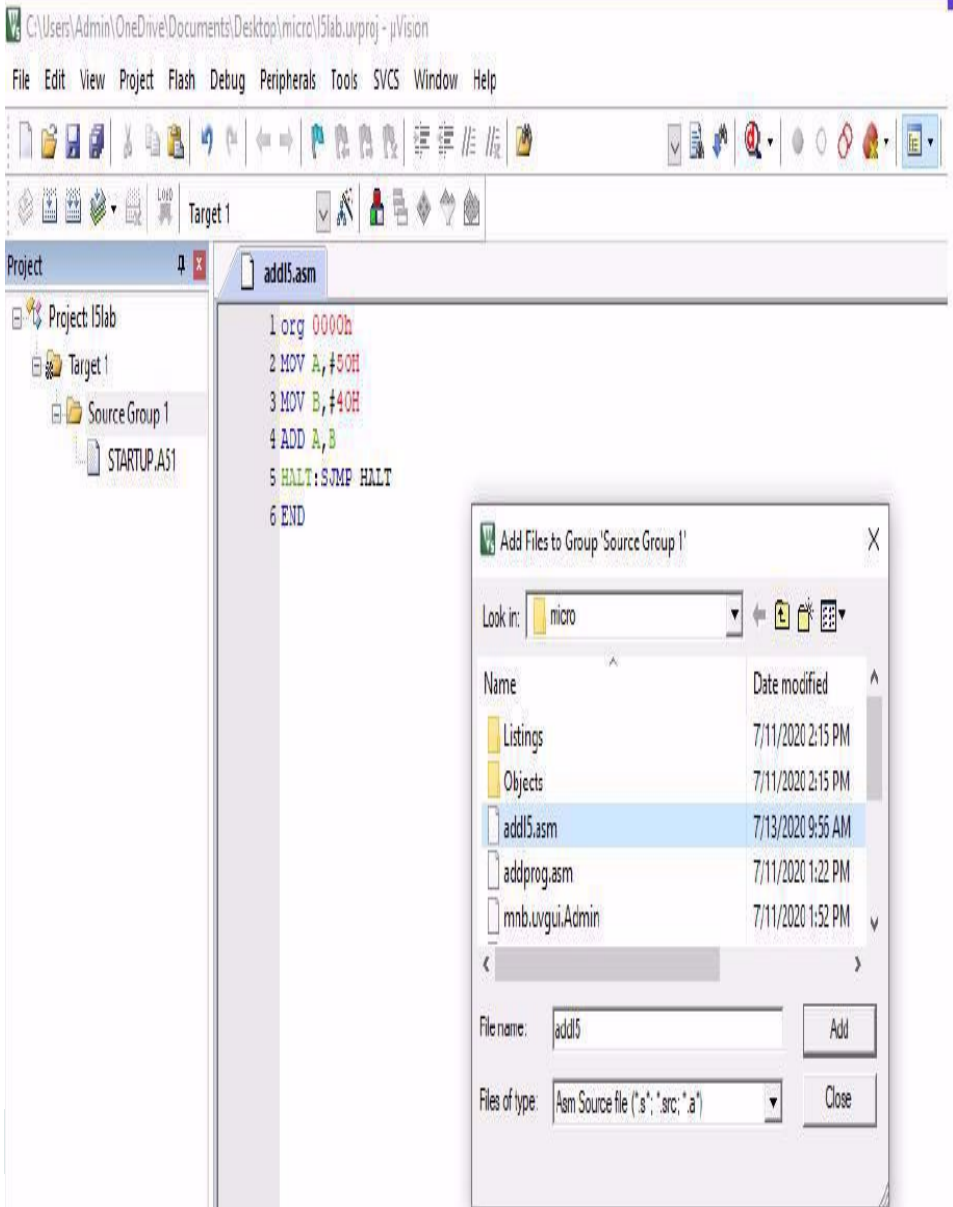
STEP 9: Save the file name with extension as : .asm

Instructions to use KEIL SIMULATOR

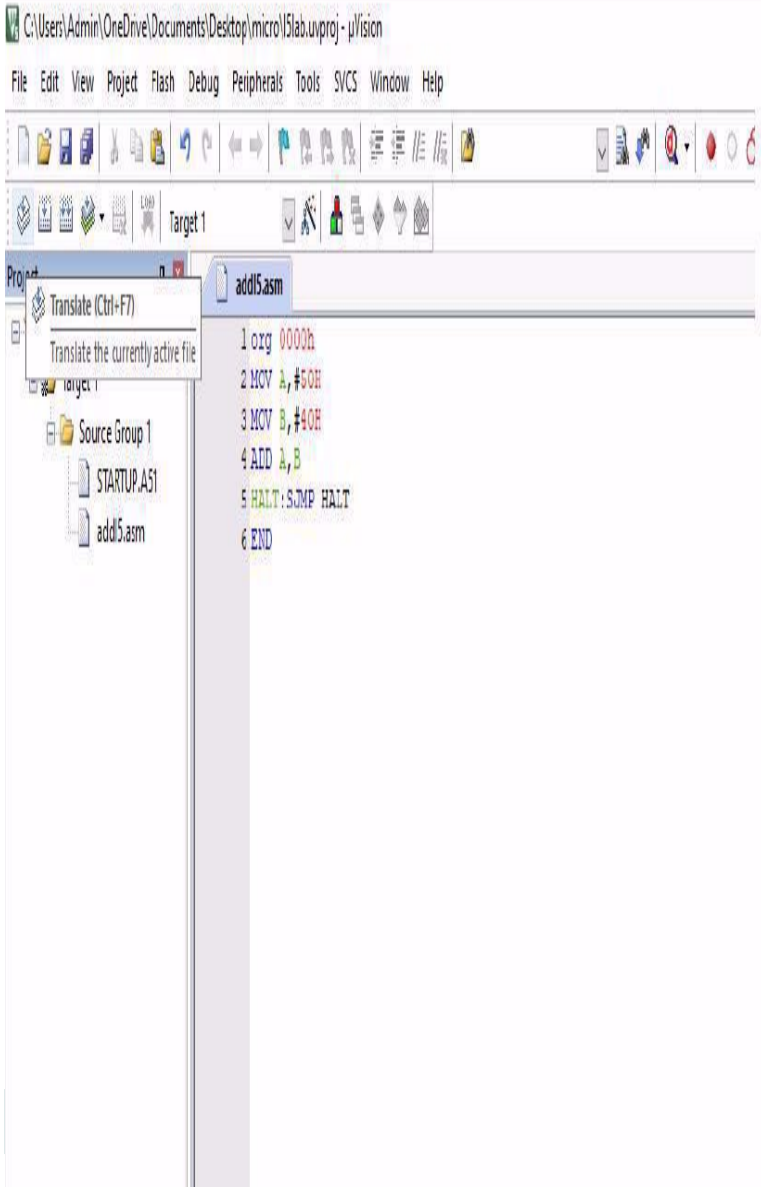


Step 10: Right click the source group and select: Add existing file to the source group

Instructions to use KEIL SIMULATOR



Instructions to use KEIL SIMULATOR

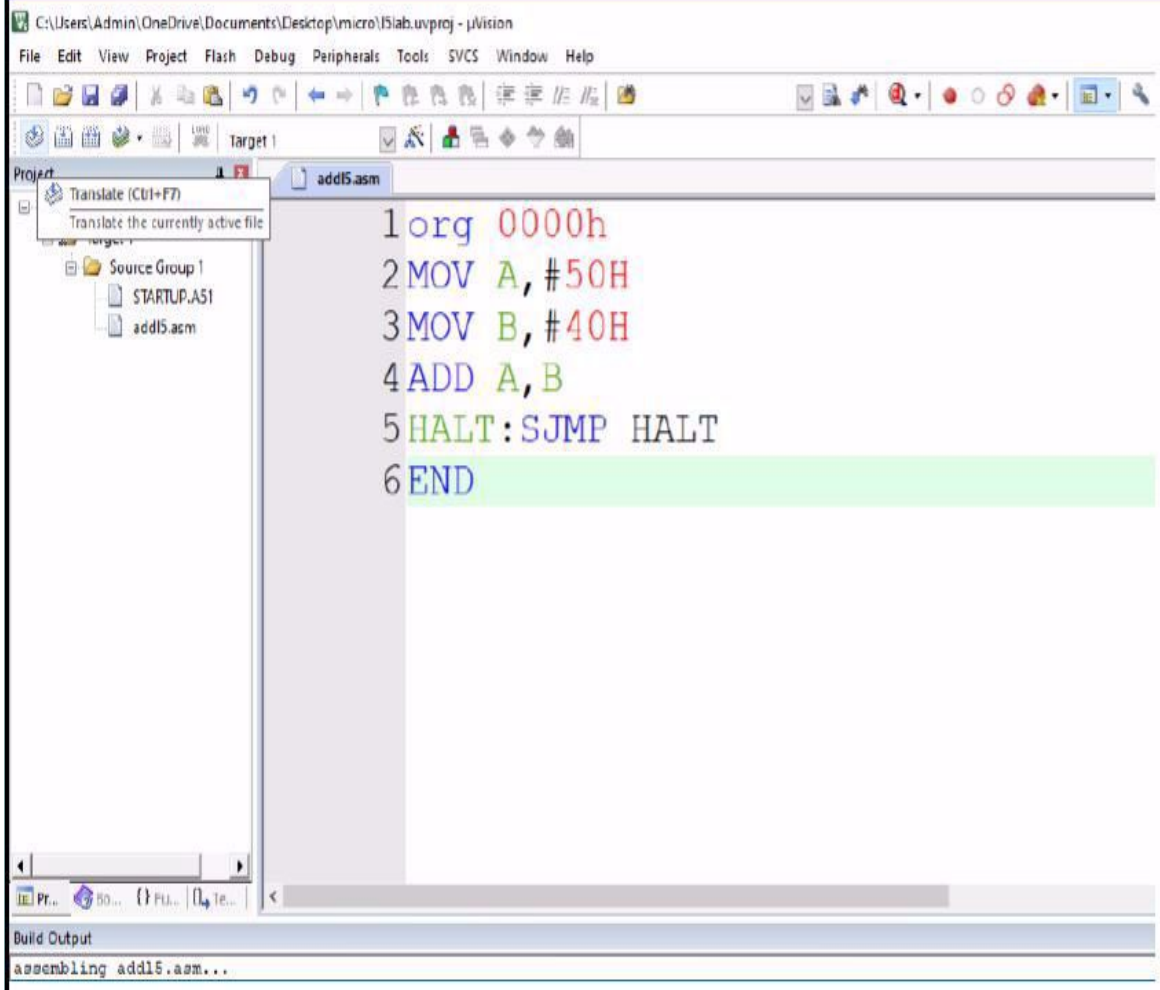


Step12: Expand the source group

in project work space and ensure

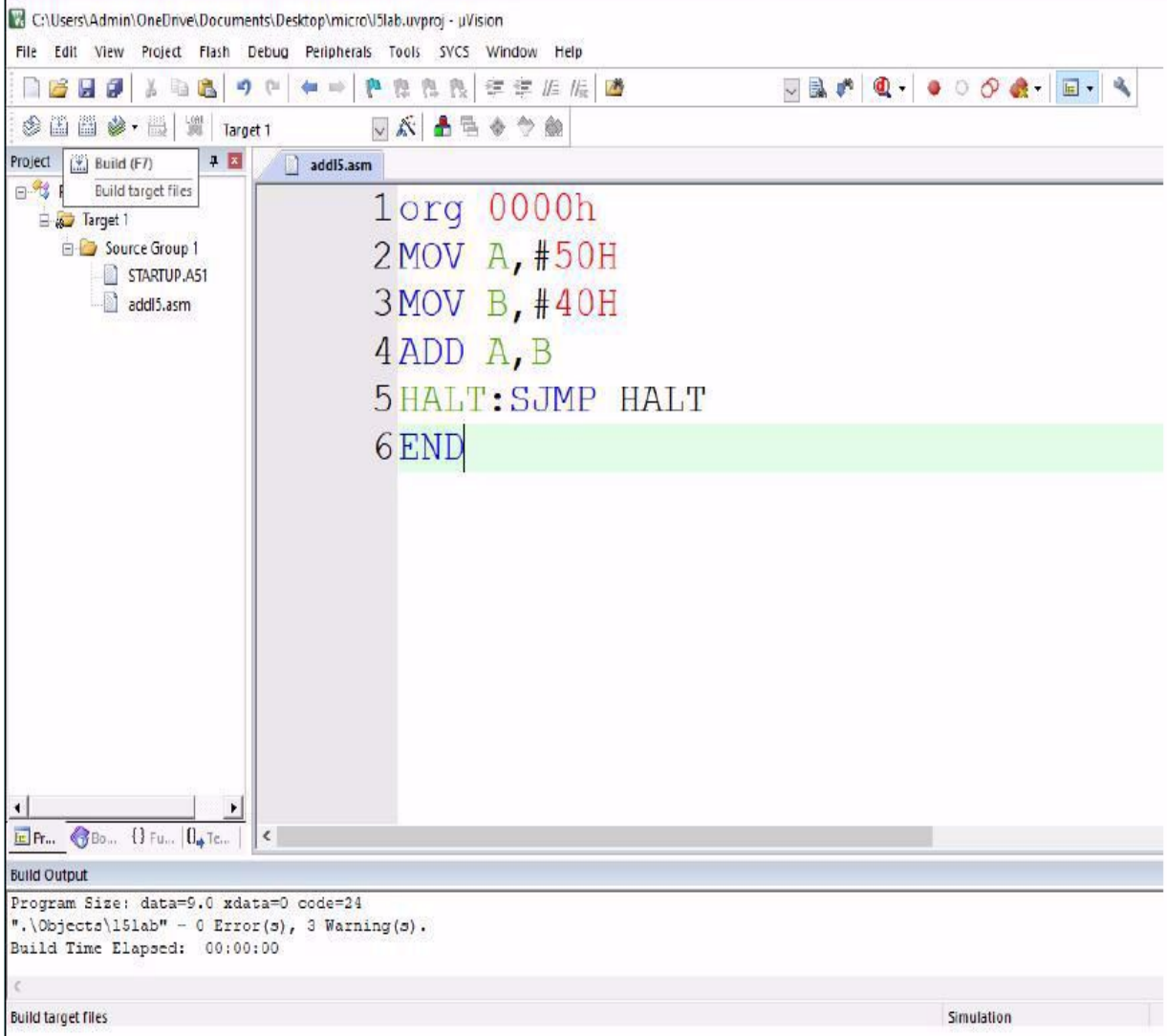
your file is under source group

Instructions to use KEIL SIMULATOR



Step13: Select :"Translate" on the left corner and check for syntax Errors

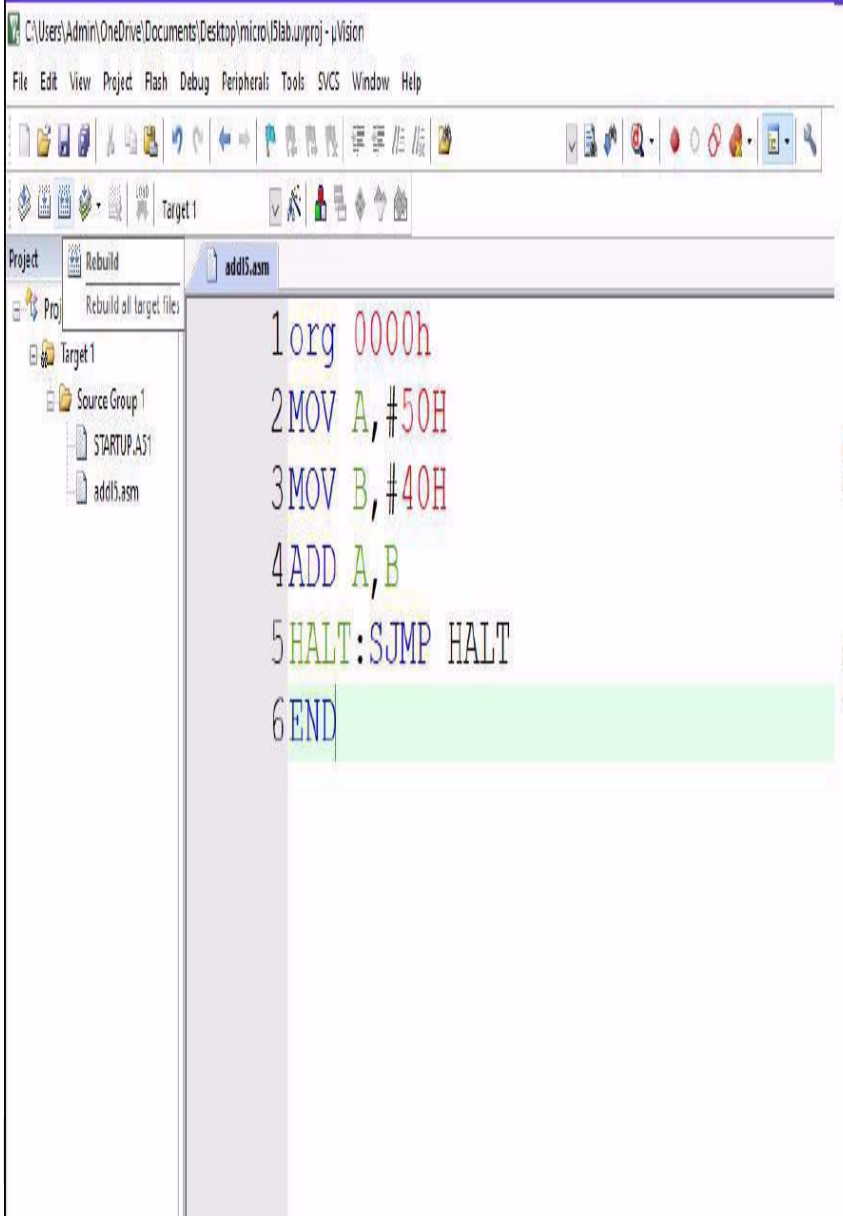
Instructions to use KEIL SIMULATOR



Step14: Select: "Build" on the

left corner

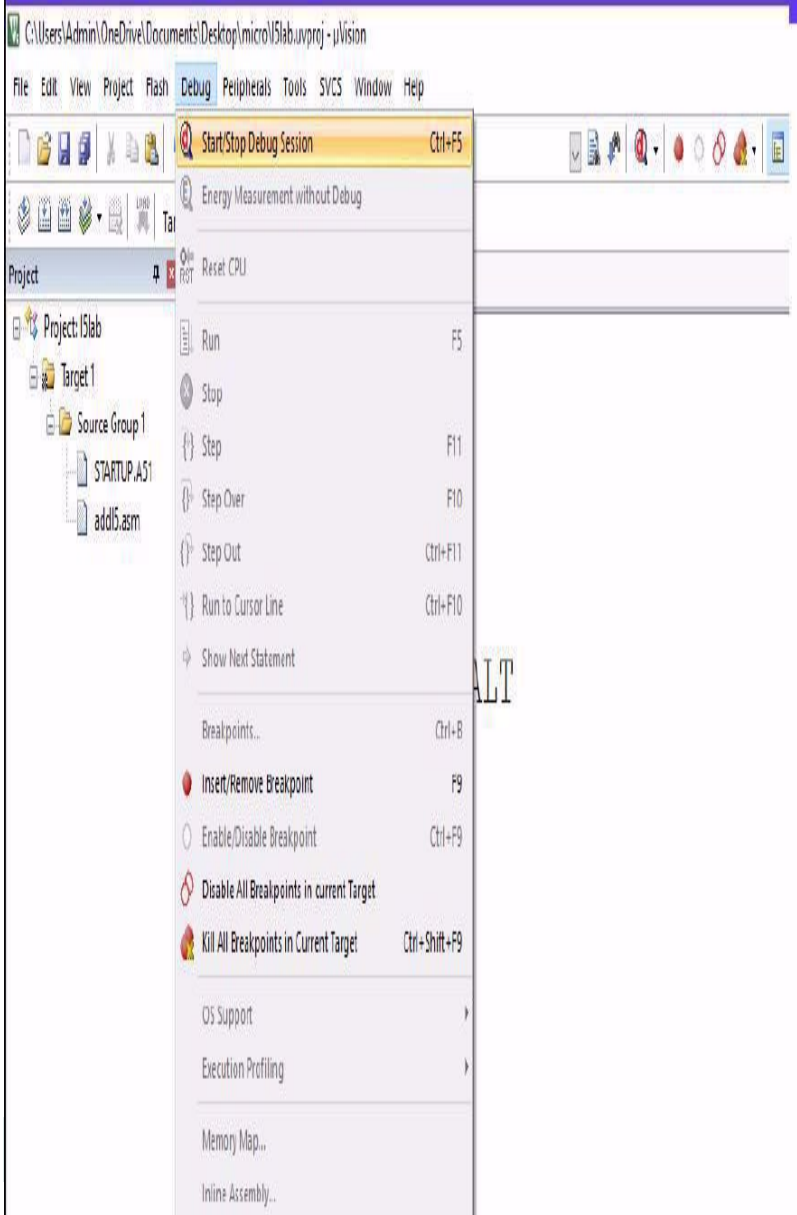
Instructions to use KEIL SIMULATOR



Step15: Select: "Rebulid" on

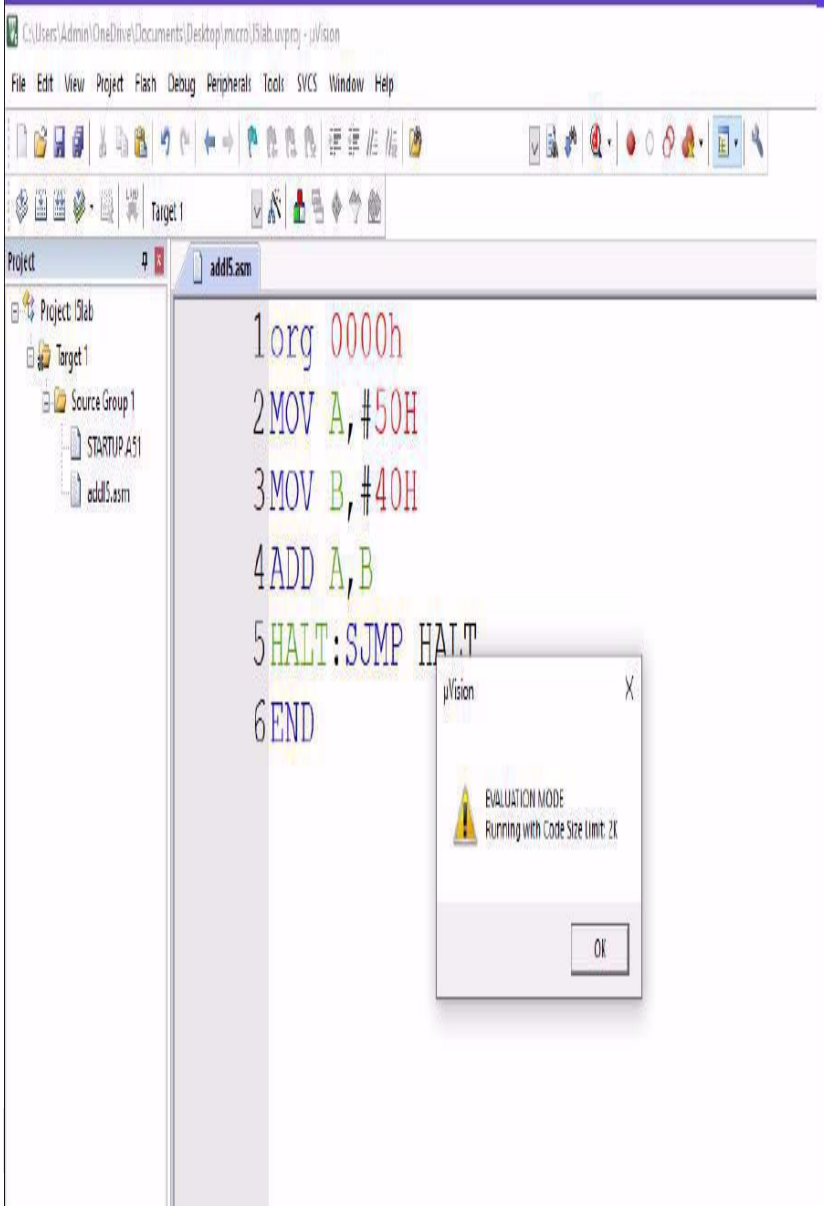
the left corner

Instructions to use KEIL SIMULATOR



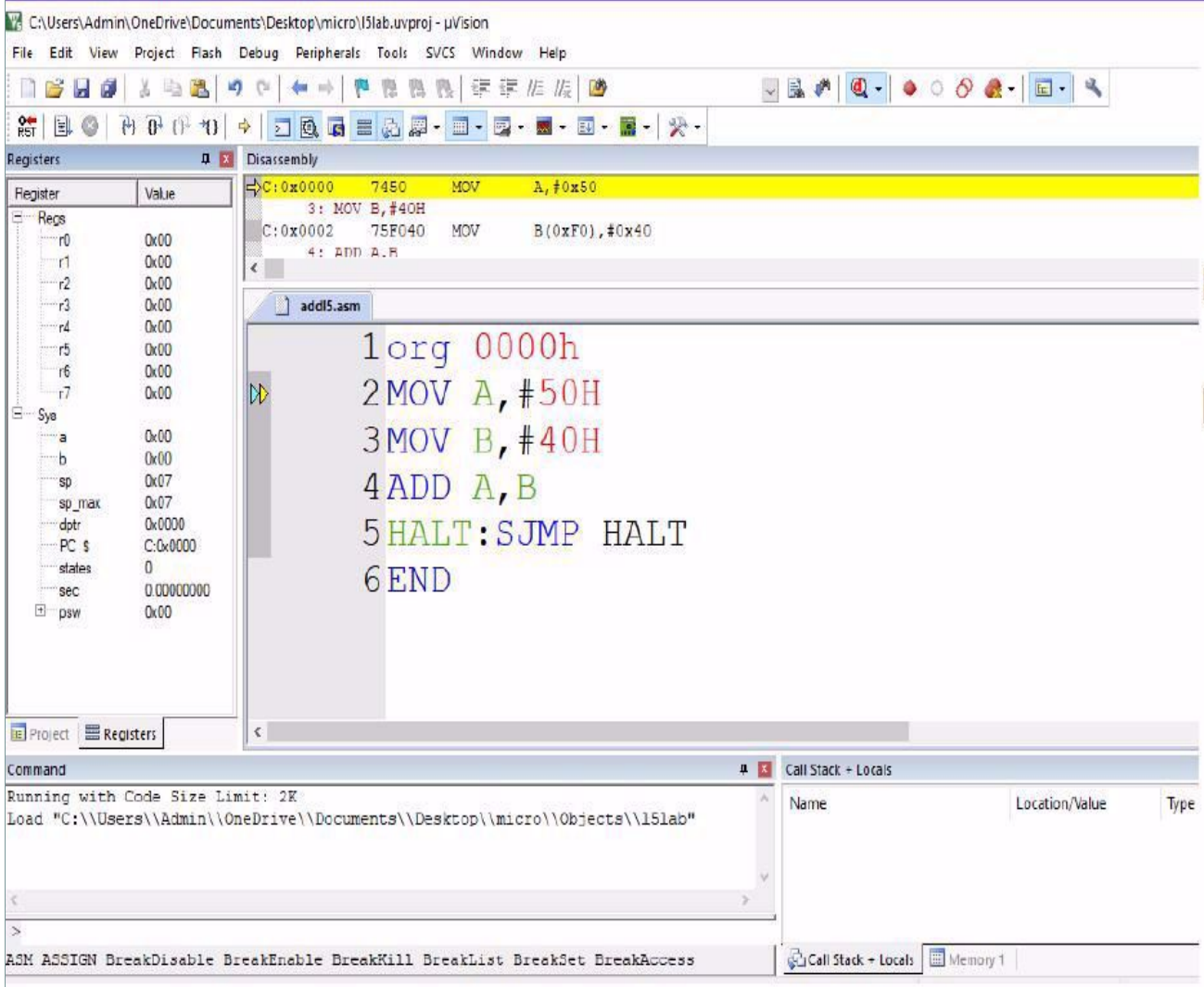
Step16: Debug: Start Debug

Instructions to use KEIL SIMULATOR



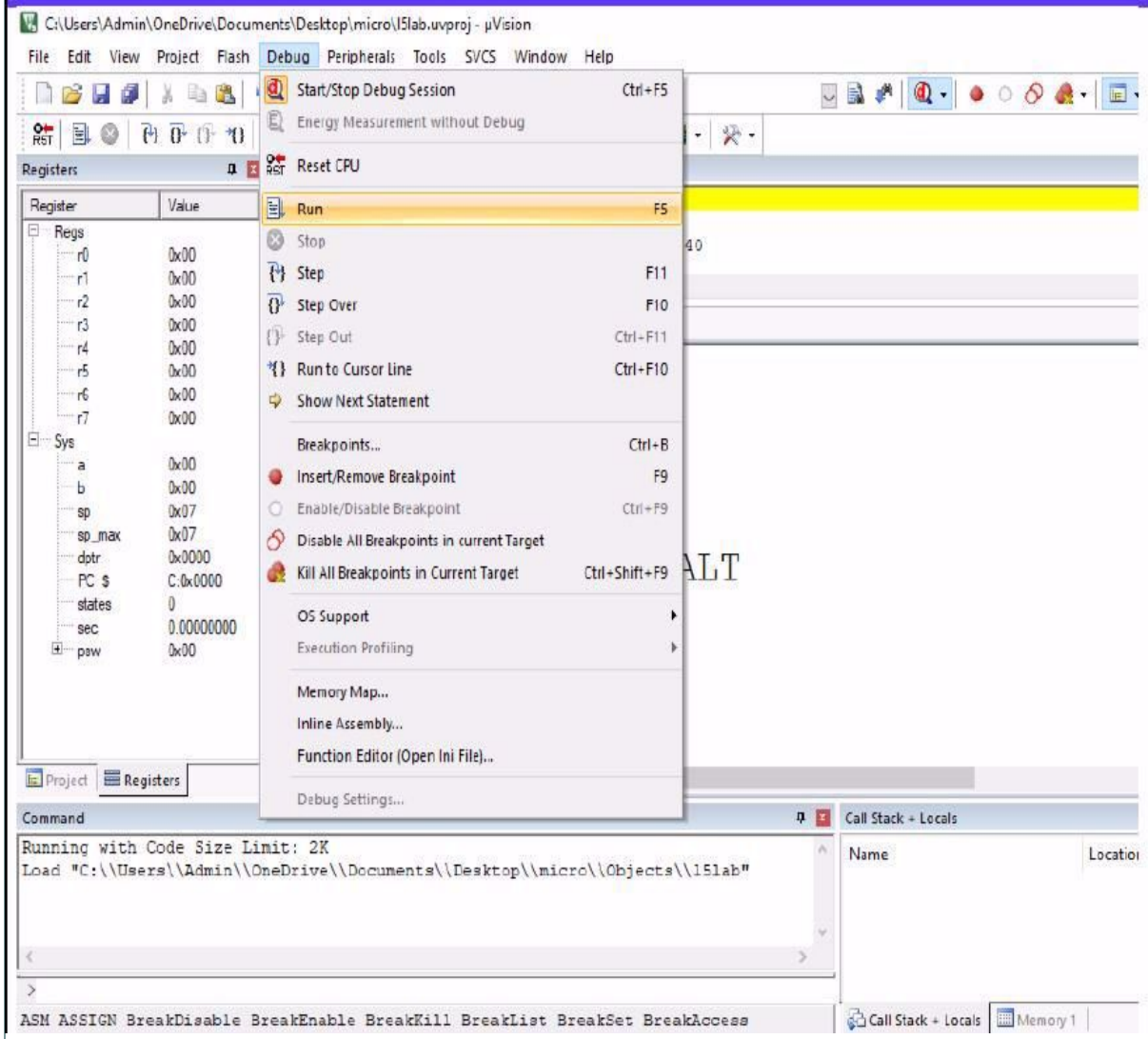
Step17: Click : "ok"

Instructions to use KEIL SIMULATOR



Registers will appear on the left

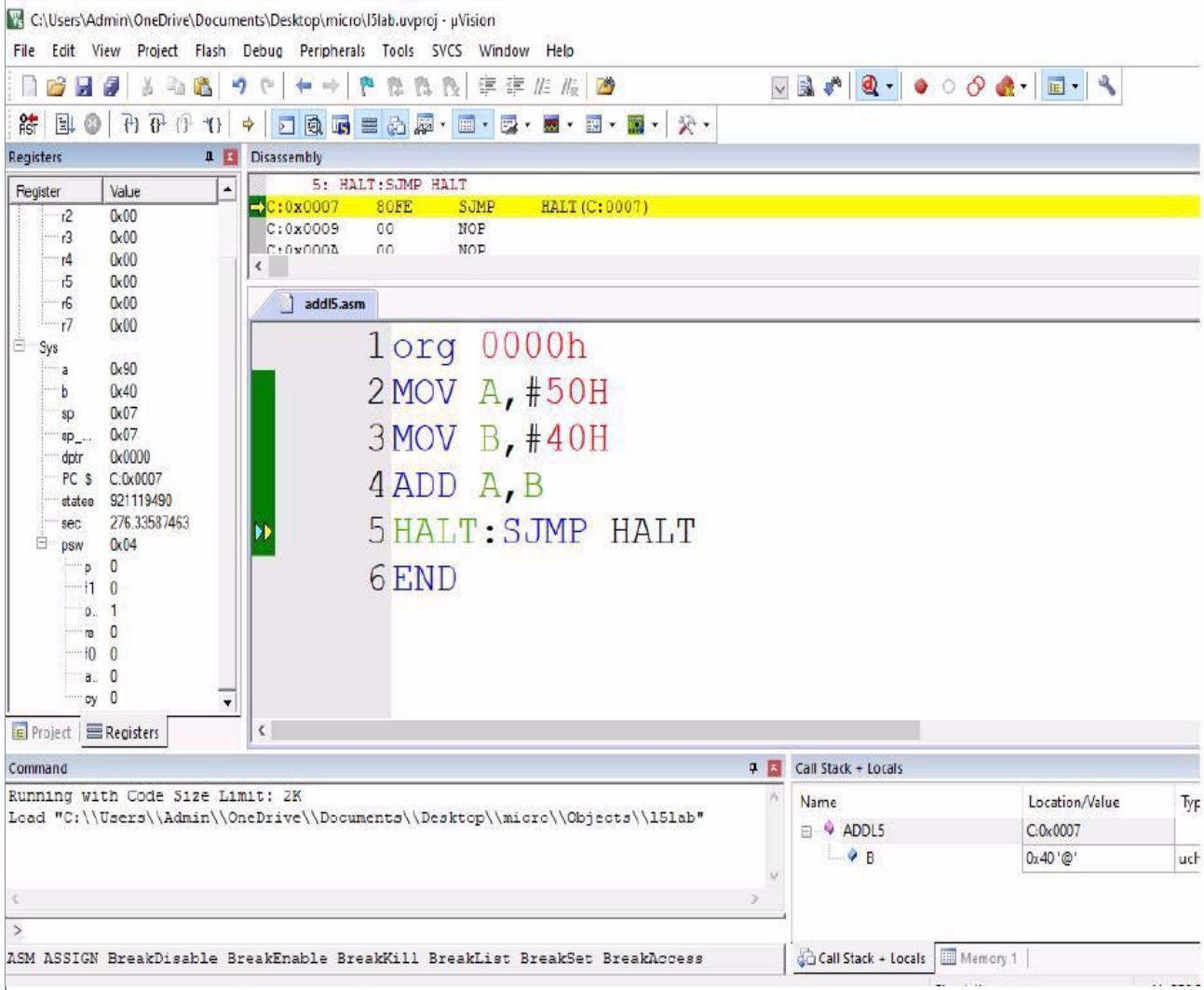
Instructions to use KEIL SIMULATOR



Step18: Click Debug: Run

Debugging in Keil is a crucial part of the embedded systems development process, enabling developers to ensure the correctness, reliability, and performance of their software on the target hardware.

Instructions to use KEIL SIMULATOR



Check the results

THANK YOU

TASK1 EXPT2

WAP to perform basic arithmetic operation using Keil micro vision and verify the status of flag registers.

PROGRAM 1-ADDITION

```
ORG 0000H
MOV A, #05H
MOV R0, #05H
ADD A, R0
H: SJMP H
END
```

PROGRAM 2-SUBTRACTION

```
ORG 0000H
MOV A, #0AH
MOV R0, #05H
SUBB A, R0
H: SJMP H
END
```

PROGRAM 3-MULTIPLICATION

```
ORG 0000H
MOV A, #05H
MOV B, #05H
MUL AB
H: SJMP H
END
```

PROGRAM 4-DIVISON

```
ORG 0000H
MOV A, #20H
MOV B, #05H
DIV AB
H: SJMP H
```

END

Write a program to calculate y where $y = x^2 + 2x + 9$.

x is between 0 and 9 and the look-up table for x^2 is located at the address (code space) of 200H.

Register R0 has the x, and at the end of the program R2 should have y.

Use the simulator to change the x value and single-step through the program,

examining the registers as you go.

```
ORG 000H
MOV DPTR, #200H
MOV A, #03H; loading x value, eg.x=3
MOV R1, A; copy x value to R1
MOV R0,A; copy x value to R0
ADD A, R1; Add with carry( adding to get 2x)
MOV R1, A; R1 will have 2x value
MOV A, R0; Load A with x value
MOVC A, @A+DPTR; Mov from code space (ROM) to A register(get X2 from lookup table into register A)
ADD A,#09H;  x2+9
ADD A, R1; x2+2x+9
MOV R2, A
HERE: SJMP HERE
ORG 200H
DB 00H,01H,04H,09H,10H,19H,24H,31H,40H,51H
```

END

ADDRESS(h)	Data(h)
0200h	00
0201h	01
0202h	04
0203h	09
0204h	10
:	:

Program - 1

Write and assemble a program to add the following data and then use the simulator to examine the CY flag.

Input Data □ 92H, 23H, 66H, 87H, F5H

```
Org 0000h

MOV A, #92H;   Load first i/p data in A register.

MOV B, #23H;   Load second i/p data in B register.

ADD A, B;Add the two data

JNC  L1 ;check for carry

INC R0;Increment reg R0, If CARRY occurs

L1: MOV B, A

MOV A, #66H;Load third i/p data in A register.

ADD A, B

JNC L2

INC R0

L2: MOV B,A

MOV A,#87H;Load fourth i/p data in A register.

ADD A,B

JNC L3
```

```
INC R0
L3: MOV B,A
MOV A,#0F5H;Load fifth i/p data in A register.
ADD A, B
JNC L4
 INC R0
L4: SJMP L4
END
```

Program 2

Experiment -3

Stack operations

Program 1

Write and assemble a program to load values into each of registers R0 - R4 and then push each of these registers onto the stack. Single-step the program, and examine the stack and the SP register after the execution of each instruction.

```
ORG 0000H
MOV R0,#25H
MOV R1,#35H
MOV R2,#45H
MOV R3,#55H
MOV R4,#65H
PUSH 0
PUSH 1
PUSH 2
PUSH 3
PUSH 4
Hlt:SJMP Hlt
END
```

Program 2

Write and assemble a program to load values into each of registers R0 - R4 and then push each of these registers onto the stack and pop them back. Single-step the program, and examine the stack and the SP register after the execution of each instruction.

```
ORG 0000H
MOV R0,#25H
MOV R1,#35H
MOV R2,#45H
MOV R3,#55H
MOV R4,#65H
PUSH 0
PUSH 1
PUSH 2
PUSH 3
PUSH 4
POP 0
POP 1
POP 2
POP 3
POP 4
H1:SJMP H1
END
```

Program 3

Write an 8051 assemble language program to:
Set SP = 0D, (b) Put a different value in each of RAM locations 0D, 0C, 0B, 0A, 09, and 08, (c) POP each stack location into registers R0 - R4. Use the simulator to single-step and examine the registers, the stack, and the stack pointer.

```
ORG 0000H
MOV SP,#0DH
MOV 08H, #10H
MOV 09H, #11H
MOV 0AH, #12H
MOV 0BH, #13H
MOV 0CH,#14H
MOV 0DH,#16H
POP 0
POP 1
POP 2
POP 3
POP 4
L1:SJMP L1
END
```

Experiment -4

Data transfer between ROM and RAM

Program – 1

Write a program to transfer a string of data from code space starting at address 200H to RAM locations starting at 40H. The data is as shown below:

0200H: DB "VIT UNIVERSITY"
Using the simulator, single-step through the program and examine the data transfer and registers.

```
        ORG 0000H
        MOV A, #00H ;  clear accumulator
        MOV DPTR,#0200H;  Load 200h in DPTR which will act as source pointer
        MOV R7,#0EH;  The length of the string is loaded in R1 register
        MOV R0,#40H;  Load the destination pointer in R0 register
LOOP:   CLR A
        MOVC A, @A+DPTR;  Move from ROM  to Accumulator
        MOV @R0,A;  Move from Accumulator to RAM
        INC DPTR
        INC R0
        DJNZ R7, LOOP ; Decrement the count and check for zero
HERE:   SJMP HERE
        ORG 0200H
        DB "VIT UNIVERSITY"
        END
```

Program 2

Add the following subroutine to the program 1, single-step through the subroutine and examine the RAM locations. After data has been transferred from ROM space into RAM, the subroutine should copy the data from RAM locations starting at 40H to RAM locations starting at 60H.

```
ORG 000H
MOV DPTR, #200H
MOV R0, #40H
MOV R1, #0EH
LOOP:CLR A
MOVC A,@A+DPTR
MOV @R0,A
INC R0
INC DPTR
DJNZ R1,LOOP
MOV R0,#40H; Initializing R0 as source pointer
MOV R1,#60H; Initializing R1 as destination pointer
MOV R3,#0EH; Length of the string
```

```
LOOP2: CLR A
MOV A,@R0
MOV @R1,A
INC R0
INC R1
DJNZ R3, LOOP2
HERE:SJMP HERE
ORG 200H
DB "VIT UNIVERSITY"
END
```