JSPM UNIVERSITY, PUNE

FACULTY OF SCIENCE AND TECHNOLOGY

SCHOOL OF COMPUTATIONAL SCIENCES

Introduction to Embedded Systems Lab Manual (230GETB41)

B.Tech in _____

Name: _____

PRN: _____ Roll No: _____

Branch: _____ Division: _____

2025 - 2026

JSPM University Pune B.Tech. "CSE" V/VII Sem		
Course Type: MMC	**Lab Course Title: Introduction to Embedded System Lab**	
Course Code: **230GETB41**	**Teaching Scheme:** **(Hrs./Week)**	**Examination Scheme:**
Credits: 1	**Lecture (L):** **Tutorial (T):** **Practical (P): 2** **Experiential Learning (EL):**	**Theory (TH): --** **Practical (PR): 50 Marks** **Oral (OR): 50 Marks**
Prerequisite Courses, if any: -		

List of Laboratory Experiments

[Conduct the following experiments by writing program using 8051/AT89C51 using an evaluation board/simulator and the required software tool]

1	Write a program to find the GCD and LCM of a given two bytes in a memory locations
2	Write a program to convert the given BCD number into its equivalent seven segment value.
3	Toggling of p1.5 with the delay of 5ms without interrupts using timer 1
4	Toggling of p1.2 with the delay of 5ms with interrupts using timer 0
5	Count no. 0f pulses without interrupts using counter 0
6	Count no. 0f pulses with interrupts using counter 1
7	Program to transfer data serially with baud rate of 9600
8	Program to receive data serially with baud rate of 9600
9	Sequence Generator Using Serial Interface in 8051
10	Programming for Delay Generation and Effect of CPU Clock
11	Programming with On-Chip Timers/Counters
12	Programming with On-Chip ADC
13	Programming for PWM Generation
14	Programming for I/O Interfacing (LED and Switch Interfacing)

Virtual LAB Links:

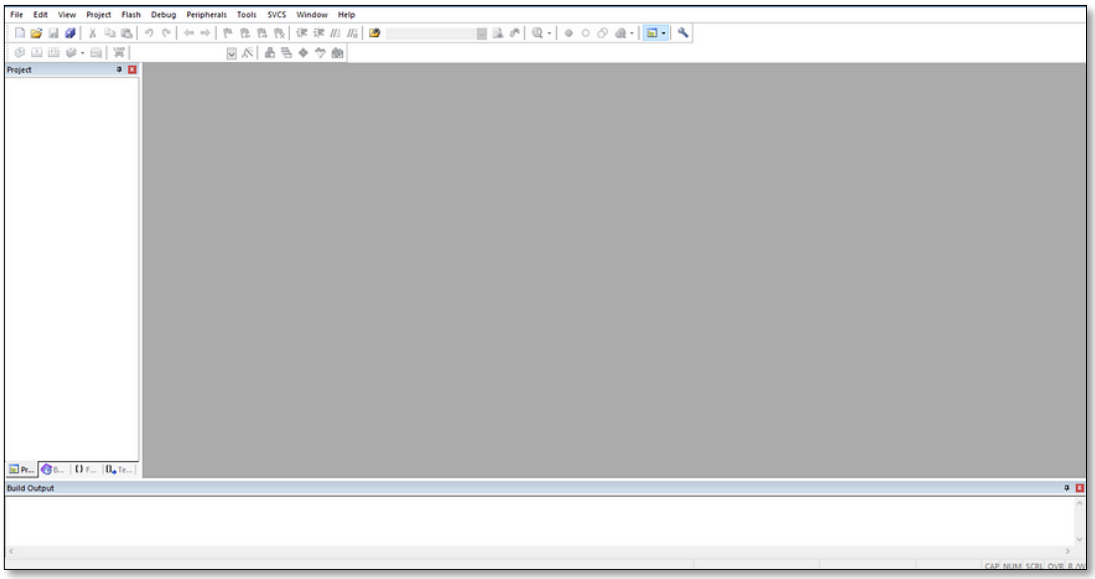
1. **Lab Name: Embedded Systems Link of the Virtual Lab:https://esd-coep.vlabs.ac.in/Introduction.html**
2. **Lab Name: Real Time Embedded Systems Laboratory**
   Link of the Virtual Lab: http://vlabs.iitkgp.ac.in/rtes/#

INTRODUCTION TO KEIL µvision

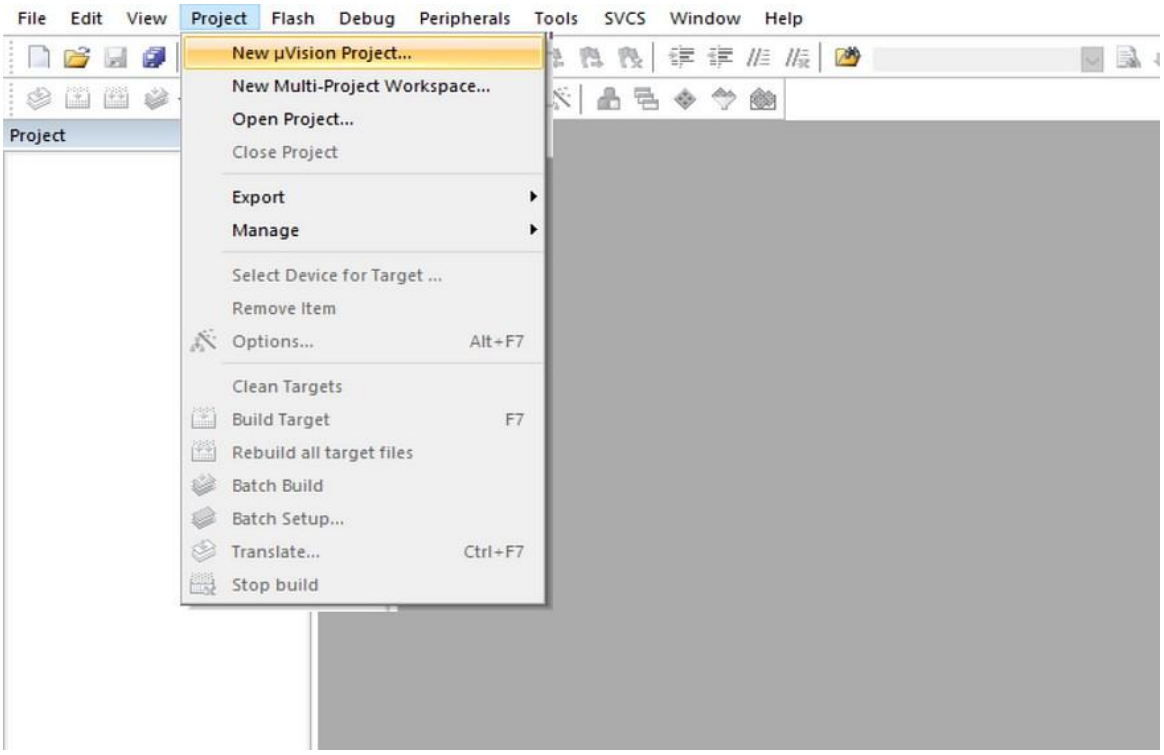
We will explore how to program the 8051 microcontroller using Keil uVision5.

Creating a Project in Keil µvision

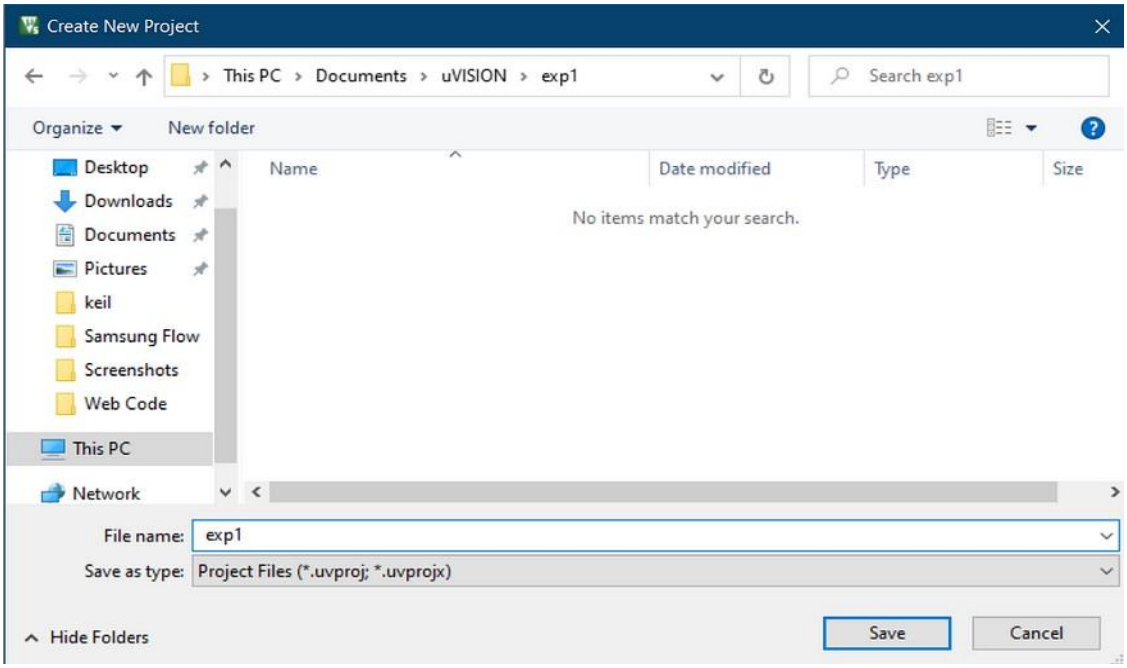
When we open Keil µvision for first time we will be able to see IDE as shown in figure below.



To create a new 8051 project using Keil IDE, click on the 'Project 'item on the IDE Menu bar and select 'New uVision Project… 'as shown in the above image.

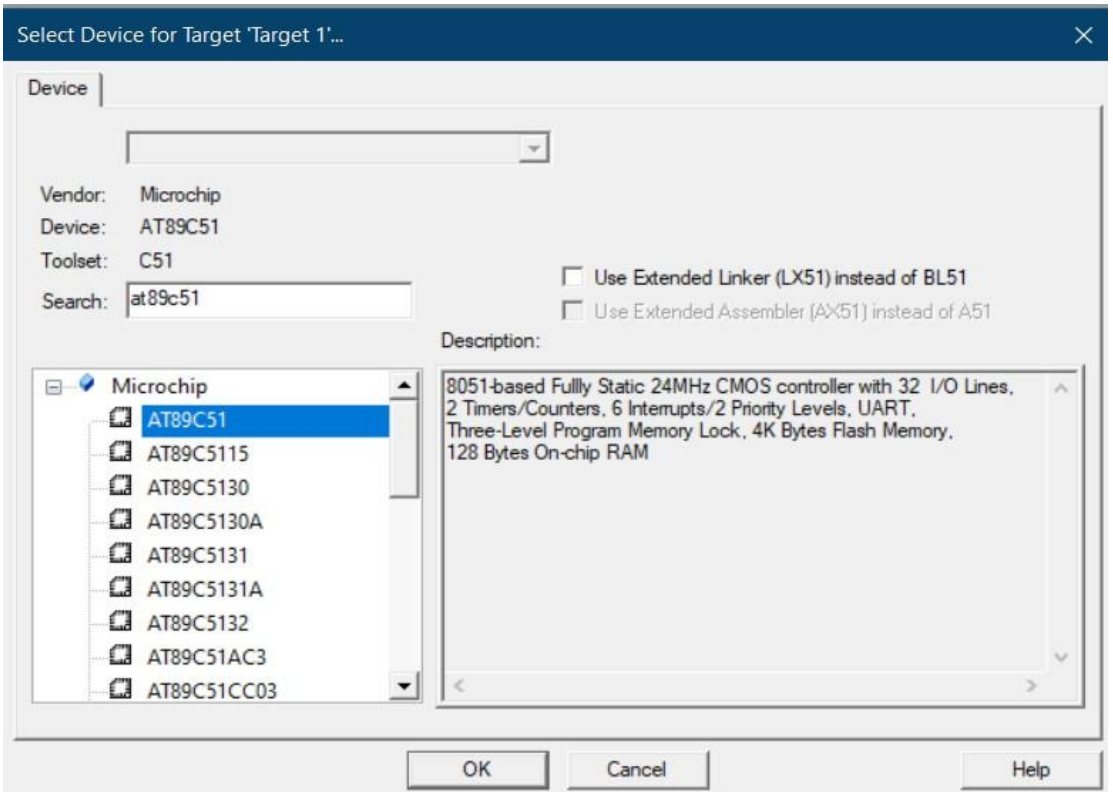


Now create a Folder to store your project and give a name to your Project files (*.uvproj), for eg exp1 (exp1.uvproj) , then save it.

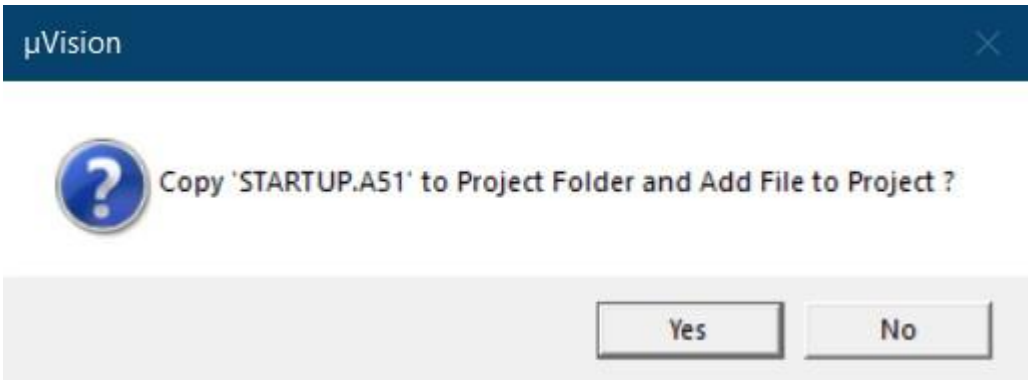


Selecting 8051 Microcontroller

- After saving the project , a dialog box will appear to select a Device
- Then type “AT89C51” in search box come under Microchip
- Click on AT89C51 and click on OK

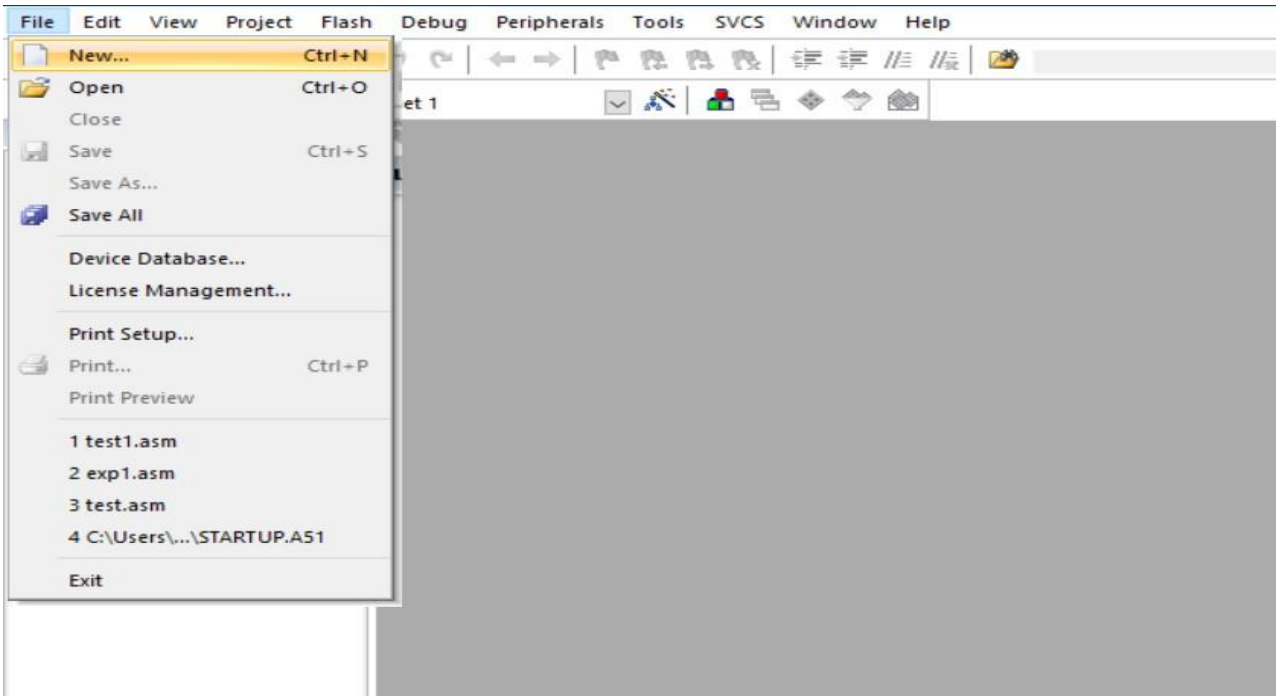


You will get another dialog as shown below. Asking to copy STARTUP.A51, Click on No

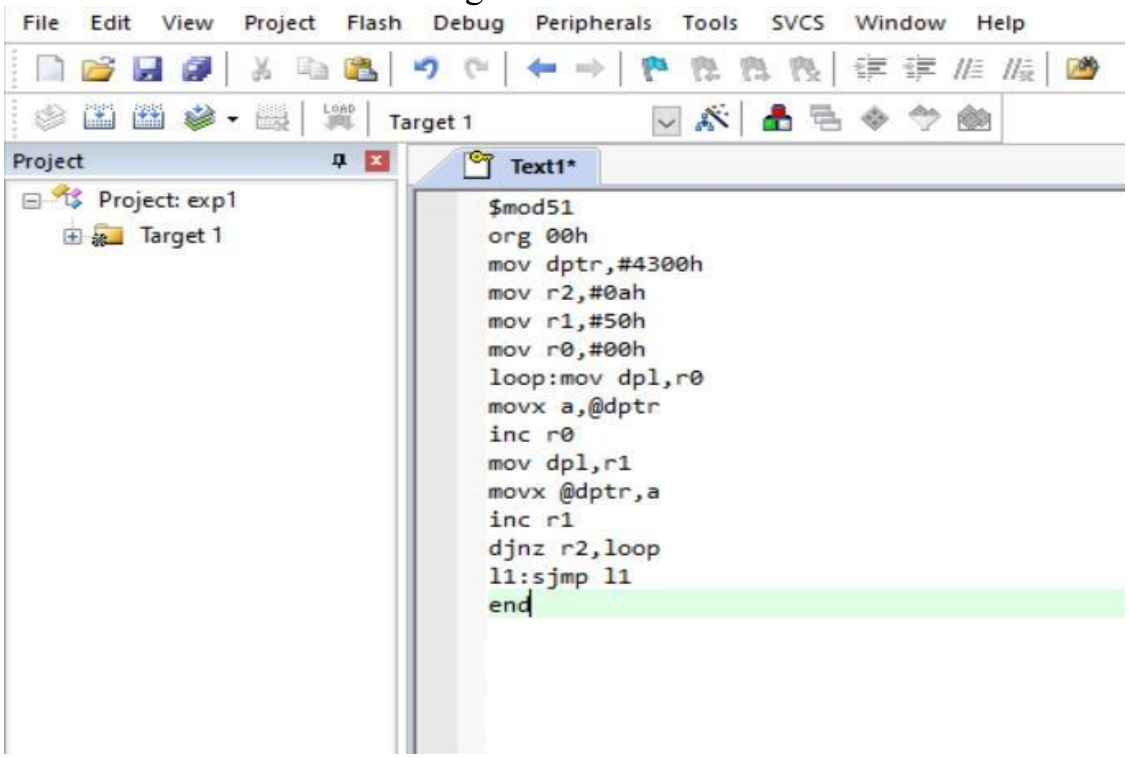


Writing Code

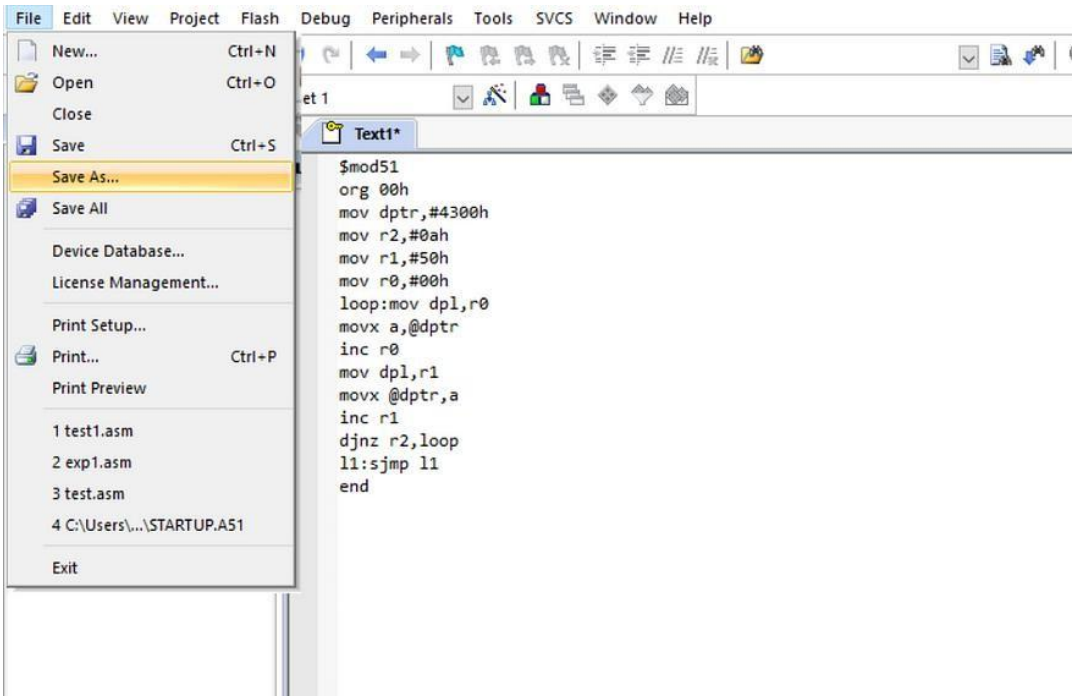
Now in order to write code we need a blank file, Click on File >> New... or can use keyboard shortcut Ctrl+ N



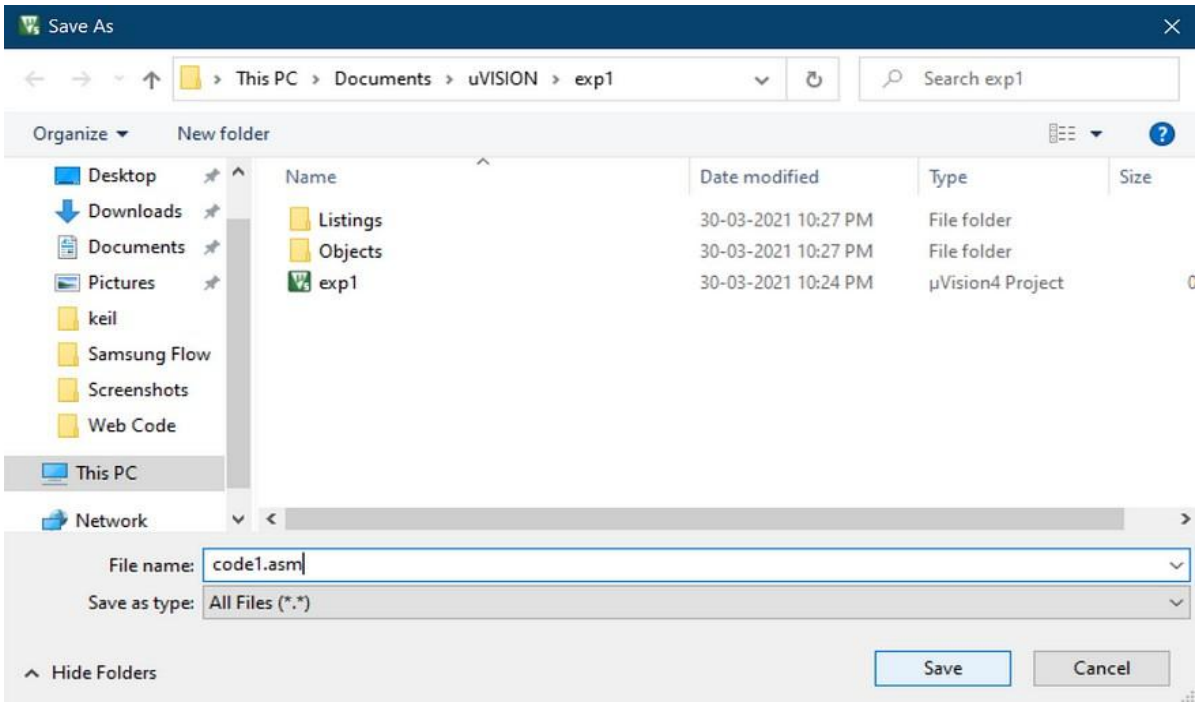
Write a code as shown in the figure below



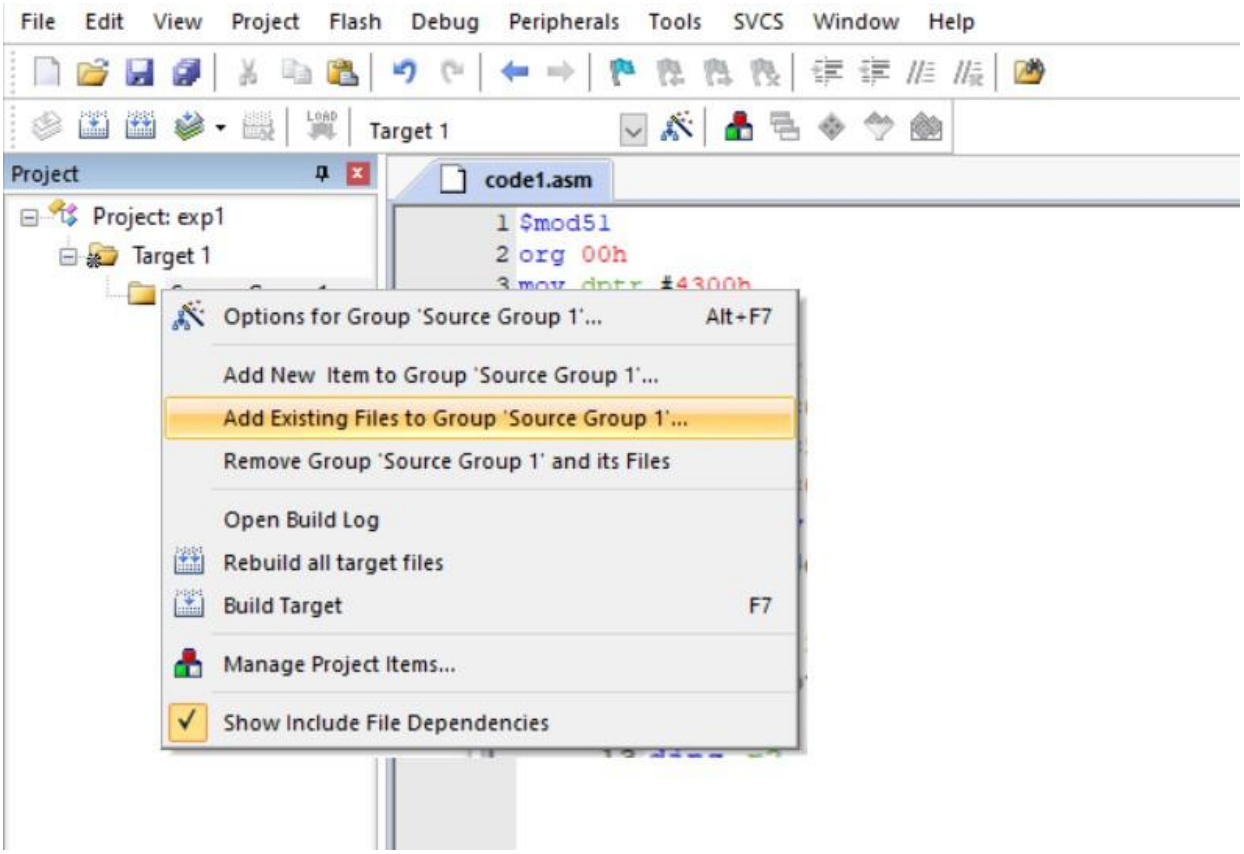
Now “Save As” the file , Click on File >> Save As...



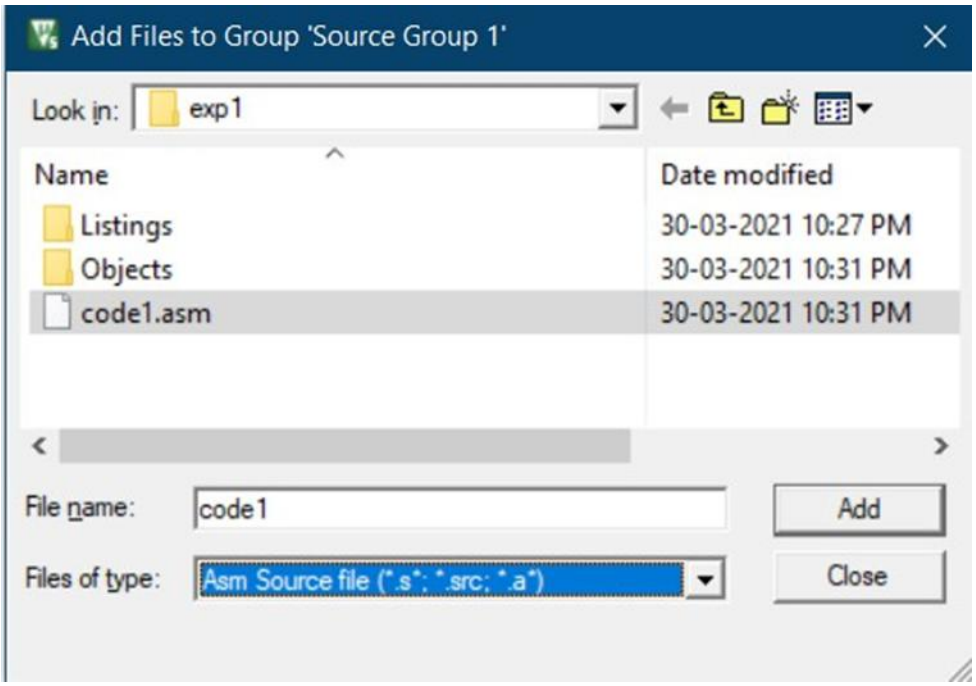
Give a file name to your code with extenstion *.asm, for eg code.asm , then save it.



In left corner you can see a Project Box , in that Project Box expand Target1 , then Right click on Source Group 1, Then click on Add Existing Files to Group 'Source Group 1'...

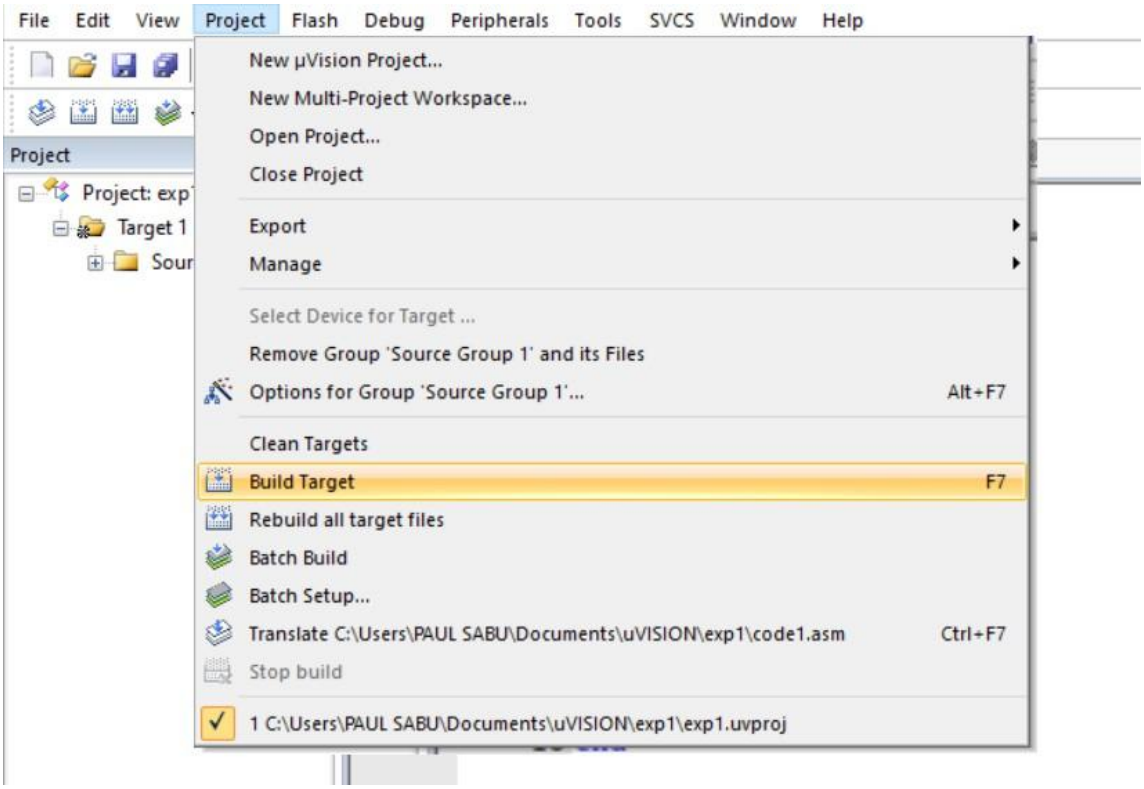


Select the Files of type : Asm Source file (*.s*;*.src*;*.a*) , Select the file with extenstion *.asm , here in my case it is "code1.asm" , Click on Add and click on Close

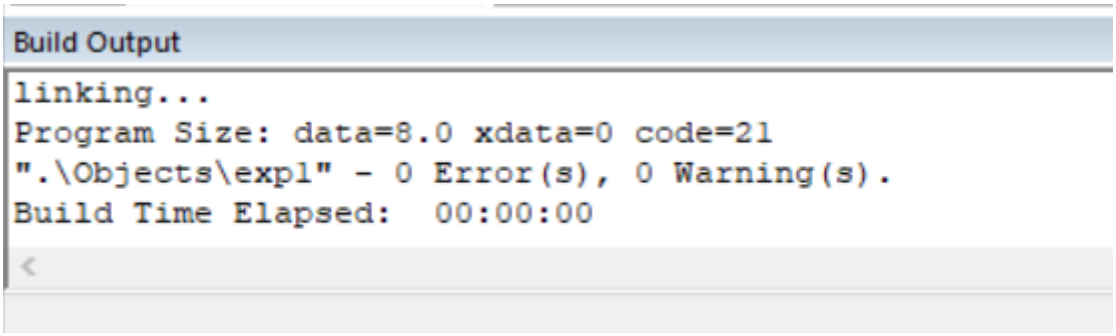


Project Build

To Bulid Project , Click on Project >> Build Target or press on F7 key

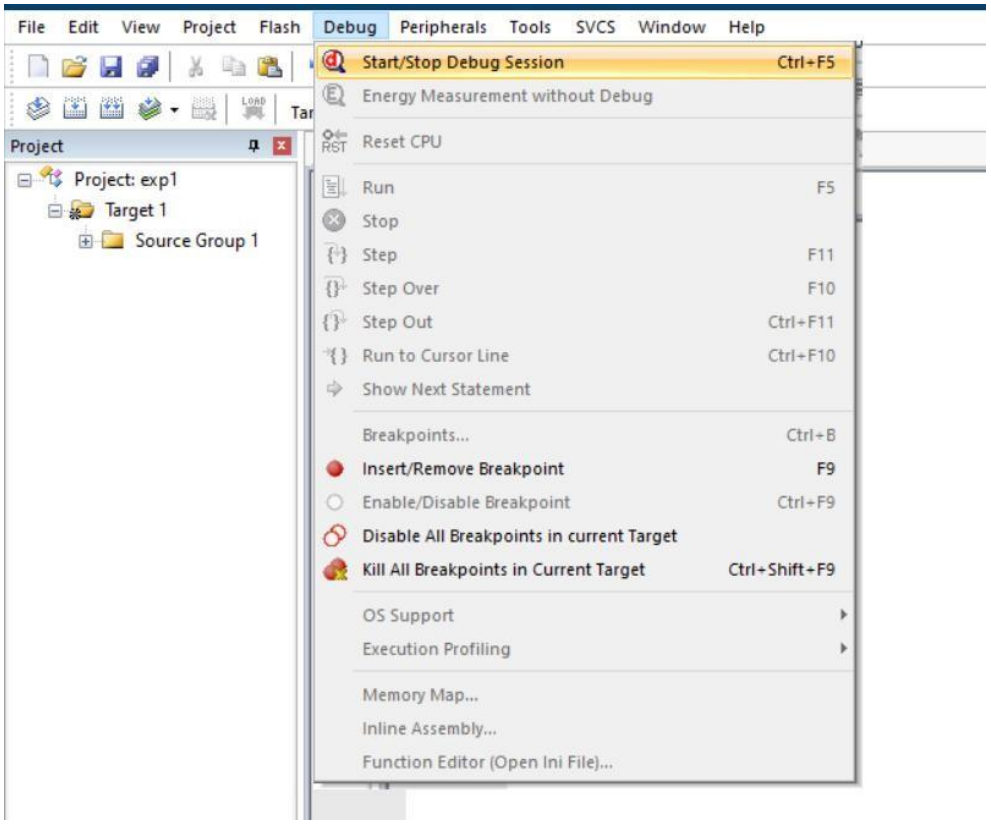


If the Build is successful there will not be any error or any warning as shown in the figure below , else verify the code and rebuild it.

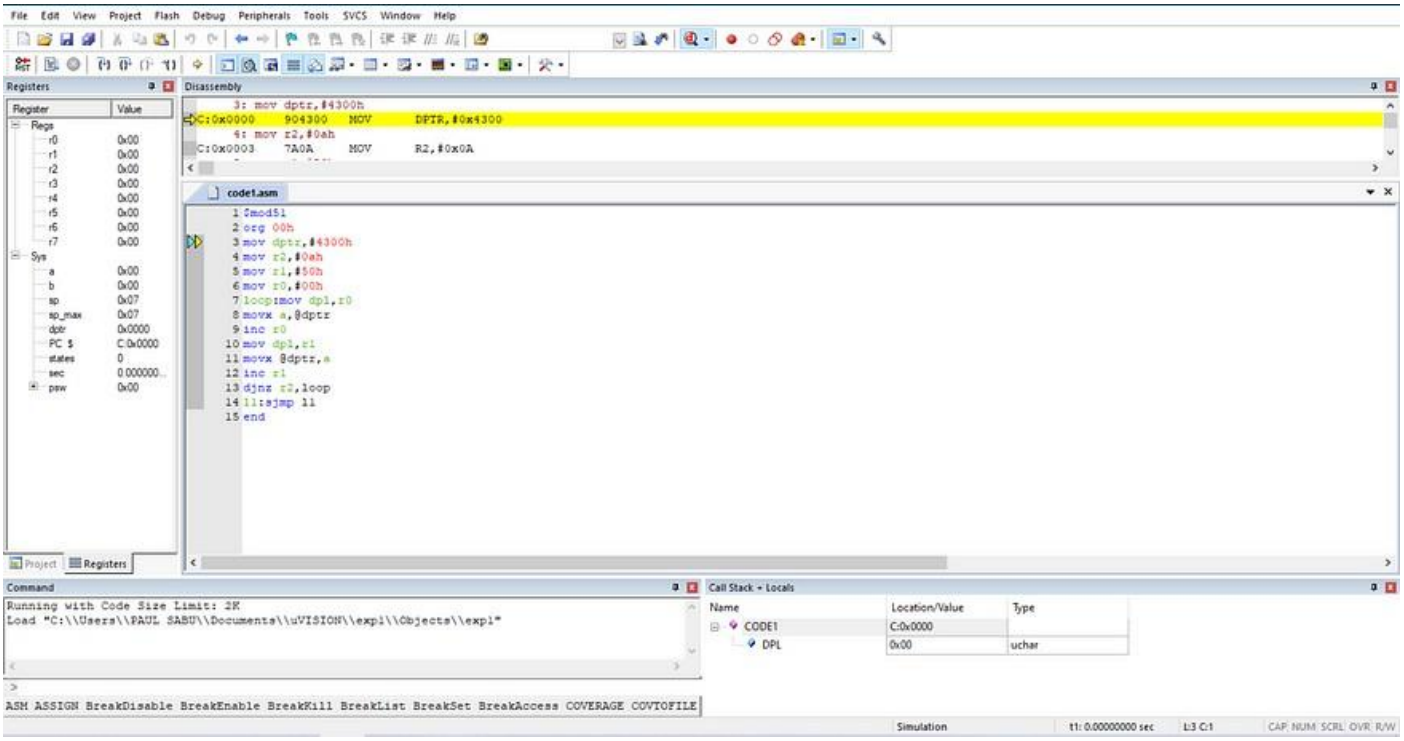


Debug and Run Code

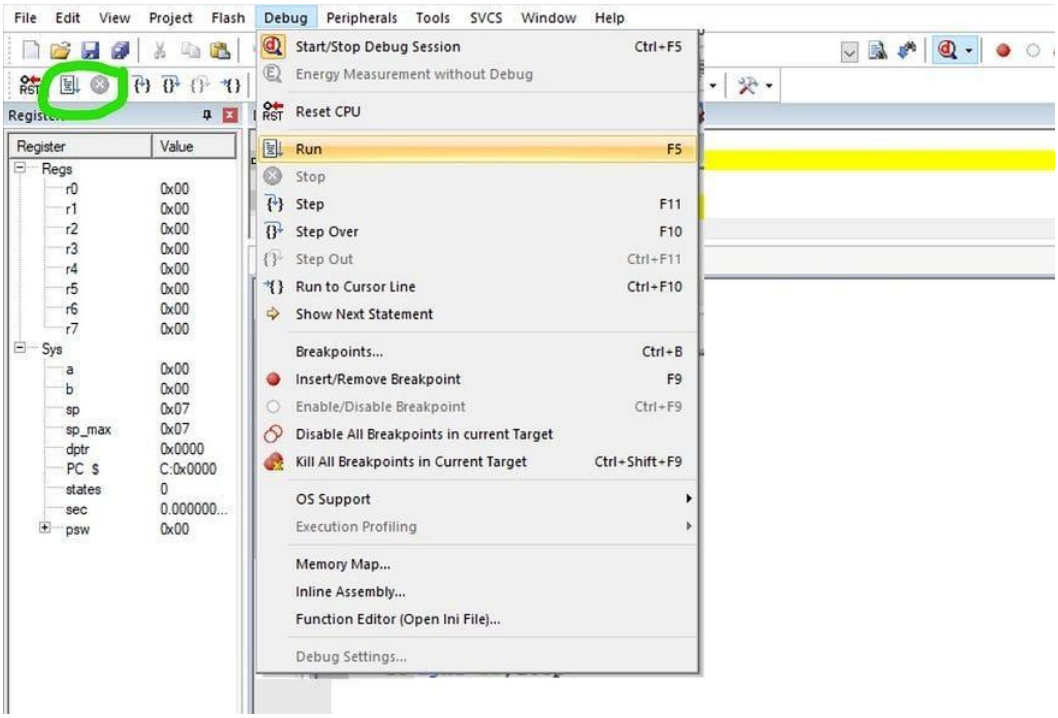
Next step is to debug the code, For that click on Debug >> Start/StopDebugSession or press on Ctrl+ F5



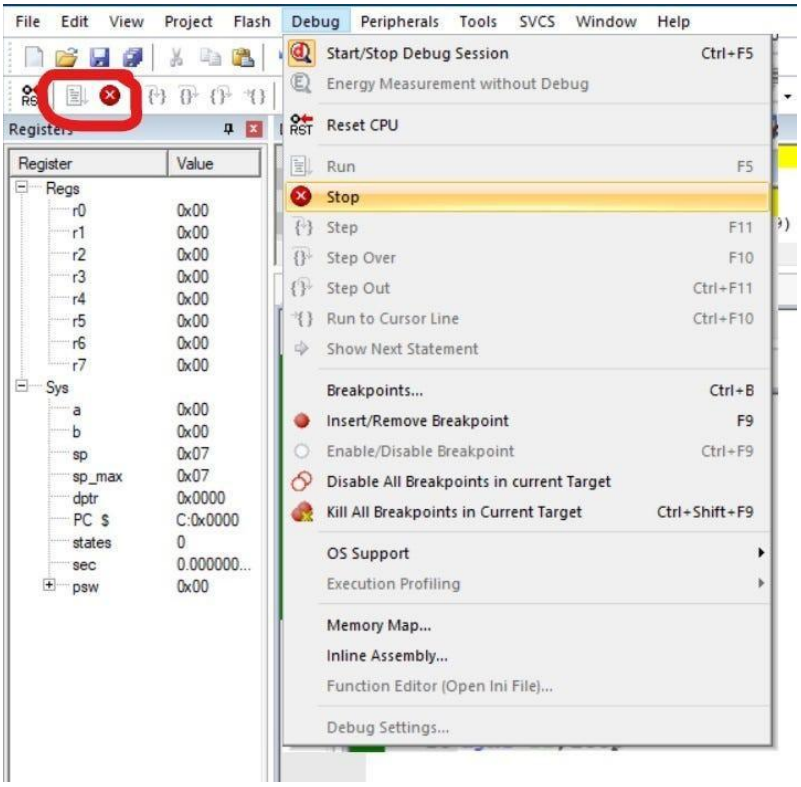
After Entering to debug , you will have a similar screen as shown in the figure below



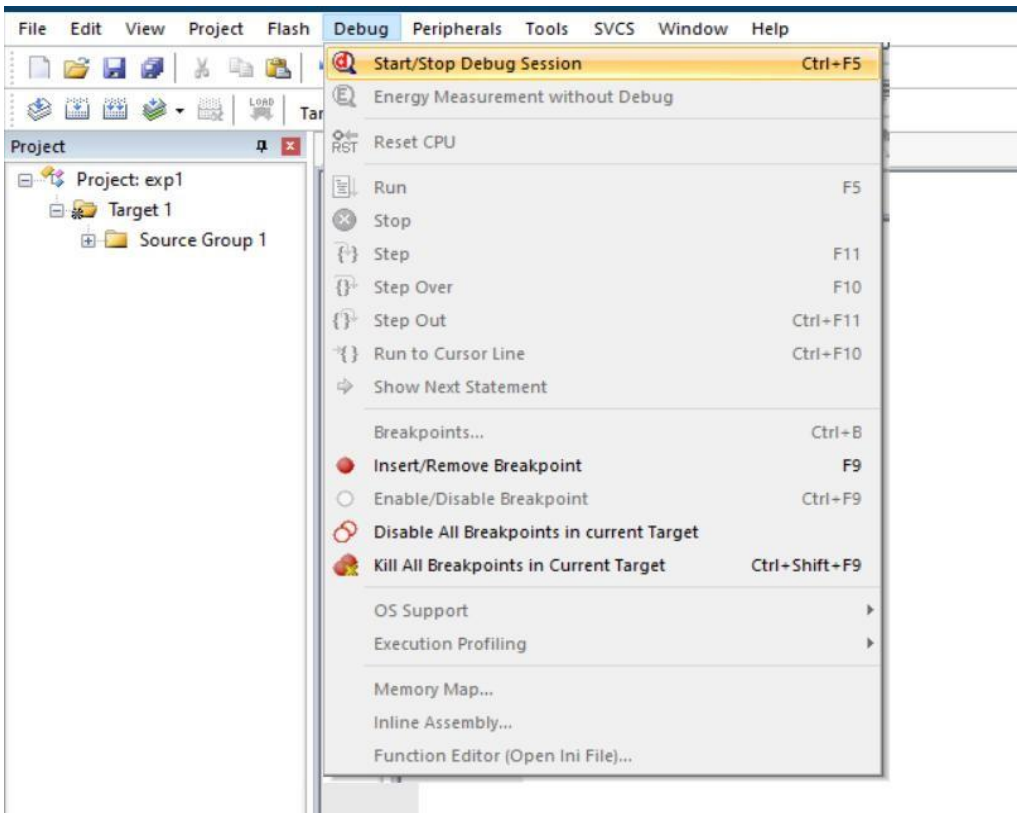
Next step is to Run the code , For that click on Debug >> Run or press on F5 or click "Run" button highlighted in green color



After a few seconds we need to Stop running the code , For that click on Debug >> Stop or click "Stop" button highlighted in red color



Before Closing the Keil uVision We need to stop Debugging, for that click on Debug >> Start/Stop Debug Session or press on Ctrl+ F5, Then close it



Experiment - 1

AIM:

a) Write an assembly language program to find the GCD and LCM of a given two bytes in a memory locations.

SOFTWARE REQUIRED: Keil µvision

***ALGORITHM*:**

Algorithm for GCD (Greatest Common Divisor)

Let the two input numbers be A and B.

Step 1: Start

Step 2: Read two positive integers A and B

Step 3: Repeat the following until B becomes 0:

a. Find the remainder R = A mod B

b. Assign A = B

c. Assign B = R

Step 4: When B = 0, the value of A is the GCD

Step 5: Print GCD

Step 6: Stop

Example (GCD of 48 and 18)

A	B	A mod B (R)
48	18	12
18	12	6
12	6	0

GCD = 6

Algorithm for LCM (Least Common Multiple)

Step 1: Start
Step 2: Read two positive integers A and B
Step 3: Find GCD of A and B using the GCD algorithm above
Step 4: Compute LCM = (A × B) / GCD
Step 5: Print LCM
Step 6: Stop

Example (LCM of 12 and 18)

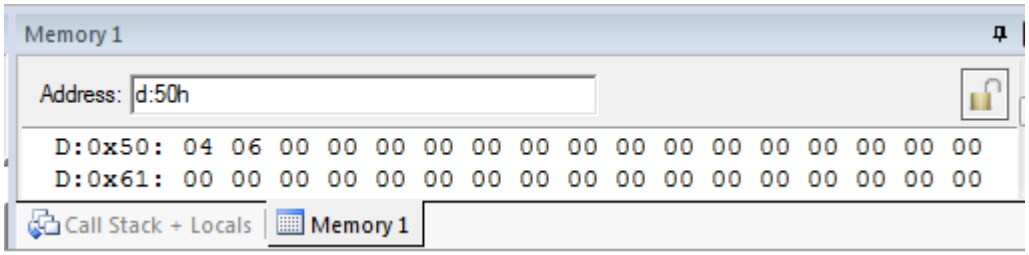
GCD(12, 18) = 6

LCM = (12 × 18) / 6 = 216 / 6 = **36**

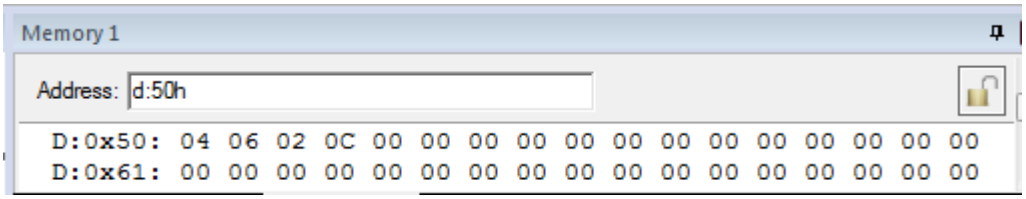
***CODE*:**

Program	Comments
a) Program to find GCD and LCM	
ORG 0000H	Starting address of Memory (Origin)
MOV A, 50H	First number is in 50H moved to register A
MOV B, 51H	Second number is in 51H moved to register B
BACK: MOV R1,B	Second number moved from B to R1 Register
DIV AB	Divide A/B (Result The quotient is stored in register
MOV A, B	A, and the remainder is stored in register B after the
CJNE A, #00H, L1	division), Remainder is compared with zero
L1: JZ LAST	If zero ,go to last and terminate; otherwise repeat
MOV A, R1	
SJMP BACK	
LAST: MOV A, R1	
MOV 52H, A	Store the GCD value in 52H
MOV A, 50H	First number is in 50H moved to register A
MOV B, 51H	Second number is in 51H moved to register B
MUL AB	Multiply A and B result stored in A
MOV B, 52H	Move GCD to B register
DIV AB	Divide A/B
MOV 53H, A	Quotient will be LCM and store in 53H
END	End of the program

***RESULT*:**
Before Execution:



After Execution:



Conclusion: We have written ALP program for GCD and LCM of two bytes and verified the output using Keil µvision Software

Experiment - 2

AIM:

Write a program to convert the given BCD number into its equivalent seven segment value.

SOFTWARE REQUIRED: Keil µvision

***ALGORITHM*:**

Algorithm for BCD equivalent seven segment value

Step 1: Start

Step 2: Initialize Port 1 (P1) as an input port and Port 2 (P2) as an output port, potentially to control a 7-segment display.

Step 3: Specifies the starting address of the lookup table in program memory by Initializing the Data Pointer.

Step 4: Main Loop (AGAIN): Read Input from Port 1 (P1) to the Accumulator (A) and Mask Lower Nibble by Performing a bitwise AND operation

Step5: Use the value in the Accumulator as an offset to access a value from the lookup table pointed to by DPTR.

Step 6: Move the 7-segment pattern from the Accumulator (A) to Port 2 (P2) to illuminate the corresponding segments of the 7-segment display.

Step7: Continuously read and display the input.

Step 8: Stop

Example (BCD equivalent seven segment value)

Lookup table: Seven-segment codes
3FH - Represents the digit 0.

06H - Represents the digit 1.

5BH - Represents the digit 2.

4FH - Represents the digit 3.

66H - Represents the digit 4.

6DH - Represents the digit 5.

7DH - Represents the digit 6.

07H - Represents the digit 7.

7FH - Represents the digit 8.

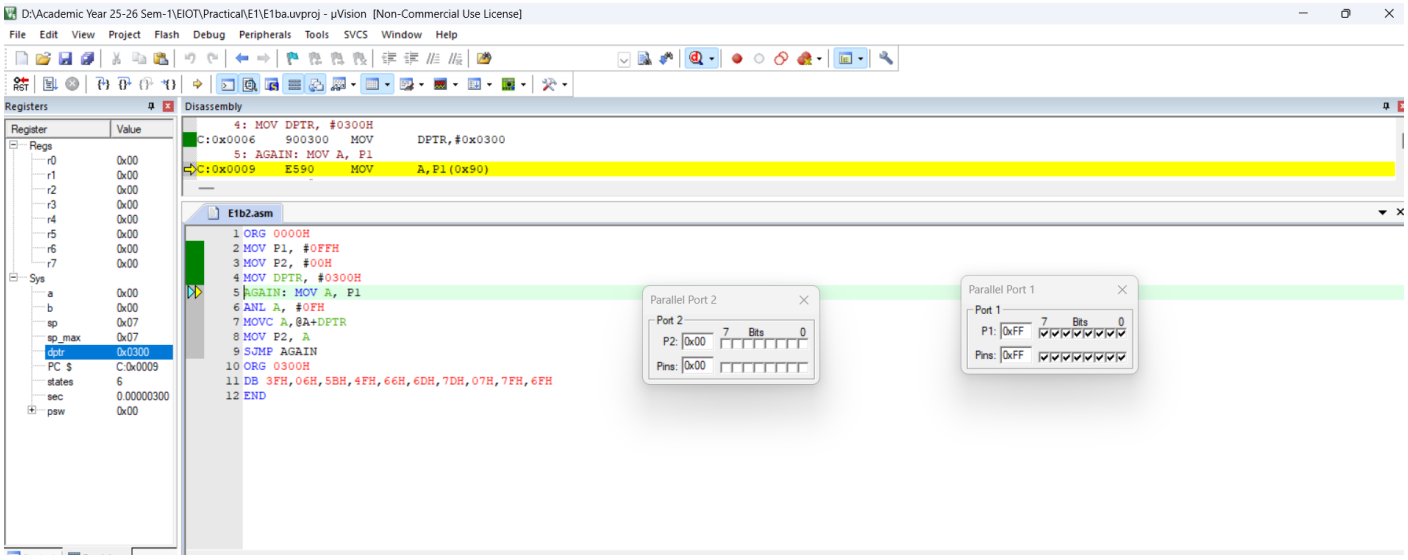
6FH - Represents the digit 9.

***CODE*:**

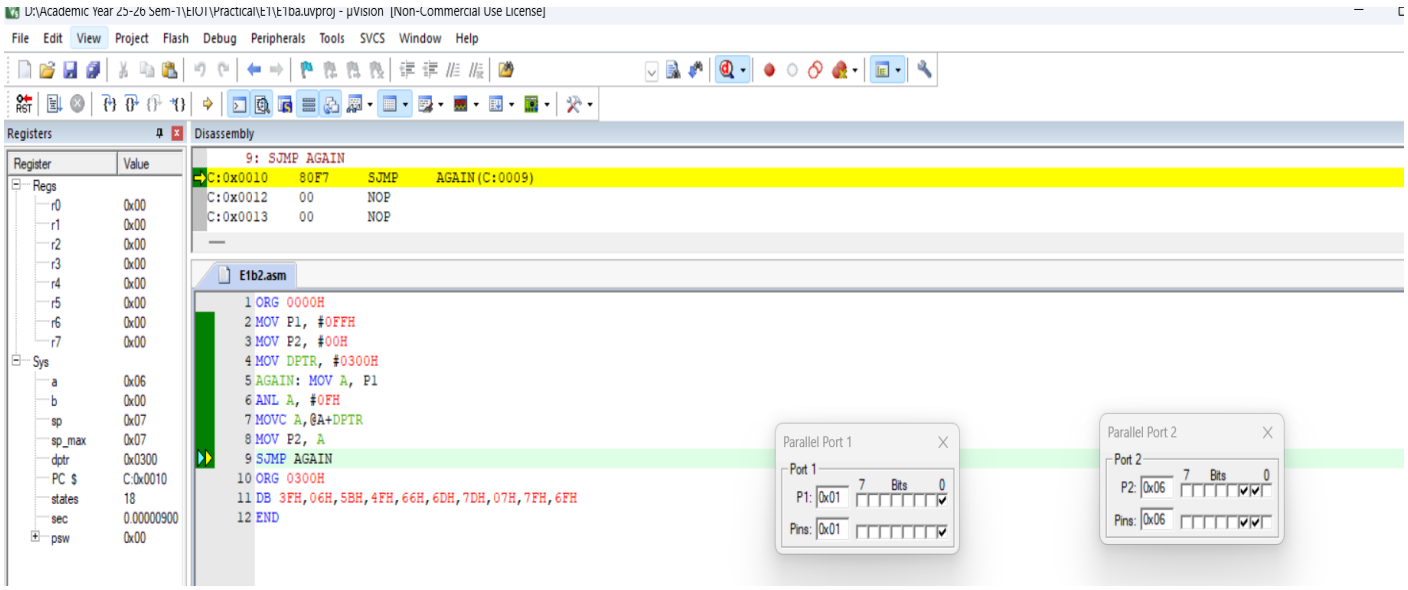
Program	**Comments**
BCD equivalent seven segment value	
ORG 0000H	Starting address of Memory (Origin)
MOV P1, #0FFH	Port 1 as an Input
MOV P2, #00H	Port 2 as an output
MOV DPTR, #0300H	Initialize the DPTR for the segment table
AGAIN: MOV A, P1	move the input value to the accumulator
ANL A, #0FH	Mask the lower nibble of the accumulator
MOVC A,@A+DPTR	Accumulator as an offset to access a value from the lookup table pointed to by DPTR.
MOV P2, A	move the accumulator's input value to Port 2
SJMP AGAIN	Loop to continuously read and display the input
ORG 0300H	Starting address of the lookup table stored in program memory
DB 3FH,06H,5BH,4FH,66H,6DH,7DH,07H, 7FH,6FH	Seven-segment codes for BCD
END	End of the program

***RESULT*:**

Before Execution:



After Execution: Input BCD Value 1 on port 1, the Seven-segment code is 6 on port 2



Conclusion:

Experiment-3

AIM:

Toggling of p1.5 with the delay of 5ms without interrupts using timer 1

SOFTWARE REQUIRED: Keil µvision

***ALGORITHM*:**

Algorithm for delay of 5ms without interrupts using timer 1

Step 1: Start

Step 2: Configure Timer 1:Load the TMOD register to set Timer 1 to operate in Mode 1 (16-bit timer mode).

Step 3: Load Timer 1 registers with the desired delay value into the Timer 1 registers, TH1 (high byte) and TL1 (low byte). Toggle P1.5 by completing the current state of pin P1.5.

Step 4: Start Timer 1 by using Set Timer 1 Run bit

Step5: Continuously check the Timer 1 Overflow Flag (TF1) until it is set (becomes 1).

Step 6: Timer Overflow Actions:Stop Timer 1 by clearing the TR1 control bit

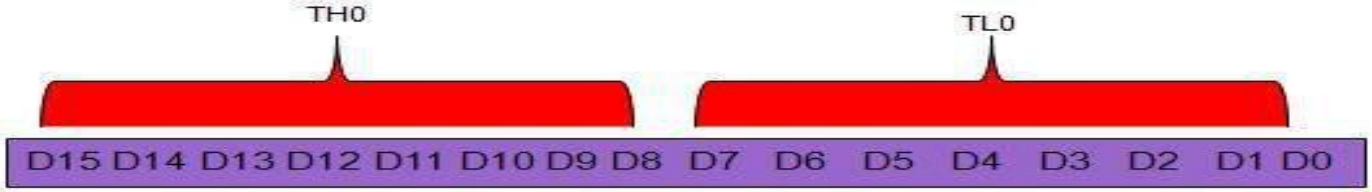
Clear Timer 1 Overflow Flag by Reset the TF1 flag to prepare for the next timer cycle

Step7: Loop Back and Return to the beginning of the AGAIN loop to repeat the entire process

Step8: Stop

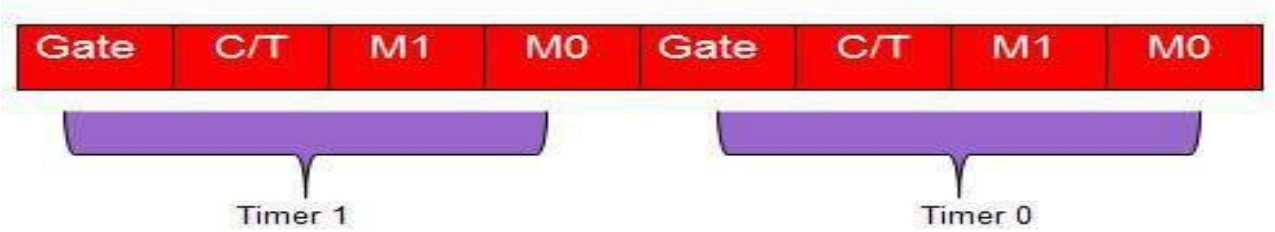
All Timer Registers(SFR):

TIMER 0 Register:



Same for the timer 1

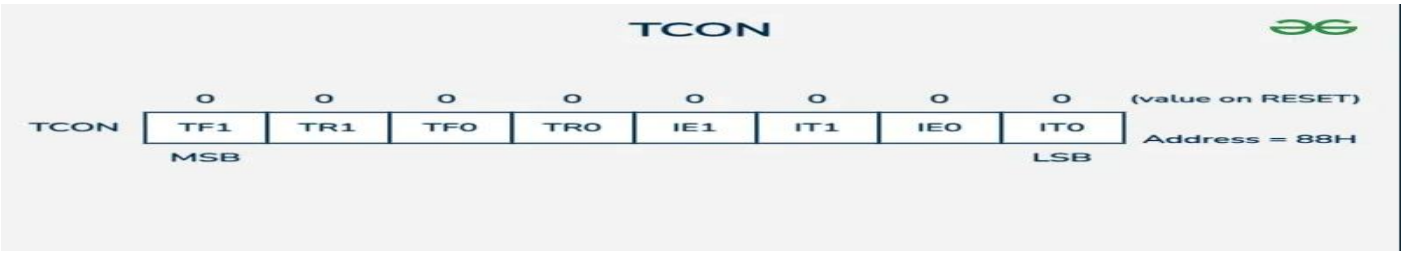
TMOD (Timer Mode) Register



Gate − When set, the timer only runs while INT(0,1) is high.
C/T − Counter/Timer select bit. For Counter 1 and for Timer 0
M1 − Mode bit 1.
M0 − Mode bit 0.

M1	M2	Mode
0	0	13-bit timer mode.
0	1	16-bit timer mode.
1	0	8-bit auto reload mode.
1	1	Spilt mode.

TCON (Timer Control Register)



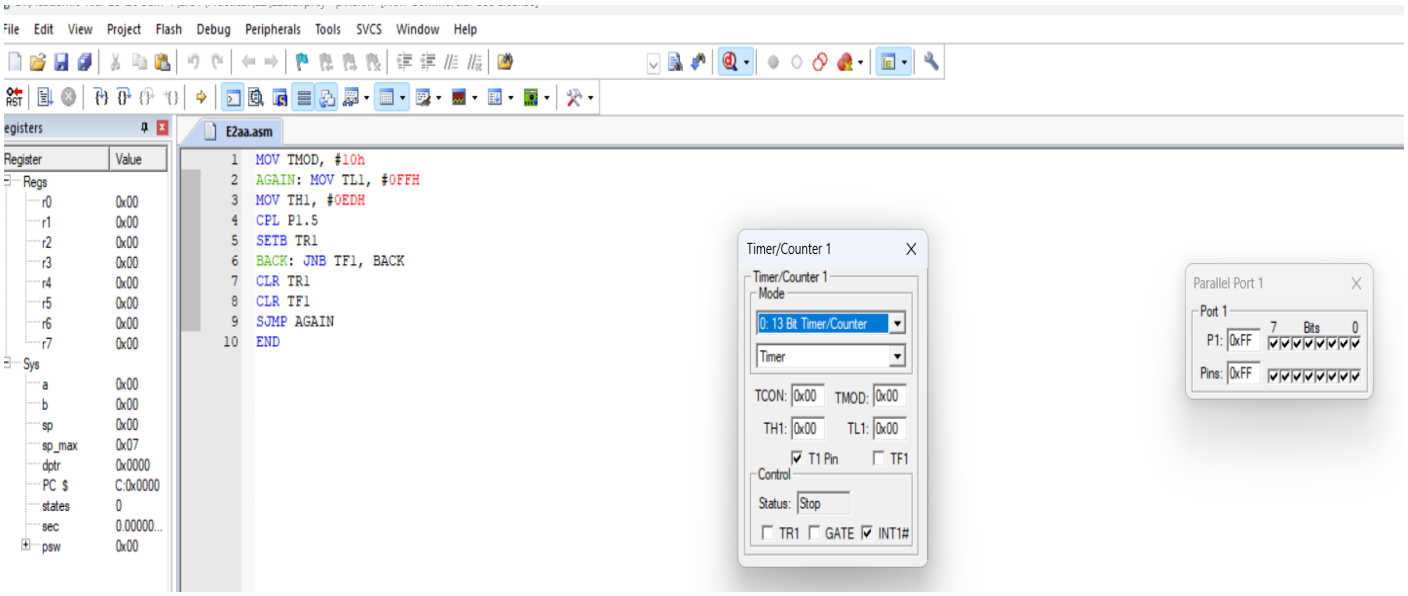
Name	Description	
7	TF1 (Timer 1 Overflow Flag)	Set to 1 when Timer 1 overflows else 0.
6	TR1 (Timer 1 Run Control Bit)	Set to 1 to start Timer 1, and set to 0 to stop Timer 1.
5	TF0 (Timer 0 Overflow Flag)	Set to 1 when Timer 0 overflows else 0.
4	TR0 (Timer 0 Run Control Bit)	Set to 1 to start Timer 0, and set to 0 to stop Timer 0.
3	IE1 (Interrupt 1 Edge Flag)	Set to 1 when an external interrupt 1 occurs else 0.
2	IT1 (Interrupt 1 Type Control Bit)	Set to 1 to configure external interrupt 1 as edge-triggered, and set to 0 for level-triggered.
1	IE0 (Interrupt 0 Edge Flag)	Set to 1 when an external interrupt 0 occurs else 0.
0	IT0 (Interrupt 0 Type Control Bit)	Set to 1 to configure external interrupt 0 as_edge-triggered, and set to 0 for level-triggered.

***CODE*:**

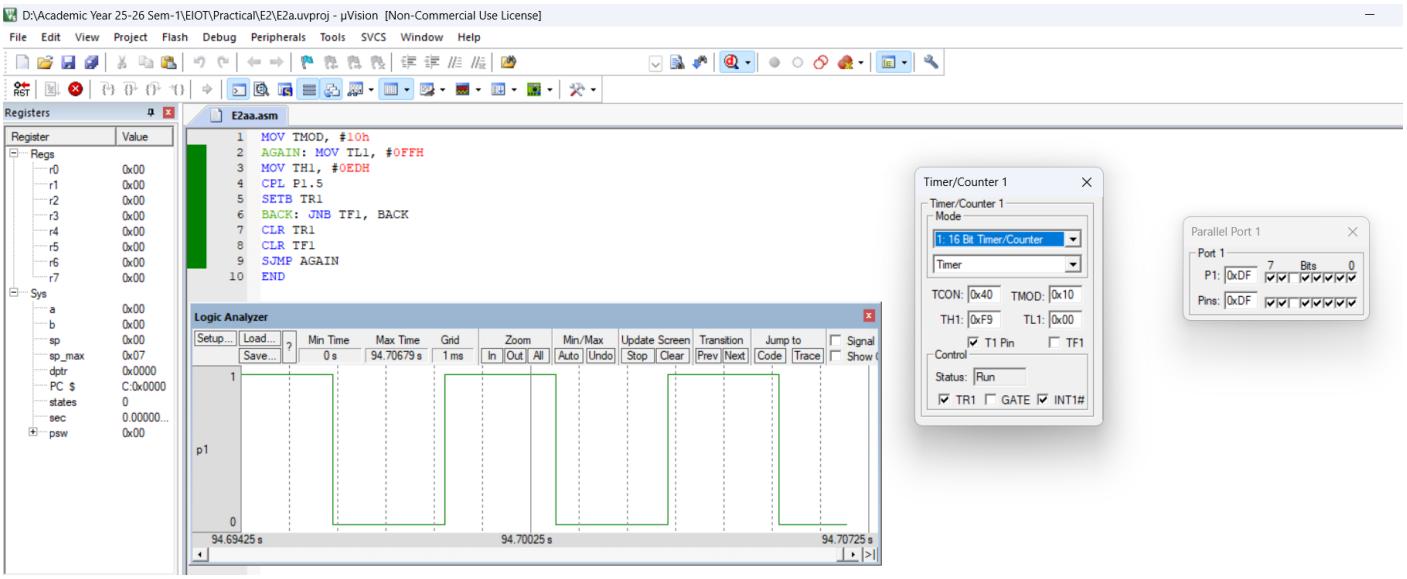
Program	Comments
Toggling of p1.5 with the delay of 5ms without interrupts using timer 1	
ORG 0000H	
MOV TMOD, #10h	Set Timer 1 to operate in Mode 1 (16-bit timer mode).
AGAIN: MOV TL1, #0FFH	Load Timer 1 registers with the desired delay value
MOV TH1, #0EDH	
CPL P1.5	Toggle P1.5 by completing the current state
SETB TR1	Set Timer 1 Run bit
BACK: JNB TF1, BACK	Continuously check the Timer 1 Overflow
CLR TR1	Clearing the Timer run(Stop the timer)
CLR TF1	Clearing the Overflow(for the next timer cycle))
SJMP AGAIN	Repeat the entire process
END	

***RESULT*:**

Before Execution:



After Execution:



Conclusion:

Experiment-4

AIM:

Toggling of p1.2 with the delay of 5ms with interrupts using timer 0

SOFTWARE REQUIRED: Keil µvision

***ALGORITHM*:**

Algorithm for delay of 5ms with interrupts using timer 0

Step 1: Start(Main Program)

Step 2: Configur Timer 0 to operate in Mode 1 (16-bit timer mode).

Step 3: Load Timer 1 registers with the desired delay value into the Timer 0 registers, TH0 (high byte) and TL0 (low byte).

Step 4: Enable the global interrupt by setting the EA bit in the IE register.

Step: 5: Enable the Timer 0 interrupt by setting the ET0 bit in the IE register.

Step 6: Start Timer 0: Set the TR0 bit in the TCON register to start Timer 0 and Enter an infinite loop

Step 7: END

Timer 0 Interrupt Service Routine (ISR)

This routine is automatically executed when Timer 0 overflows (reaches its maximum count and resets to 0) and the interrupt is enabled:

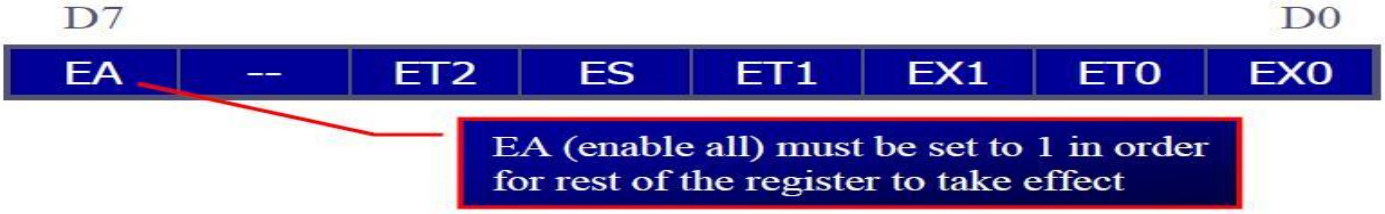
Step 1: Select the address of Interrupt T0 (000B)

Step 2: Toggle P1.2: Complement the state of the P1.2 pin

Step 3: Reload the registers TL and TH with initial count

Step 4: RETI instruction returns control to the main program loop

Interrupt Enable Register



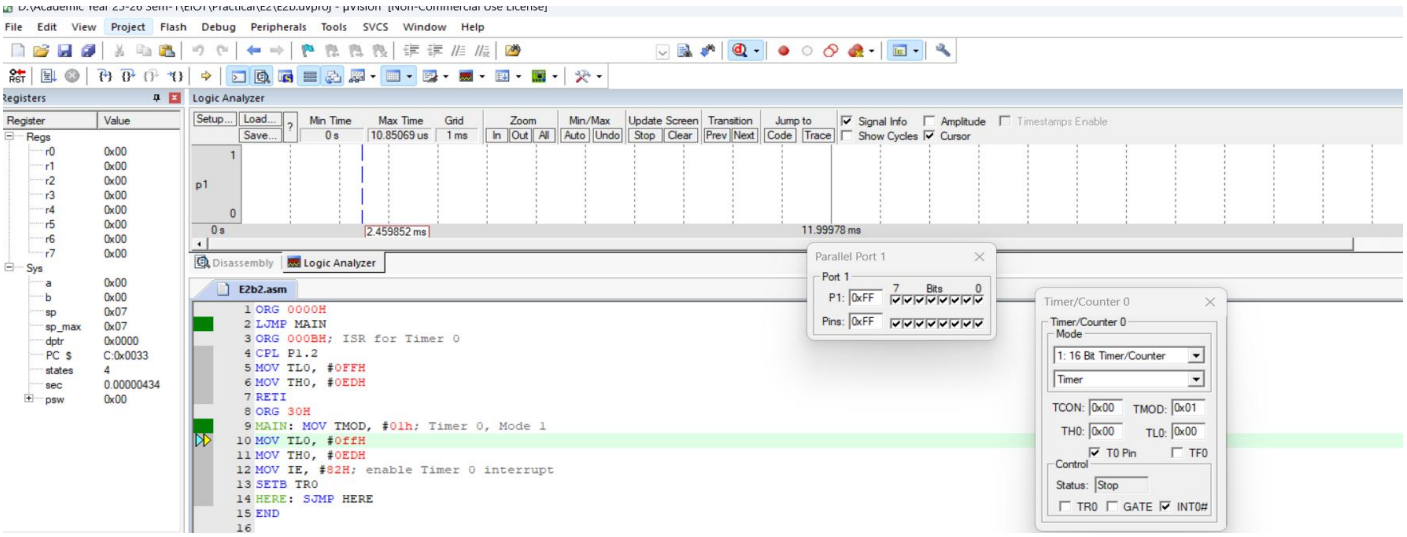
EA	IE.7	Disables all interrupts
--	IE.6	Not implemented, reserved for future use
ET2	IE.5	Enables or disables timer 2 overflow or capture interrupt (8952)
ES	IE.4	Enables or disables the serial port interrupt
ET1	IE.3	Enables or disables timer 1 overflow interrupt
EX1	IE.2	Enables or disables external interrupt 1
ET0	IE.1	Enables or disables timer 0 overflow interrupt
EX0	IE.0	Enables or disables external interrupt 0

***CODE*:**

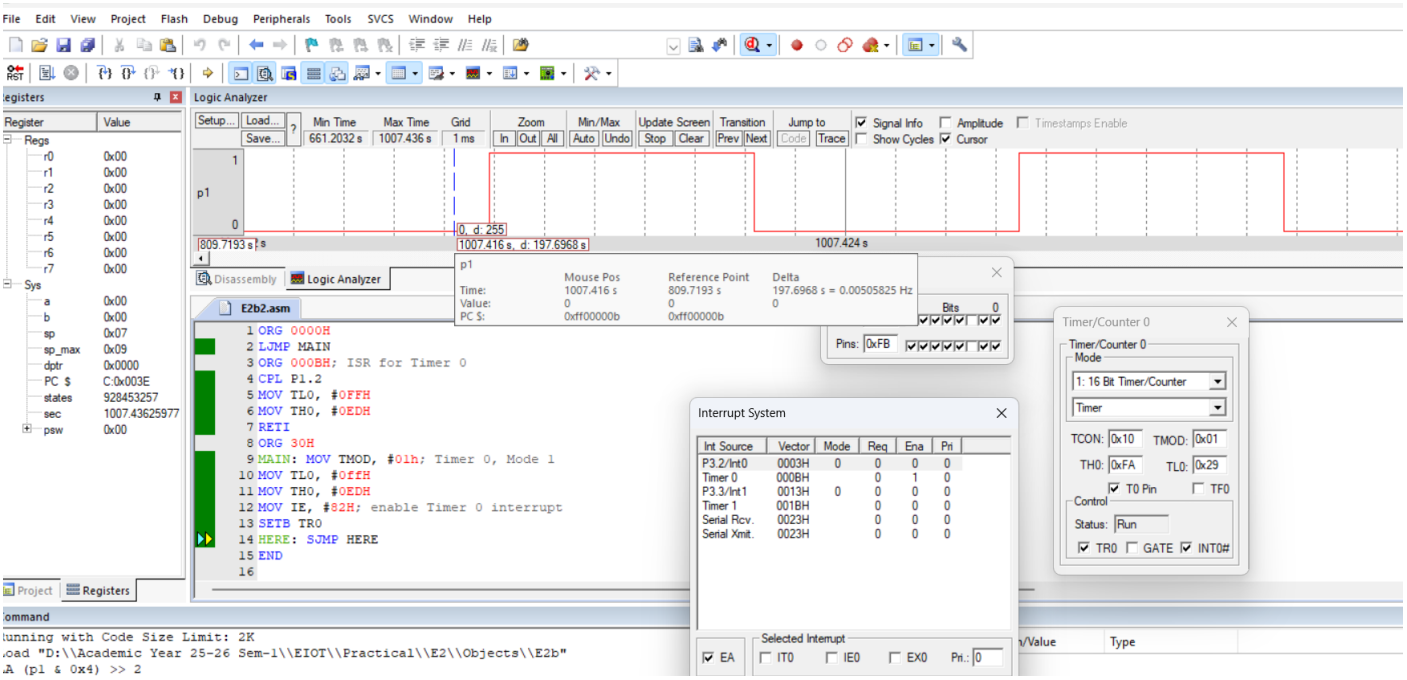
Program	**Comments**
Toggling of p1.2 with the delay of 5ms with interrupts using timer 0	
ORG 0000H	Initialization of the starting address
LJMP MAIN	Long jump to main program
ORG 000BH	ISR for Timer 0
CPL P1.2	Toggle or complement the state of P1.2
MOV TL0, #0FFH	Load calculated count for delay in Timer, lower byte
MOV TH0, #0EDH	in lower register and higher byte in higher register
RETI	Return from Interrupt
ORG 30H	Initialization of the main program address
MAIN: MOV TMOD,#01h	Timer 0, Mode 1
MOV TL0, #0FFh	Delay count
MOV TH0, #0EDH	
MOV IE, #82H;	Enable the global interrupt , enable Timer 0 interrupt
SETB TR0	
HERE: SJMP HERE	
END	

RESULT:

Before Execution:



After Execution:



Conclusion:

Experiment-5

AIM:

Count no. of pulses without interrupts using counter 0

SOFTWARE REQUIRED: Keil µvision

***ALGORITHM*:**

Algorithm for Count no. of pulses using counter 0

Step 1: Start(Main Program)

Step 2: Initialization /Configure Timer 0 for Counter Mode (Mode 1, 16-bit counter)

Step 3: Configure the input pin to behave like external pulse source (P3.4 for Timer 0)

Step 4: Initialize the Timer 0 registers (TH0 and TL0) with an initial count zero.

Step 5: Start Timer 0 by Set the TR0 bit in the TCON register (TCON.4)

Step 6: Continuously read the values in TL0 and TH0 to get the current count of pulses.

Step 7: Display the Count on Port 1 and Port 2 and repeat it.

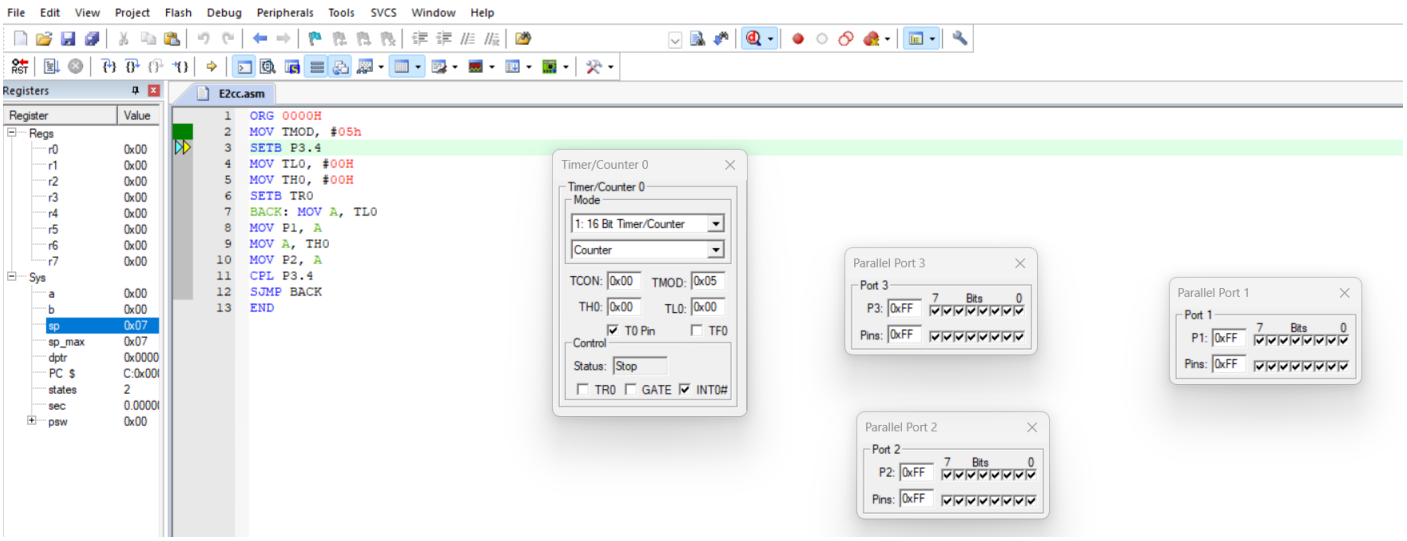
Step 8: END

***CODE*:**

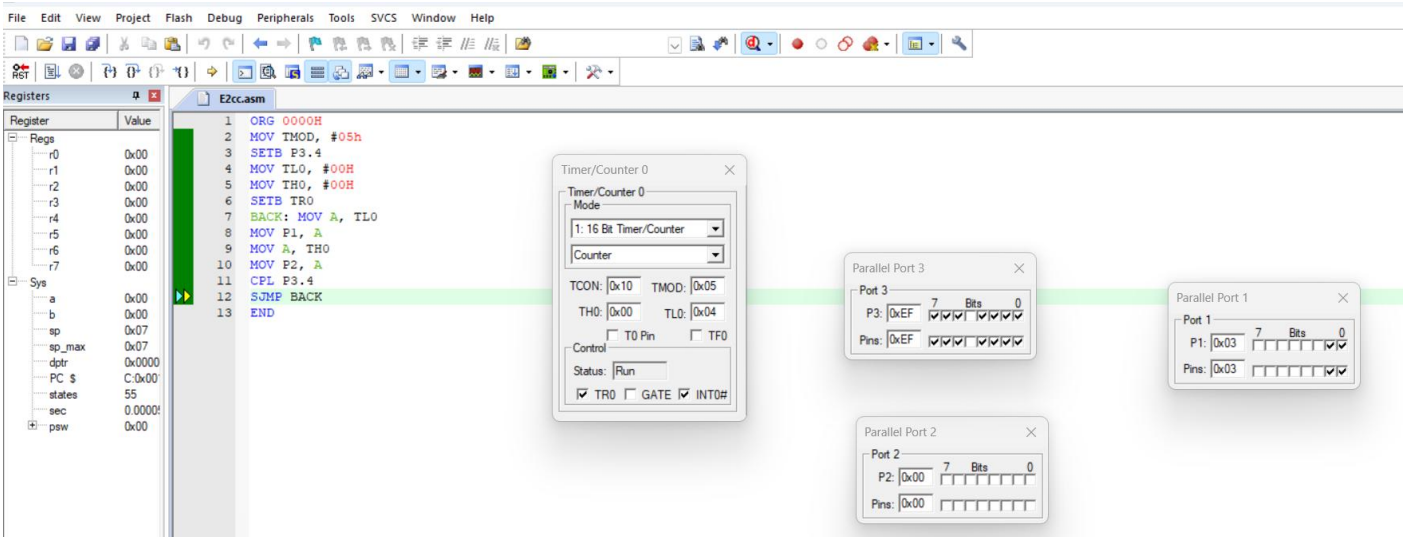
Program	**Comments**
Count no. of pulses without interrupts	
ORG 0000H	
MOV TMOD, #05h	Set Timer 0 for Counter Mode 1 (16-bit counter)
SETB P3.4	Set the P3.4 for external pulse source
MOV TL0, #00H	Load the initial count value into TH0 and TL0 to set
MOV TH0, #00H	the starting point
SETB TR0	Set the TR0 bit in the TCON register
BACK: MOV A, TL0	Move the lower value of count in A
MOV P1, A	Display lower count on Port 1
MOV A, TH0	Move the higher value of count in A
MOV P2, A	Display higher count on Port 1
SJMP BACK	Repeat the loop
END	

***RESULT*:**

Before Execution:



After Execution:



Conclusion:

Experiment-6

AIM: Count no. of pulses with interrupts using counter 1

SOFTWARE REQUIRED: Keil µvision

***ALGORITHM*:**

Algorithm for Count no. of pulses using counter 1 with an interrupt

Main program loop

Step 1: Start(Main Program)

Step 2: Initialization /Configure Timer 1 for Counter Mode (Mode 1, 16-bit counter):

Step 3: Configure the input pin by connecting it to the external pulse source (P3.5 for Counter 1)

Step 4: Initialize Timer 1 with a pre-count value: Load the TL1 and TH1 registers with 0FAH and 0FFH, respectively.

Step 5 : Enable Counter 1 interrupt in the IE register by loading it with 88H.

Step 6: Set the TR1 bit to start the counter and continuously set the output of Port 1 to 0AAH

Step 7: END

Counter 1 Interrupt Service Routine (ISR)

This routine is automatically executed when Counter 1 overflows (reaches its maximum count and resets to 0) and the interrupt is enabled:

Step 1: Select the address of Interrupt T1 at address 001BH

Step 2: Set the output of Port 1 to 55H to signal that the interrupt has been handled.

Step 3: Load the TL1 and TH1 registers with 0FAH and 0FFH again to prepare for the next counting cycle

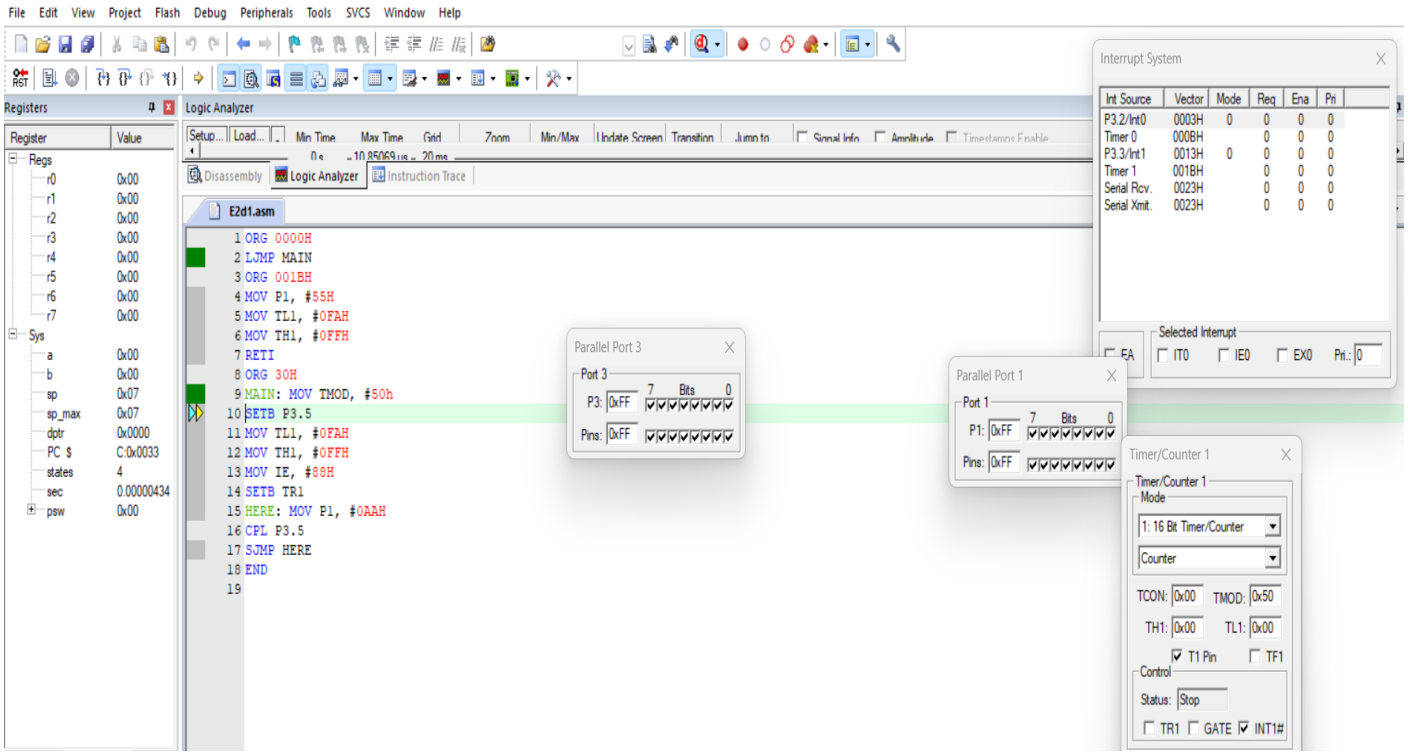
Step 4: RETI instruction returns control to the main program loop

***CODE*:**

Program	**Comments**
Count no. of pulses with interrupts using counter 1	
ORG 0000H	Initialization of the starting address
LJMP MAIN	Long jump to main program
ORG 001BH	ISR for Counter 1
MOV P1, #55H	Port 1 to 55H to signal that the interrupt has been handled.
MOV TL1, #0FAH	Load the count TL1 and TH1 registers with 0FAH and
MOV TH1, #0FFH	0FFH, respectively
RETI	Return from Interrupt
ORG 30H	Initialization of the main program address
MAIN: MOV TMOD, #50h	Counter 1, Mode 1
SETB P3.5	Pulse source (P3.5 for counter 1) is configured as an input
MOV TL1, #0FAH	
MOV TH1, #0FFH	
MOV IE, #88H	Enable the global interrupt , enable Timer 1 interrupt
SETB TR1	Start the counter
HERE: MOV P1, #0AAH	Port 1 to AAH to signal that the main program has been
SJMP HERE	handled,
END	

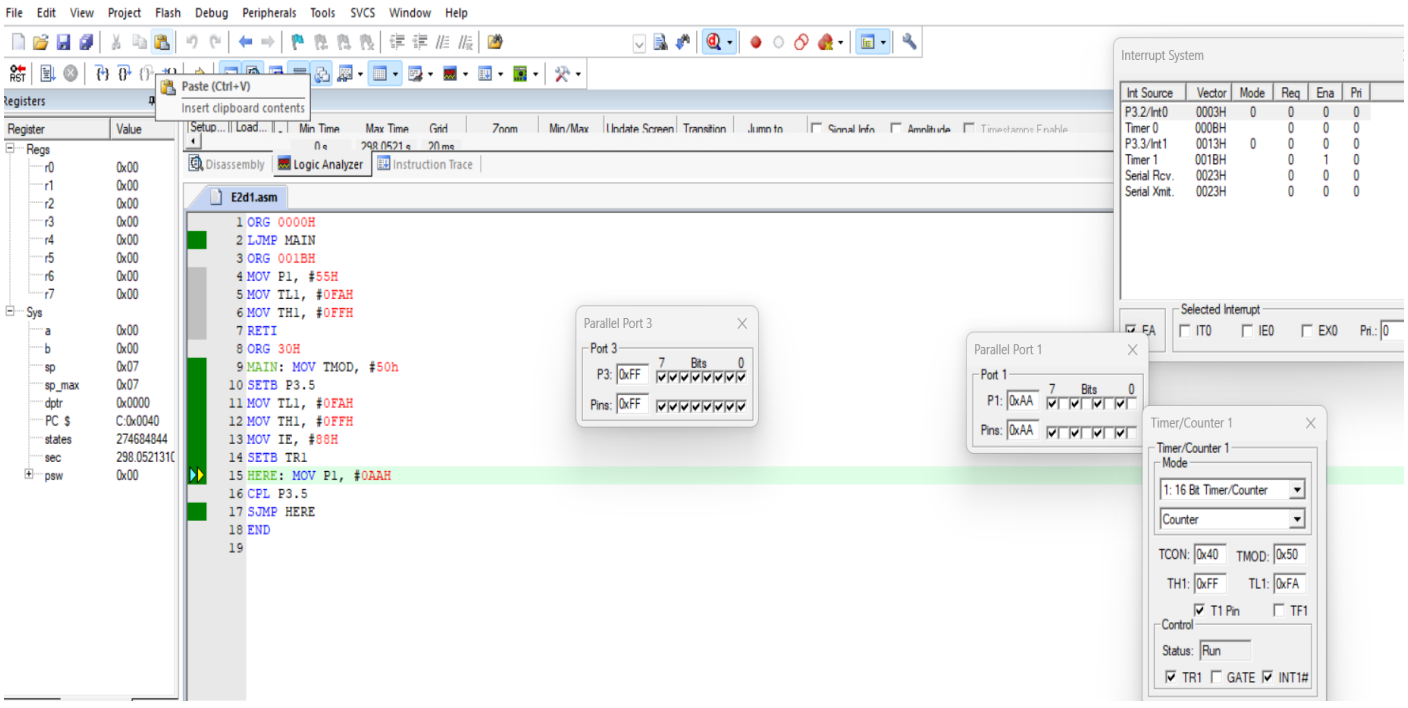
***RESULT*:**

Before Execution:

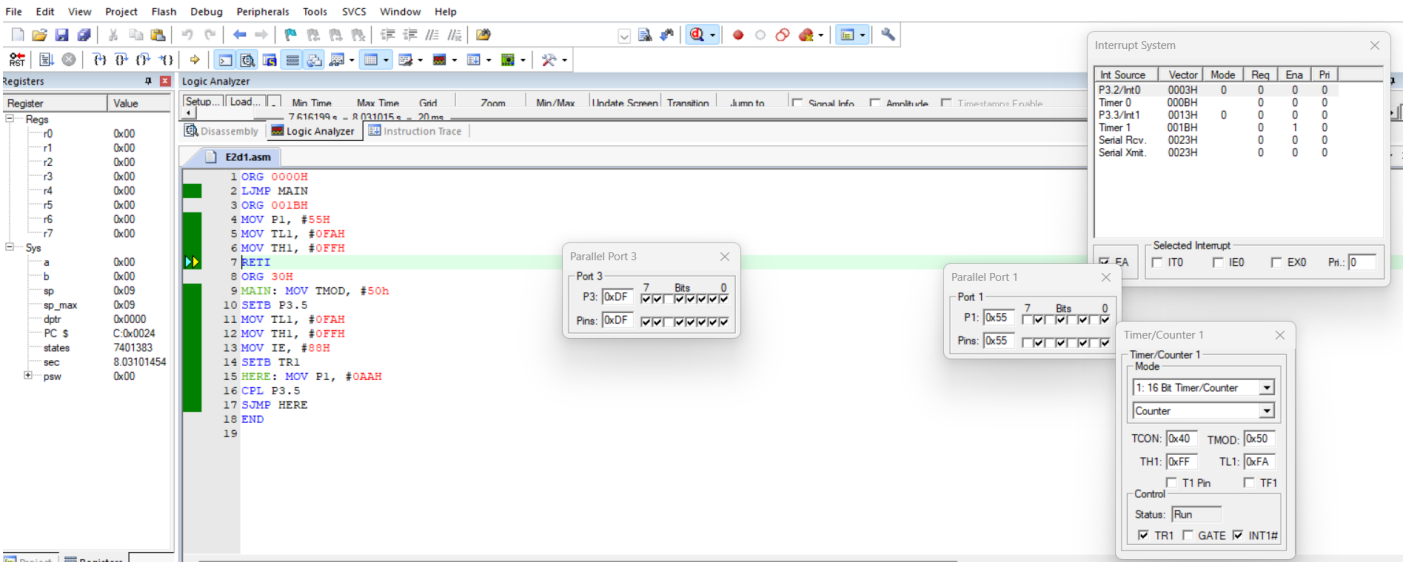


After Execution:

Main Program



Interrupt Service Routine (ISR) - at address 001BH



Conclusion:

Experiment-7

AIM: Program to transfer data serially with baud rate of 9600

SOFTWARE REQUIRED: Keil µvision

***ALGORITHM*:**

Algorithm for transferring data serially with a baud rate of 9600

Step 1: Start

Step 2: 1. Initialize Serial Communication and load the TMOD by 20H to configure Timer 1 in Mode 2 (8-bit auto-reload mode).

Step 3: Loads the TH1 register with the value -3 (equivalent to 0xFD in hexadecimal) to generate the baud rate (9600) for serial communication.

Step 4: Load 50H in the Serial Control (SCON) register to work in Serial Mode 1 (8-bit UART with variable baud rate).

Step 5: Start Timer 1 to begin the baud rate generation by setting TR1 1

Step 6: Transmit the character that is written in the SBUF Register

Step 7: Wait for Data Transmission by continuously checking the TI (Transmit Interrupt) flag in the SCON register.

Step 8: TI flag is cleared by the program to prepare for the next character transfer

Step 9: Repeat for Continuous Transmission

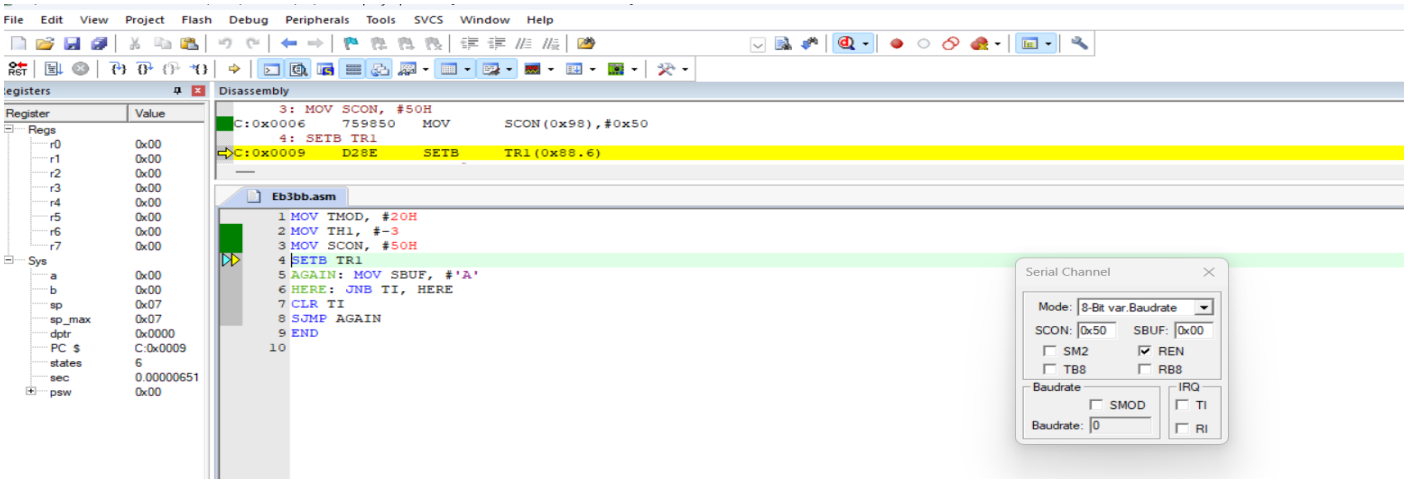
***CODE*:**

Program	**Comments**
To transfer data serially with baud rate of 9600	
MOV TMOD, #20H	Configure Timer 1 in Mode 2 (8-bit auto-reload mode).
MOV TH1, #-3	TH1 register with the value -3 for baud rate 9600
MOV SCON, #50H	Serial Mode 1 (8-bit UART with variable baud rate
SETB TR1	Start Timer
AGAIN: MOV SBUF, #'A'	Transmit the character 'A' that is written in the SBUF
HERE: JNB TI, HERE	Continuously checking the TI for transmission completion

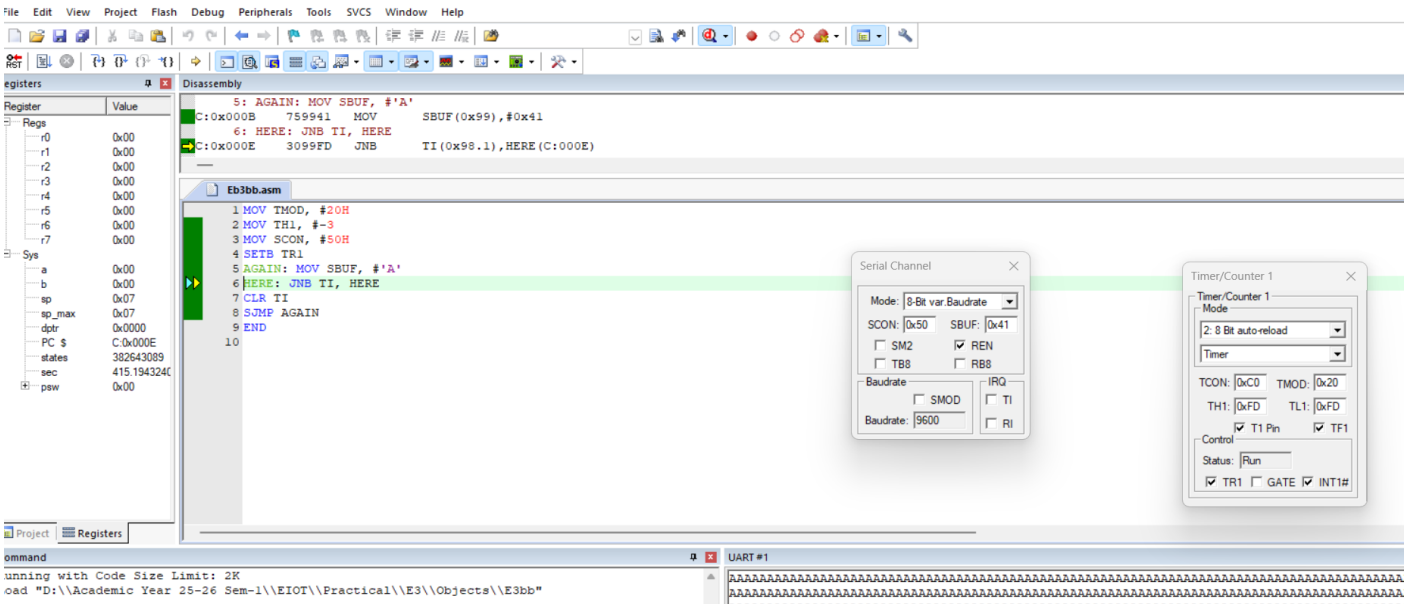
CLR TI SJMP AGAIN END	Clear Transmit Interrupt Repeat the transmission

***RESULT*:**

Before Execution:



After Execution:



Conclusion

Experiment-8

AIM: Program to receive data serially with baud rate of 9600

SOFTWARE REQUIRED: Keil µvision

***ALGORITHM*:**

Algorithm for *Program to receive data serially with baud rate of 9600*

Step 1: Start

Step 2: 1. Initialize Serial Communication and load the TMOD by 20H to configure Timer 1 in Mode 2 (8-bit auto-reload mode).

Step 3: Loads the TH1 register with the value -3 (equivalent to 0xFD in hexadecimal) to generate the baud rate (9600) for serial communication.

Step 4: Load 50H in the Serial Control (SCON) register to work in Serial Mode 1 (8-bit UART with variable baud rate).

Step 5: Start Timer 1 to begin the baud rate generation by setting TR1 1

Step 6: Wait for Data Reception by continuously checking the RI (Receive Interrupt) flag in the SCON register

Step 7: Once RI is set, the received 8-bit data is moved from the Serial Buffer (SBUF) register to the Accumulator (A) and then display using P1.

Step 8: RI flag is cleared by the program to prepare for the reception of the next character

Step 9: Repeat for Continuous Reception

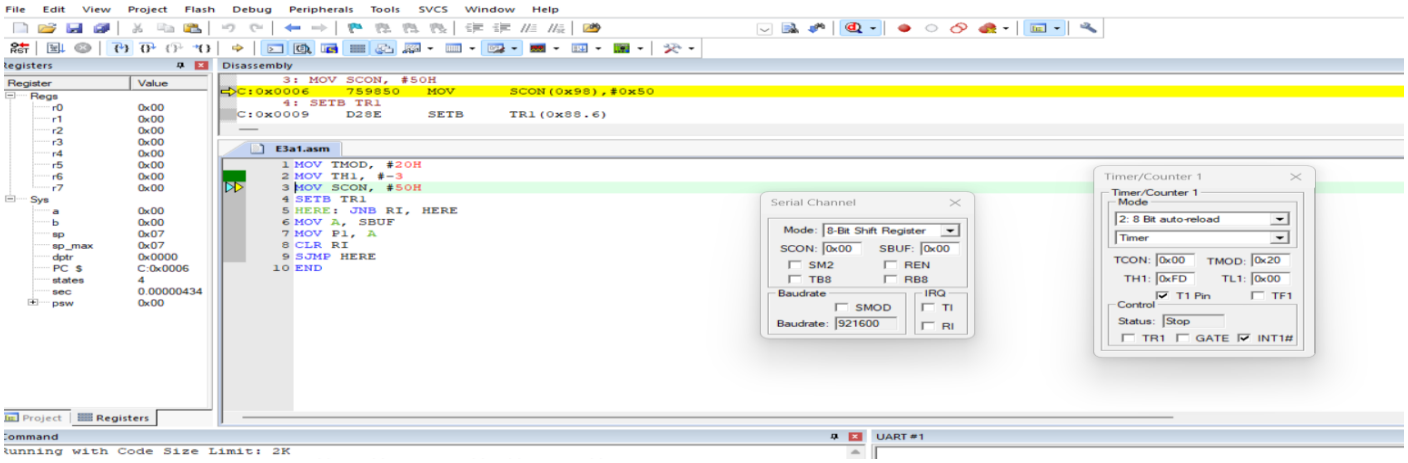
***CODE*:**

Program	**Comments**
Program to receive data serially with baud rate of 9600	
ORG 0000h	
MOV TMOD, #20H	Configure Timer 1 in Mode 2 (8-bit auto-reload mode).
MOV TH1, #-3	TH1 register with the value -3 for baud rate 9600
MOV SCON, #50H	Serial Mode 1 (8-bit UART with variable baud rate
SETB TR1	Start Timer

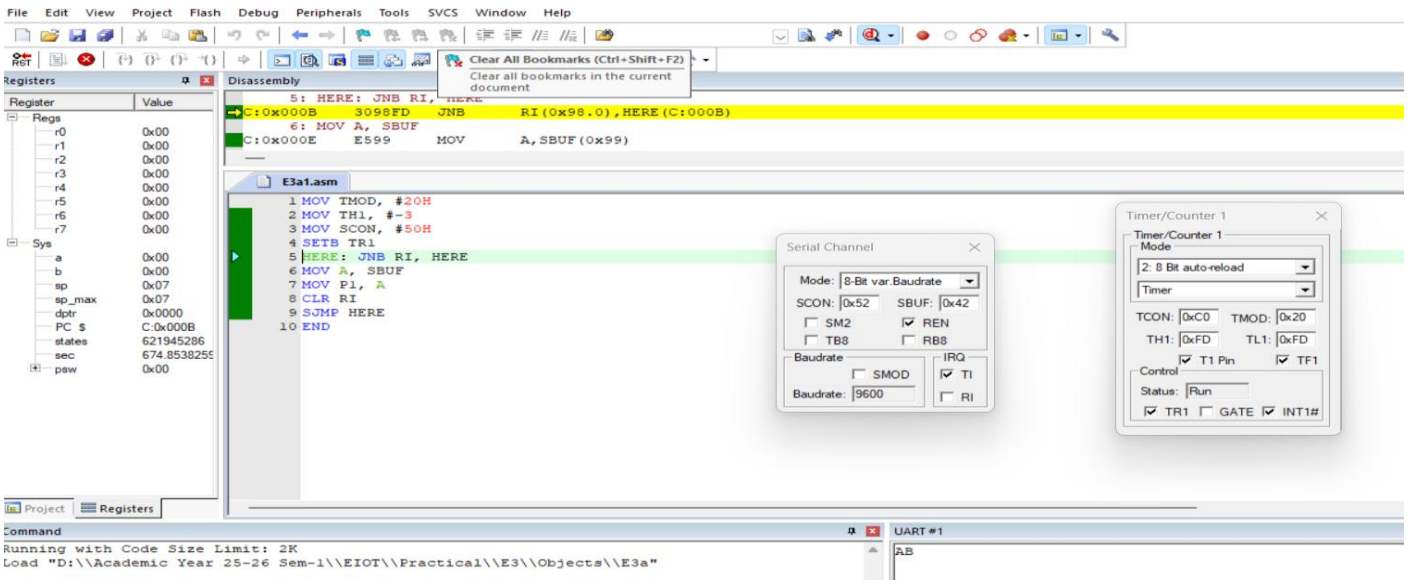
HERE: JNB RI, HERE	Wait for Data Reception by continuously checking the RI
MOV A, SBUF	Move the data from the Serial Buffer (SBUF) register to
MOV P1, A	the Accumulator (A) and then display using P1.
CLR RI	Clear Receive Interrupt
SJMP HERE	Repeat the transmission
END	

***RESULT*:**

Before Execution:



After Execution:



Conclusion :

Experiment-9

AIM: Sequence Generator Using Serial Interface in 8051.

SOFTWARE REQUIRED: Keil µvision

***ALGORITHM*:**

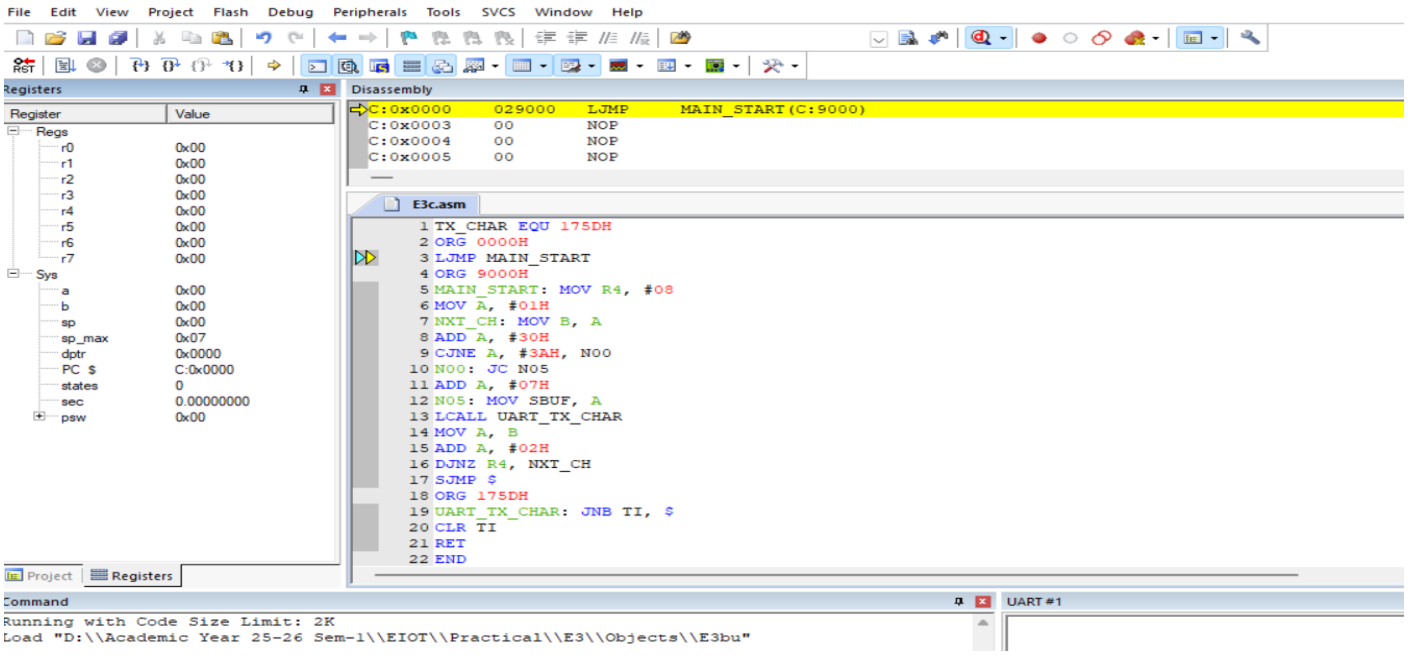
Algorithm for Sequence Generator

***CODE*:**

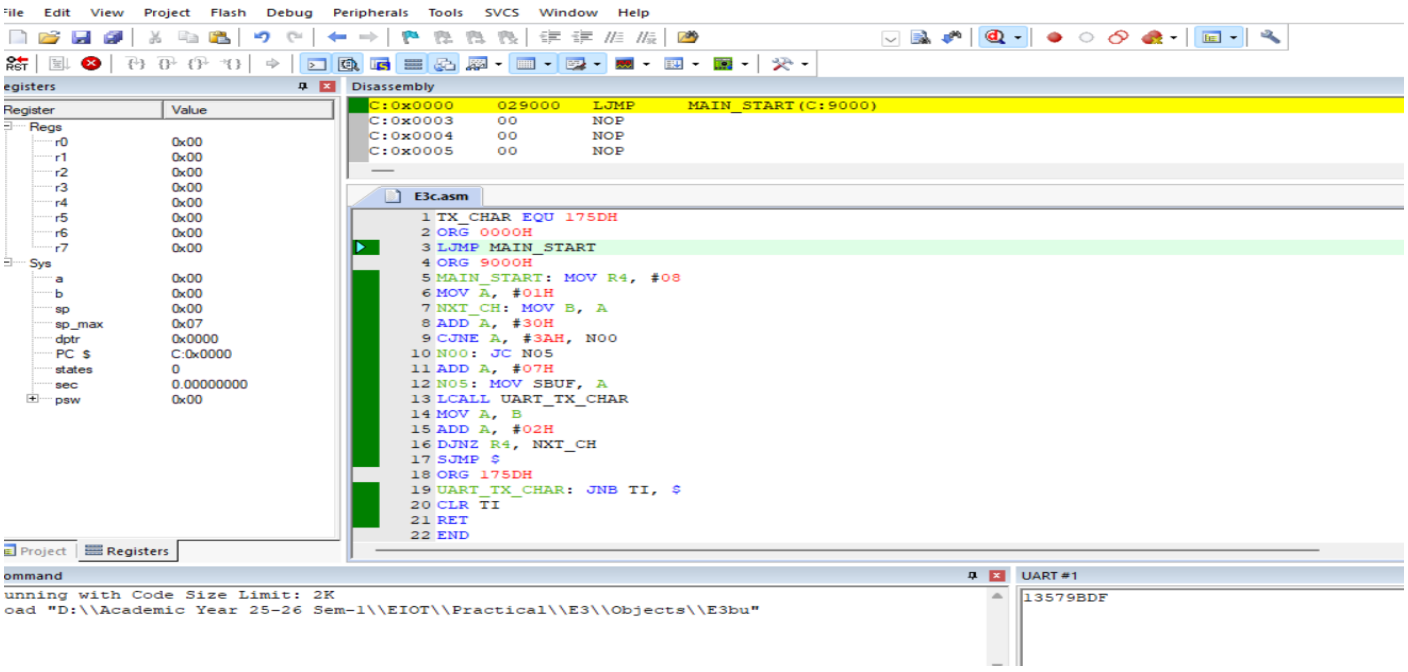
Program	Comments
Sequence Generator	
TX_CHAR EQU 175DH	Define the subroutine address to send a character
ORG 0000H	Origin of Program
LJMP MAIN_START	Jump to main program
ORG 9000H	Program starts at memory address 9000H
MAIN_START: MOV R4, #08	Initialize loop counter R4 with 8
MOV A, #01H	Load A with 1 (initial number to convert and send)
NXT_CH: MOV R6, A	Store current value of A into R6 (preserve original value)
MOV B, A	Store current value of A in B also
ADD A, #30H	Convert the number in A to ASCII by adding 0x30
CJNE A, #3AH, N00	Compare A with '9'+1 (0x3A); if not equal, jump to N00
N00: JC N05	If A < 0x3A (i.e., still a digit), jump to N05
ADD A, #07H	Else (A was a letter A–F), adjust to get correct hex letter
N05: MOV SBUF, A	Store the result in SBUF
LCALL UART_TX_CHAR	Call the subroutine to transmit the character
MOV A, B	Restore original value from B to A
ADD A, #02H	Increment by 2 for next iteration
DJNZ R4, NXT_CH	Decrement R4, loop until 8 iterations complete
SJMP $	End program (infinite loop)
ORG 175DH	
UART_TX_CHAR: JNB TI, $	UART Transmit Subroutine (Wait for TI flag)
CLR TI	Clear transmit interrupt flag
RET	Return from ISR
END	

***RESULT*:**

Before Execution:



After Execution:



Conclusion: