A. Familiarisation of 8051 assembly programming

A.1 Introduction to 8051 assembly programming

CPU can work only in binary. A program consists of 0s and 1s is called *machine language*. Although the hexadecimal system was used as a more efficient way to represent binary numbers, the process of programming in machine code was still cumbersome for humans. Assembly languages were developed that provided mnemonics for the machine code instruction that made programming faster and less prone to error. The term *mnemonics* refer to codes and abbreviations that are relatively easy to remember. Assembly language programs must be translated into machine codes by a program called an *assembler*. Assembly language is referred to as low-level language because it deals directly with the internal structure of the CPU. To program in Assembly language, the programmer must know all the registers of the CPU and the size of them, as well as other details.

A.2 Structure of Assembly language

An Assembly language instruction basically consists of four fields:

[label:] mnemonic [operands] ;comment

Brackets indicate that the field is optional, and not all lines have them. When the field has content, brackets should not be typed in.

Table A.2: Example of an assembly language program

```
[label:]    mnemonic  [operands]  ;comment
COUNT       EQU       25H         ;COUNT = 25H
            ORG       0H          ;start (origin) at location 0
            MOV       R5,#COUNT   ;load 25H into R5
            MOV       R7,#34H     ;load 34H into R7
            MOV       A,#0        ;load 0 into A

            ADD       A,R5        ;add contents of R5 to A,
                                  ;now A = A + R5

            ADD       A,R7        ;add contents of R7 to A,
                                  ;now A = A + R7

            ADD       A,#12H      ;add to A value 12H,
                                  ;now A = A + 12H
            NOP                   ;no operation
HERE:       SJMP      HERE        ;stay in this loop
            ORG       10H
DATA1:      DB        39H         ;
DATA2:      DB        "America"   ;
            END                   ;end of assembly source file
```

Note:

1. The label field allows the program to refer to a line of code by name. Any label referring to an instruction must be followed by a colon symbol, ":".
2. An assembly language instruction consists of a mnemonic, optionally followed by one, two or no operands. The operands are the data items being manipulated, and the mnemonics are the commands to the CPU. The mnemonic (instruction) and operand(s) fields together perform the real work of the program and accomplish the tasks for which the program was written.

   Instead of mnemonics and an operand, these two fields could contain assembler pseudo-instructions, or directives. e,g,

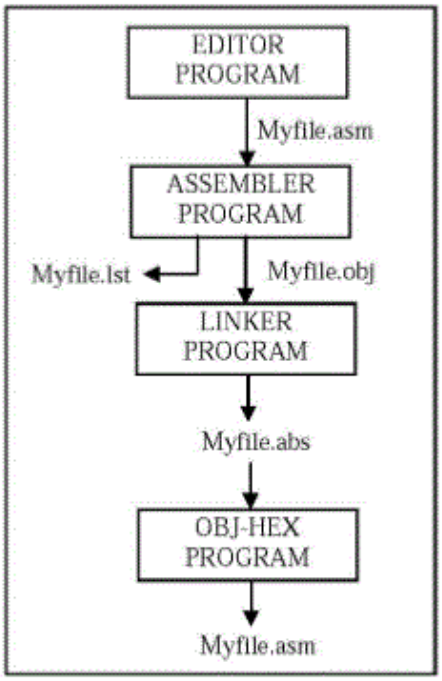
   - ORG (origin) – indicates the beginning of the address of the instructions. The number that comes after ORG can be either hex or decimal.
   - END – indicates to the assembler the end of the source assembly instructions.
   - EQU (equate) – used to define a constant without occupying a memory location. It does not set aside storage for a data item but associates a constant value with a data label so that when the label appears in the program. Its constant value will be substituted for the label.
   - DB (define byte) – used to define 8-bit data and store them in assigned memory locations. Define data can be in decimal, binary, hex, or ASCII formats.

3. The comment field begins with a semicolon comment indicator ";". Comments may be at the end of a line or on a line by themselves. They used to describe the program and make it easier for reading and understanding. The assembler ignores the comments, but they are essential to programmers.

A.3 Assembling and running of an 8051 program

The steps to create executable machine codes from assembly language program are outlined as follows.

- Use an editor program to create an assembly source program, e.g. myfile.asm
- The "asm" source file is then converted by an 8051 assembler to machine codes. Two files, an object file and a list file, are produced.
- A linker program takes the object file and produces an absolute object file with extension "abs" and fed it into a object to hex conversion program (OBJ-HEX) to create a hex program that is ready to be run by 8051.



Content of the List file of the assembly language program in Table A.2

```
LOC   OBJ        LINE   SOURCE

0025             1      COUNT  EQU   25H          ;COUNT = 25H
0000             2             ORG   0H           ;start (origin) at location 0
0000  7D25       3             MOV   R5, #COUNT   ;load 25H into R5
0002  7F34       4             MOV   R7, #34H     ;load 34H into R7
0004  7400       5             MOV   A, #0        ;load 0 into A
0006  2D         6             ADDA, R5           ;add contents of R5 to A, now A = A + R5
0007  2F         7             ADDA, R7           ;add contents of R7 to A, now A = A + R7
0008  2412       8             ADDA, #12H         ;add to A value 12H, now A = A + 12H
000A  00         9             NOP                ;no operation
000B  80FE       10     HERE:  SJMP  HERE         ;stay in this loop
0010             11            ORG   10H
0010  39         12     DATA1: DB    39H          ;
0011  416D6572   13     DATA2: DB    "America"    ;
0015  696361
                 14            END                ;end of assembly source file

SYMBOL TABLE LISTING
------ ----- -------

NAME     TYPE      VALUE     ATTRIBUTES

COUNT    N NUMB    0025H     A
DATA1    C ADDR    0010H     A
DATA2    C ADDR    0011H     A
HERE     C ADDR    000BH     A

REGISTER BANK(S) USED: 0

ASSEMBLY COMPLETE.  0 WARNING(S), 0 ERROR(S)
```

Original content of the source file

Corresponding machine codes for the mnemonics instructions

Indicates the memory locations that the machine codes being stored.

The warnings/errors messages indicate the result of the assembly process. Zero means that the program is successfully converted and ready to use, otherwise, correction on the source file is needed.

Content of the Hex file of the assembly language program in Table A.2

```
:0D0000007D257F3474002D2F24120080FEEA
:0800200039416D6572696361ED
:00000001FF
```

Separating the fields we get

:CC	AAAA	TT	Data	SS
:0D	0000	00	7D257F3474002D2F24120080FE	EA
:08	0020	00	39416D6572696361	ED
:00	0000	01	FF	

CC – the count byte. It tells the loader how many bytes are in the line.
AAAA – It is a 16-bit address which tells the loader where the first byte of data to be placed.
TT – Either 00 or 01. 00 means there are more lines to follow after this line. 01 means this is the last line.
Data – the real information and the length is 16 bytes in maximum.
SS – checksum byte which used for error checking.

B. Familiarisation of 8051 simulation software

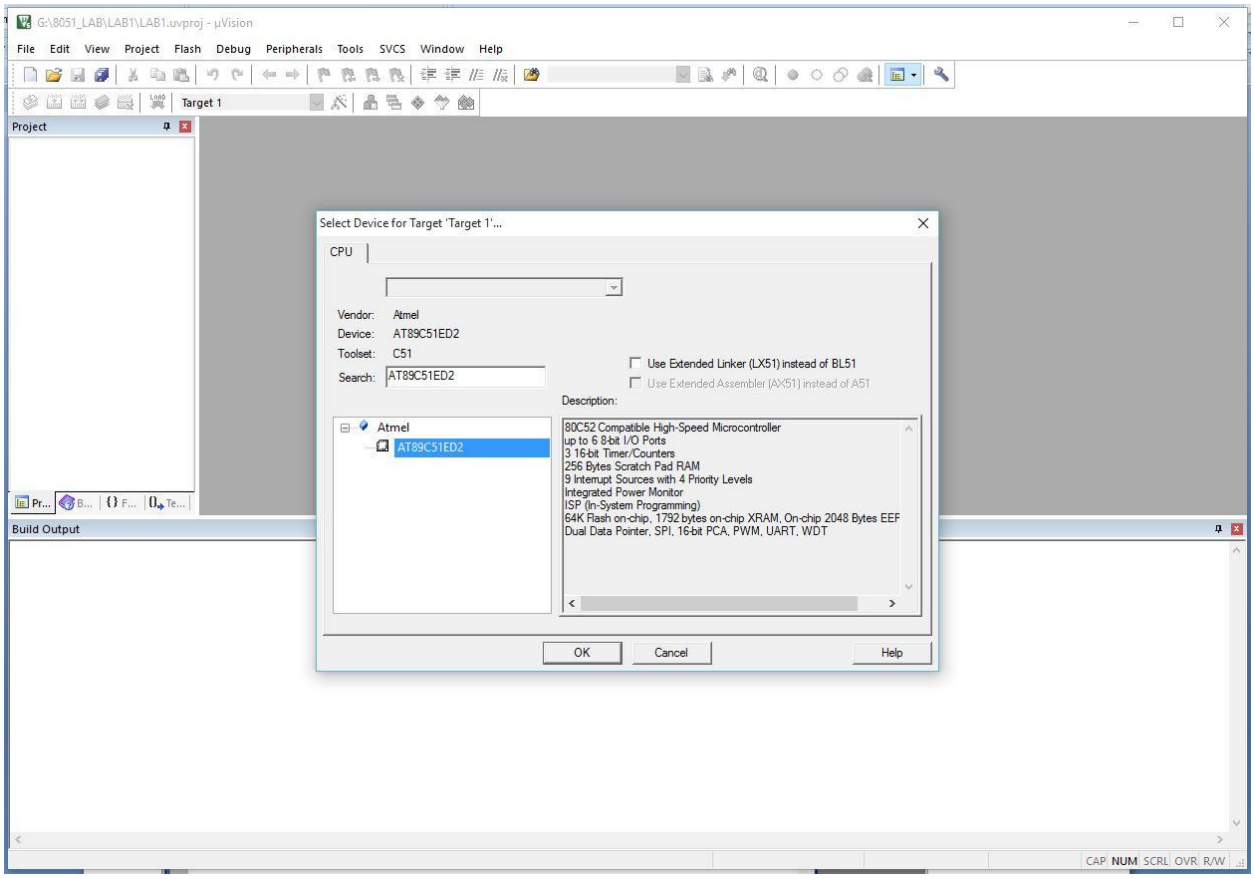
B.1 Open the 8051 simulation software

- Execute the program “UV4.exe” inside Folder “C:\Keil_v5\UV4”

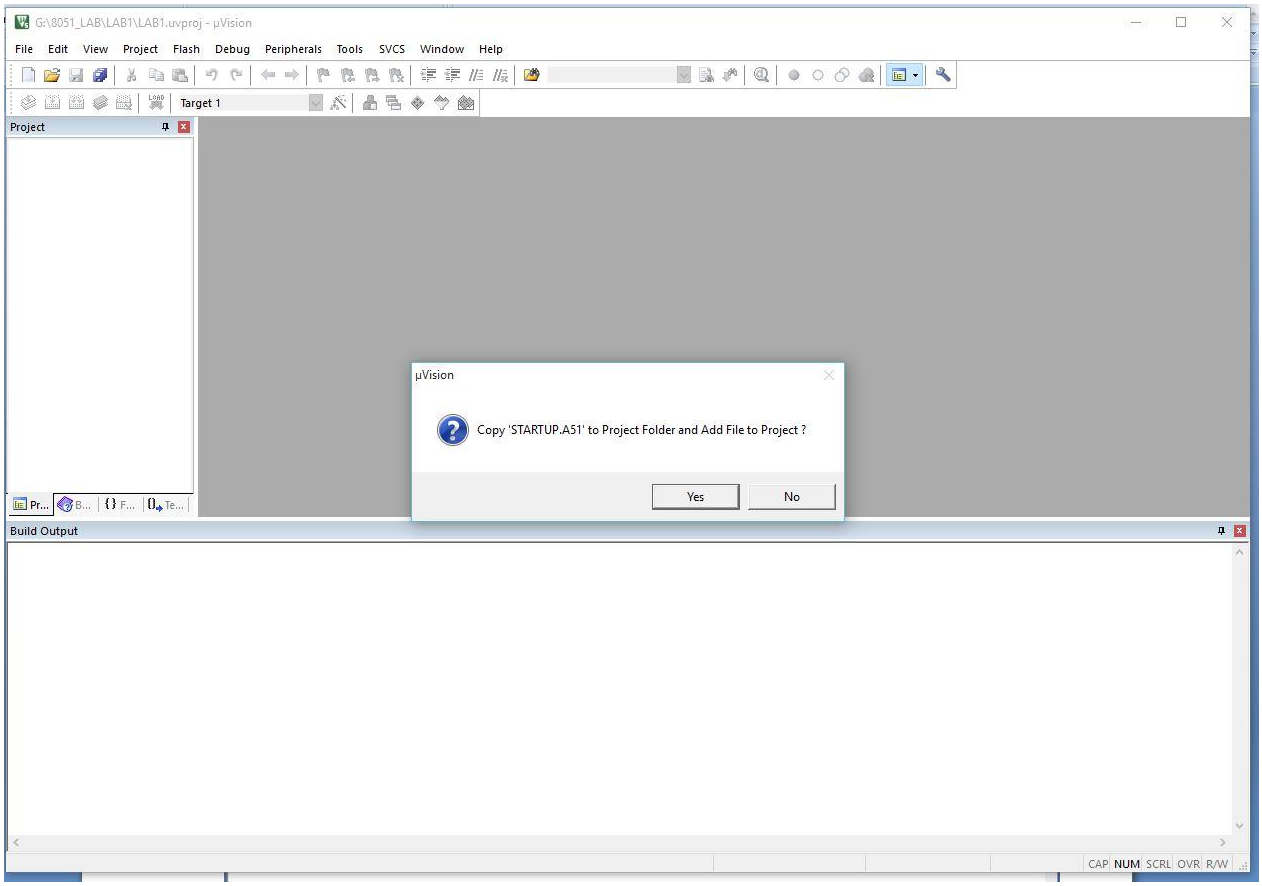
B.2 Editing/Create an assembly program

- Inside the program window of “µVision”, use “New µVision Project” command in the “Project” pull-down menu to create a new project. Go into the folder “8051_Lab”. Then type the file name “LAB1” and click save.

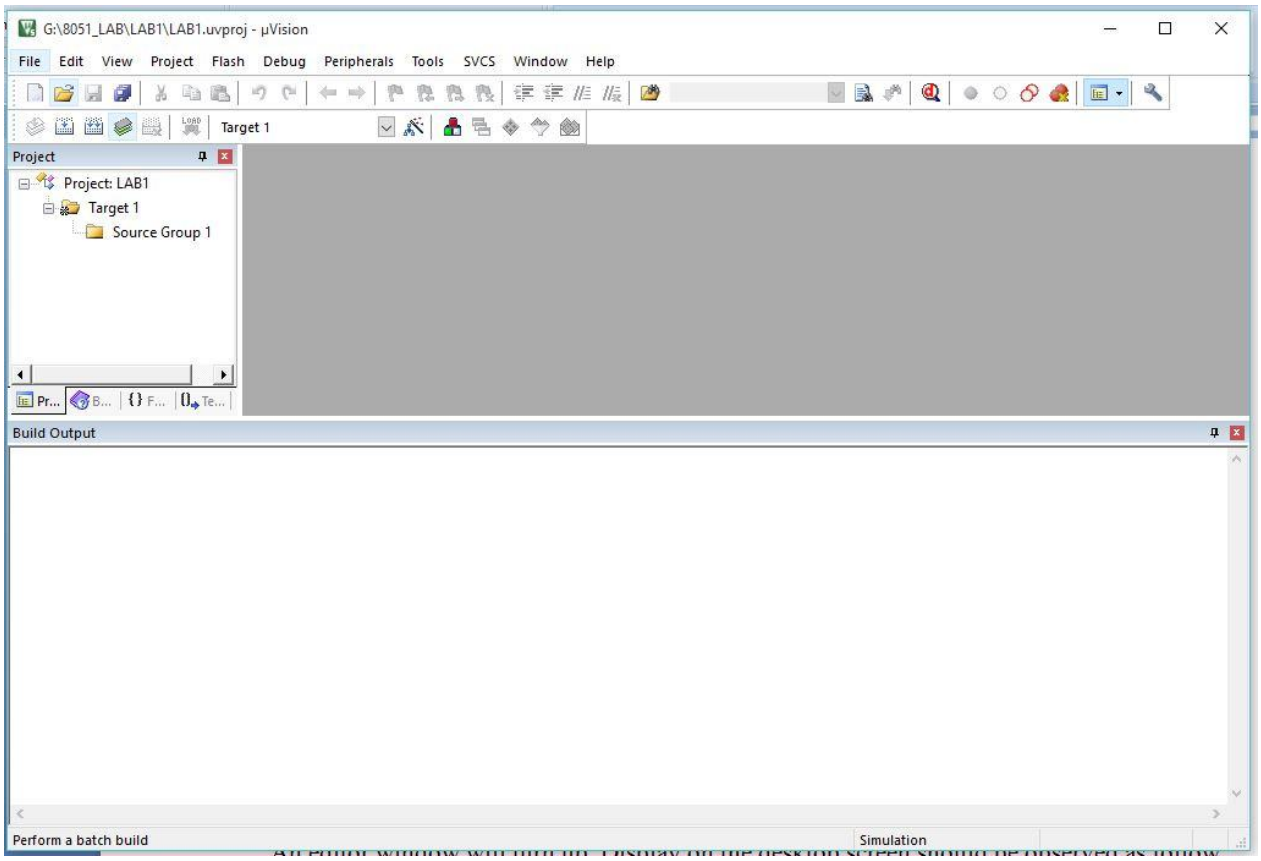
- A “Select Device for Target 'Target 1'... ” dialog box will appear to choose the desired microcontroller. In our course we will be using “AT89C51ED2”



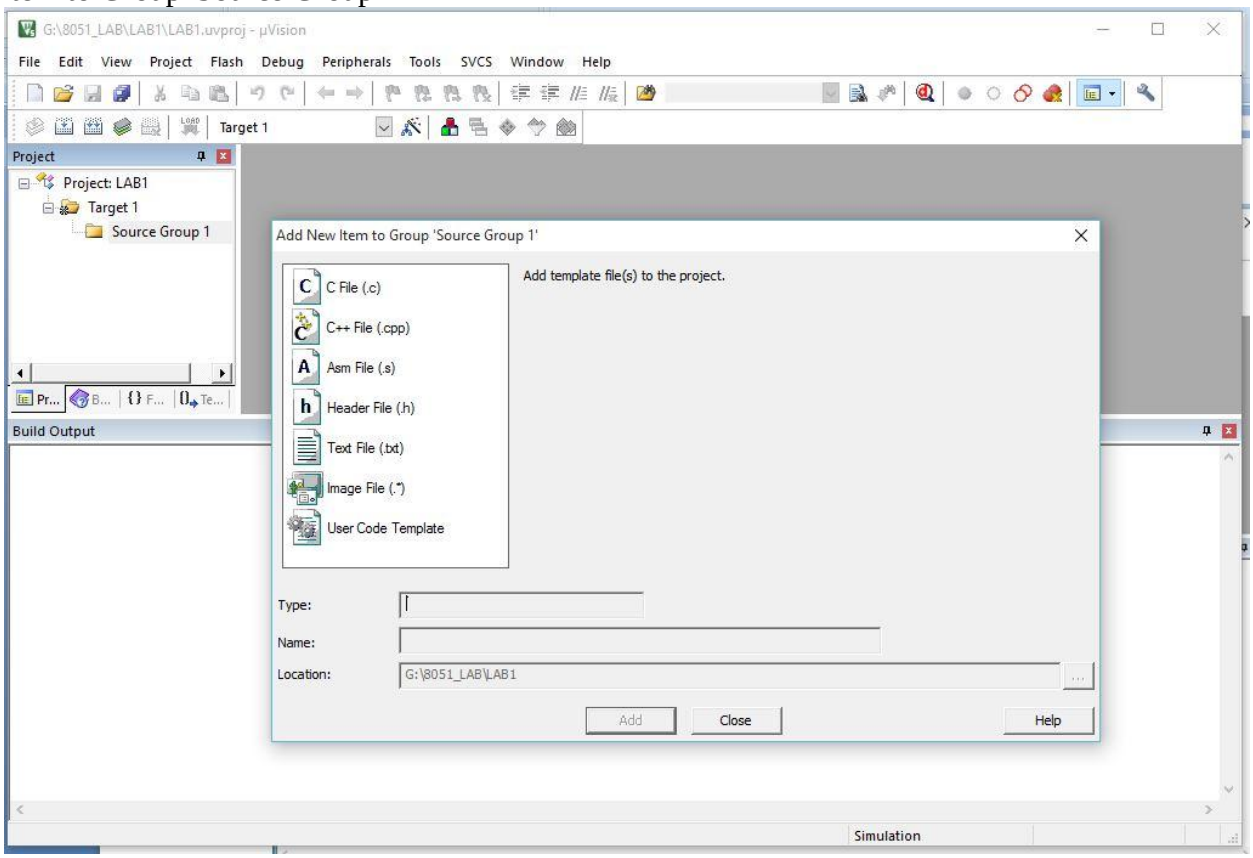
- A Yes/No Message box appear to ask to copy “Startup.A51” to project folder and Add File to project?, press “No”



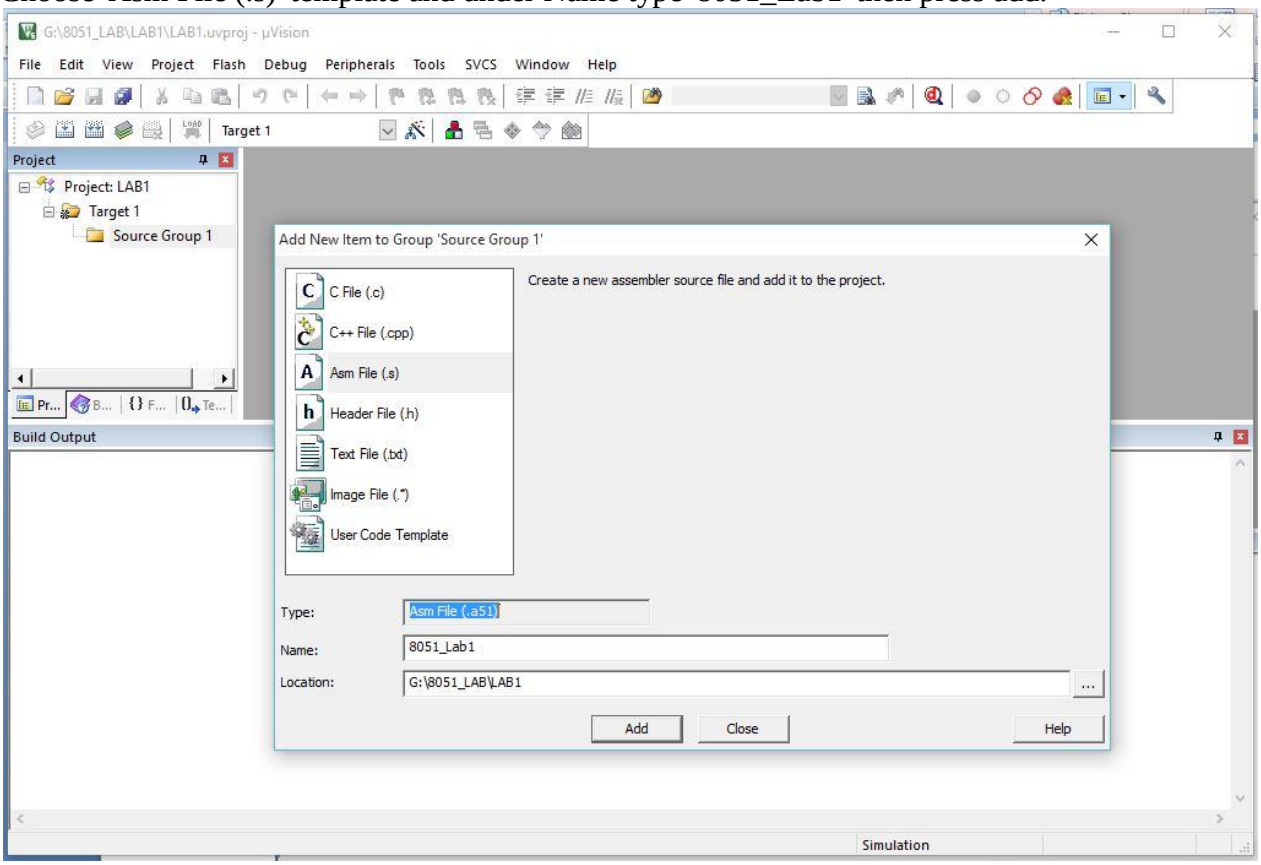
- The project is created and the following window appear



- In "Project" Window, Select "Source Group 1", right click with the mouse and choose "Add New Item to Group 'Source Group 1' "



- Choose 'Asm File (.s)' template and under Name type '8051_Lab1' then press add.



- An editor window will turn up. Display on the desktop screen should be observed as follow.

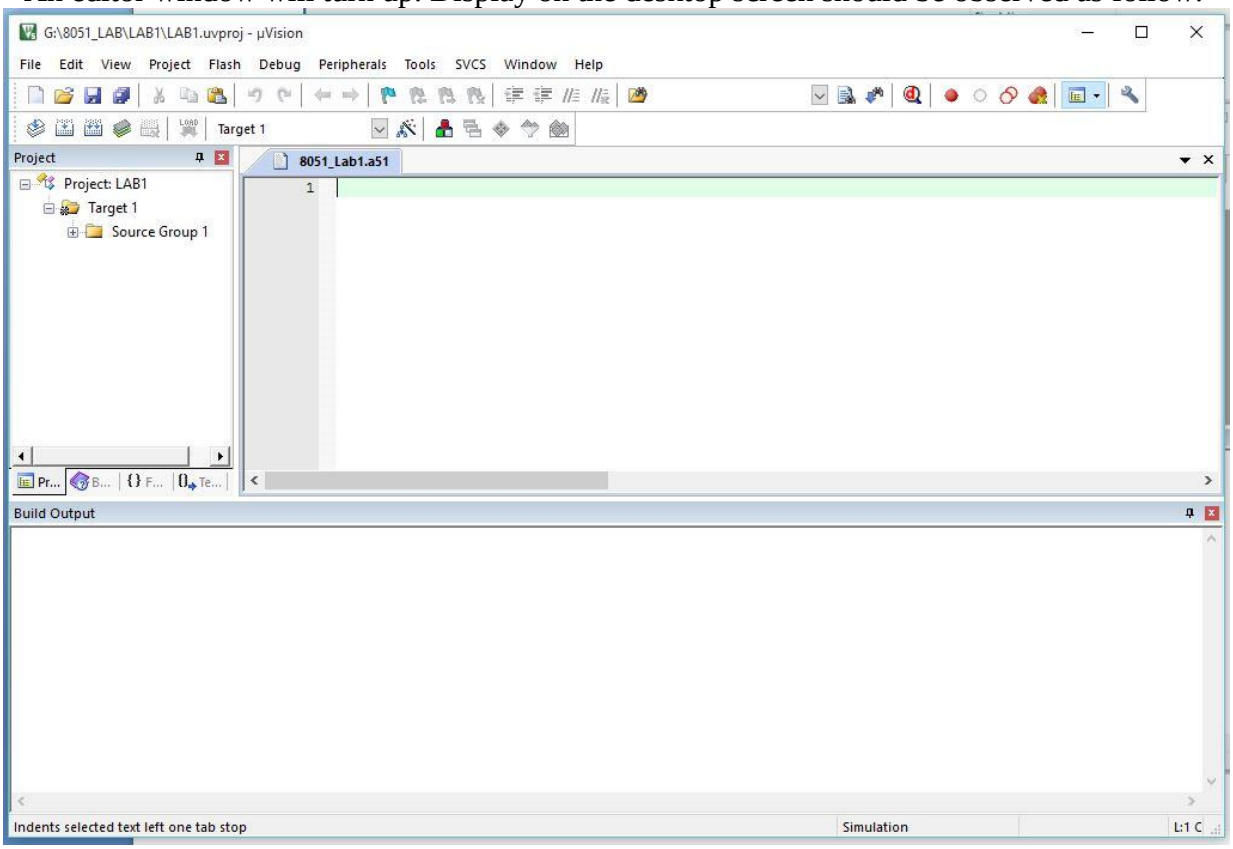
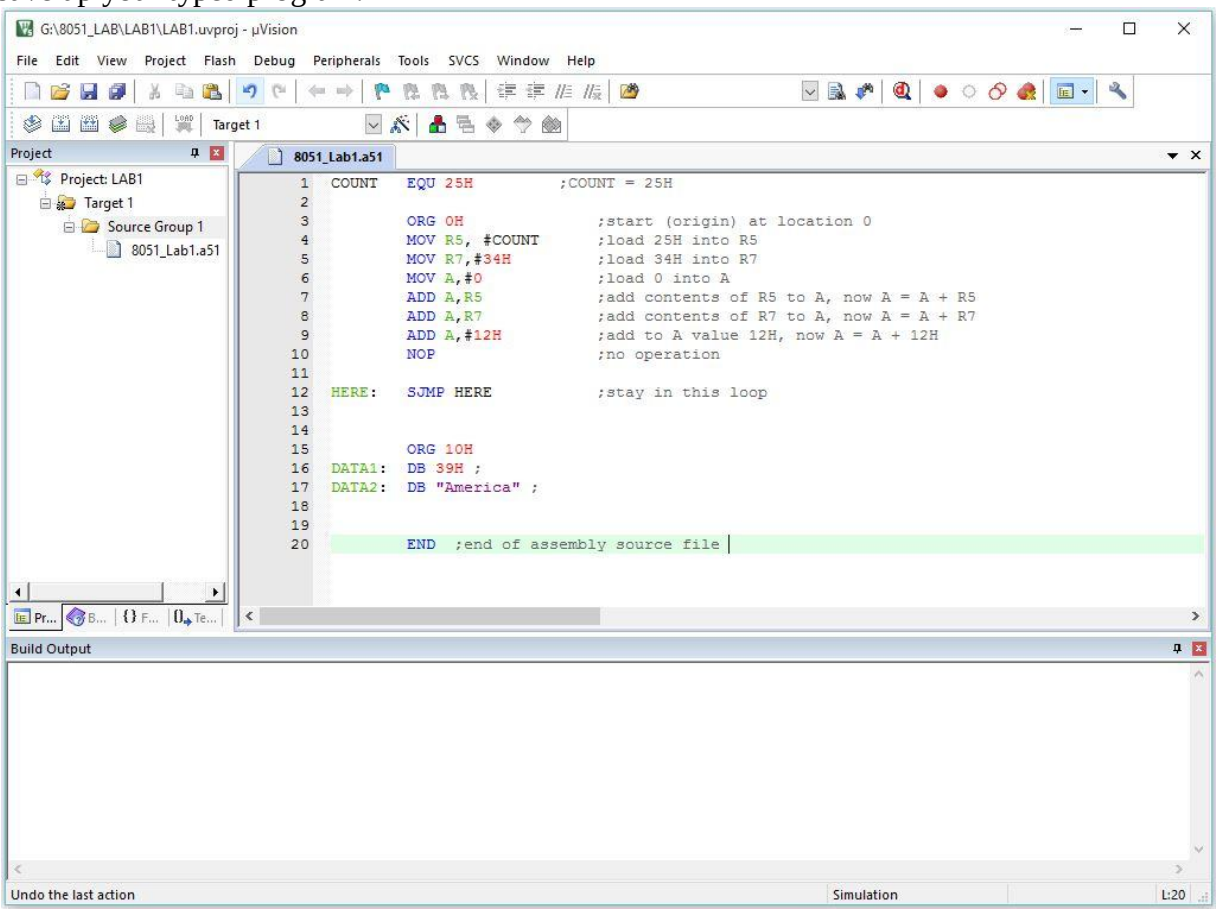
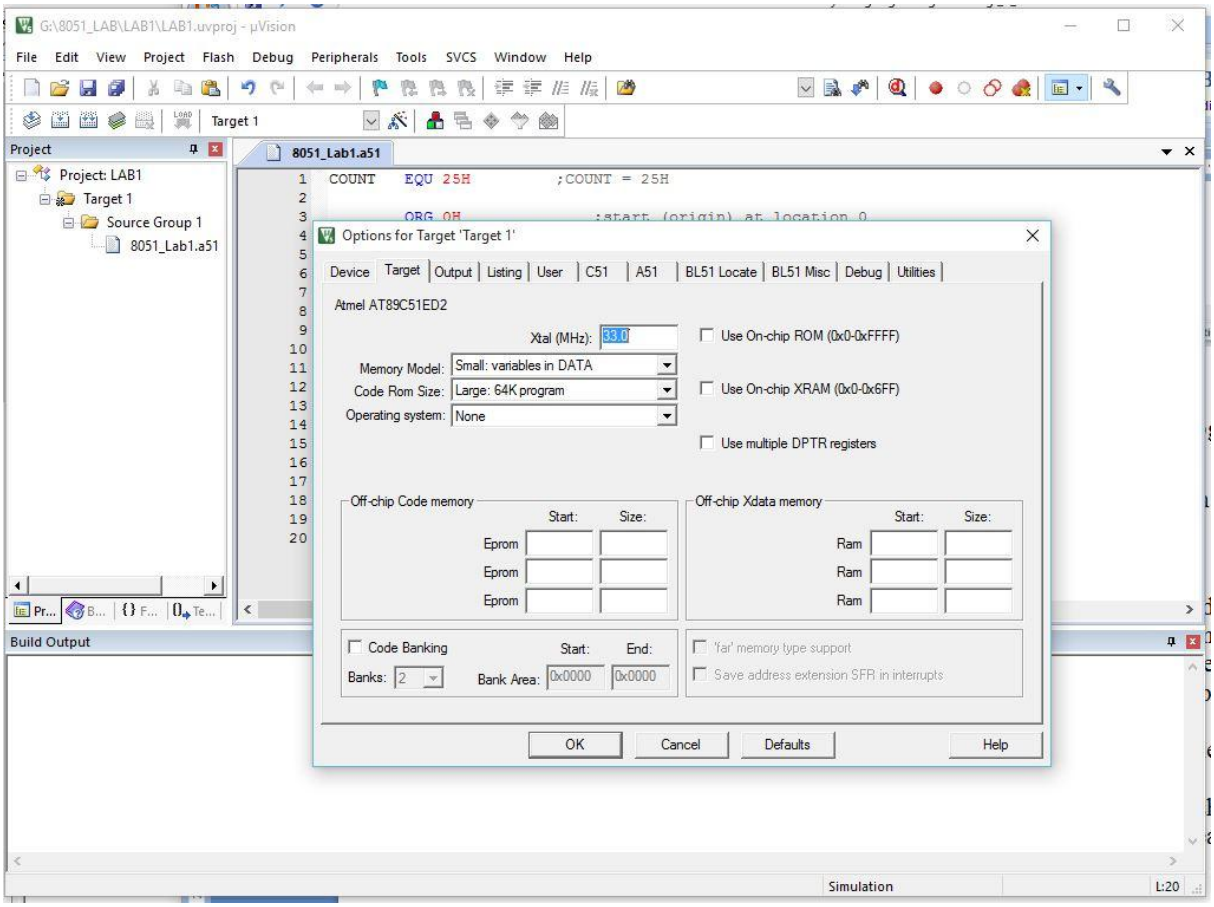


Figure B2

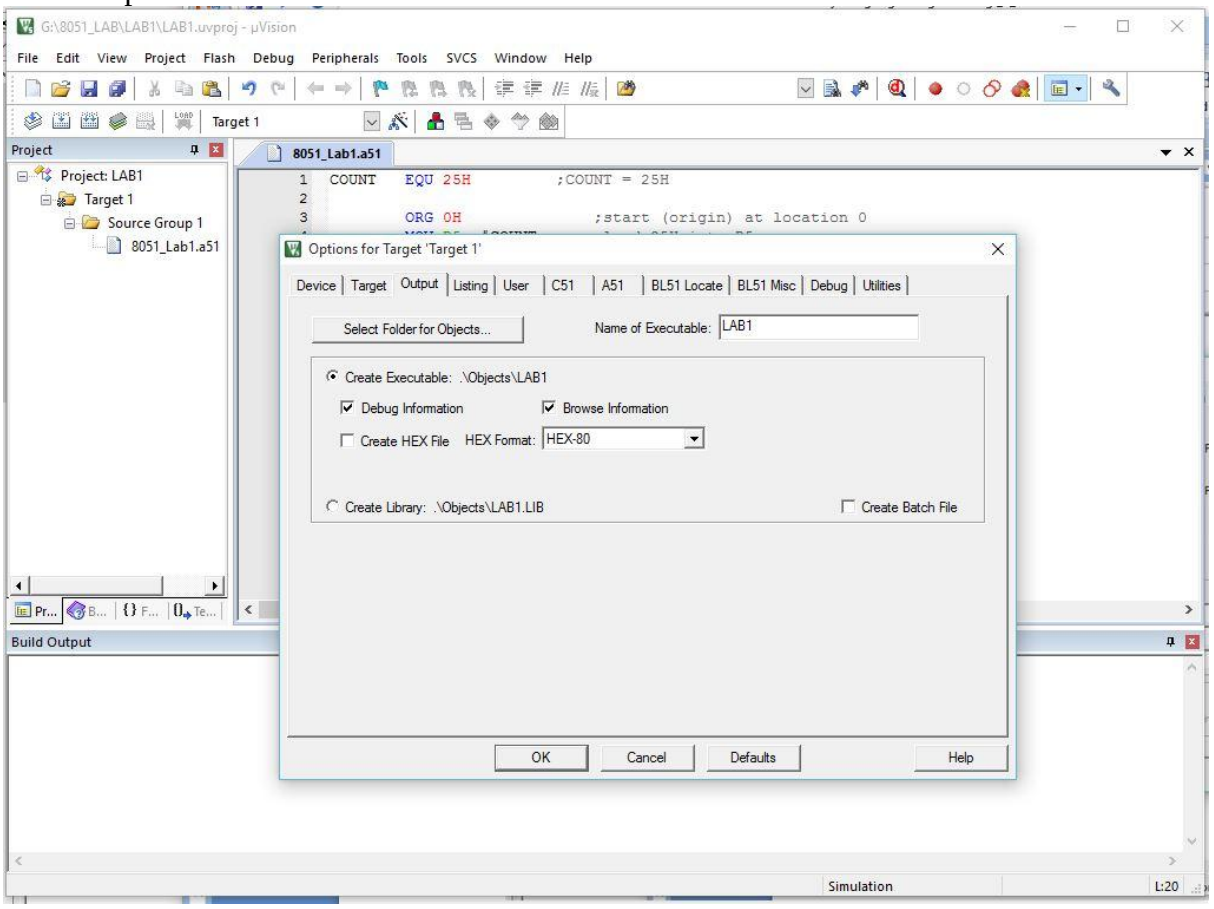
- The editor window will allow you to type in the assembly source program. Use “Save” command to save up your typed program.



- Before Assembling the code, we have to set the projects settings from "Options for Target 'Target 1' " in the “Project” pull-down menu



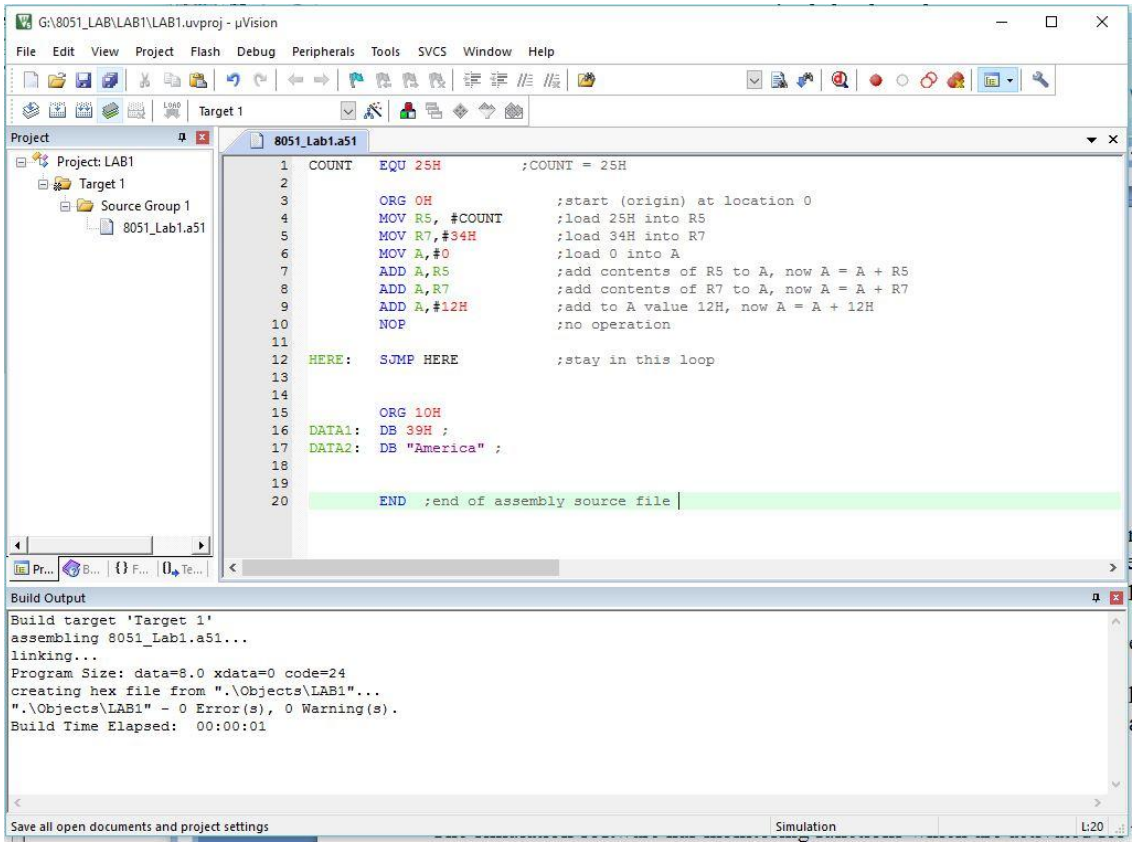
- Goto 'Output' tab



- Check 'Create Hex File' and press 'OK'

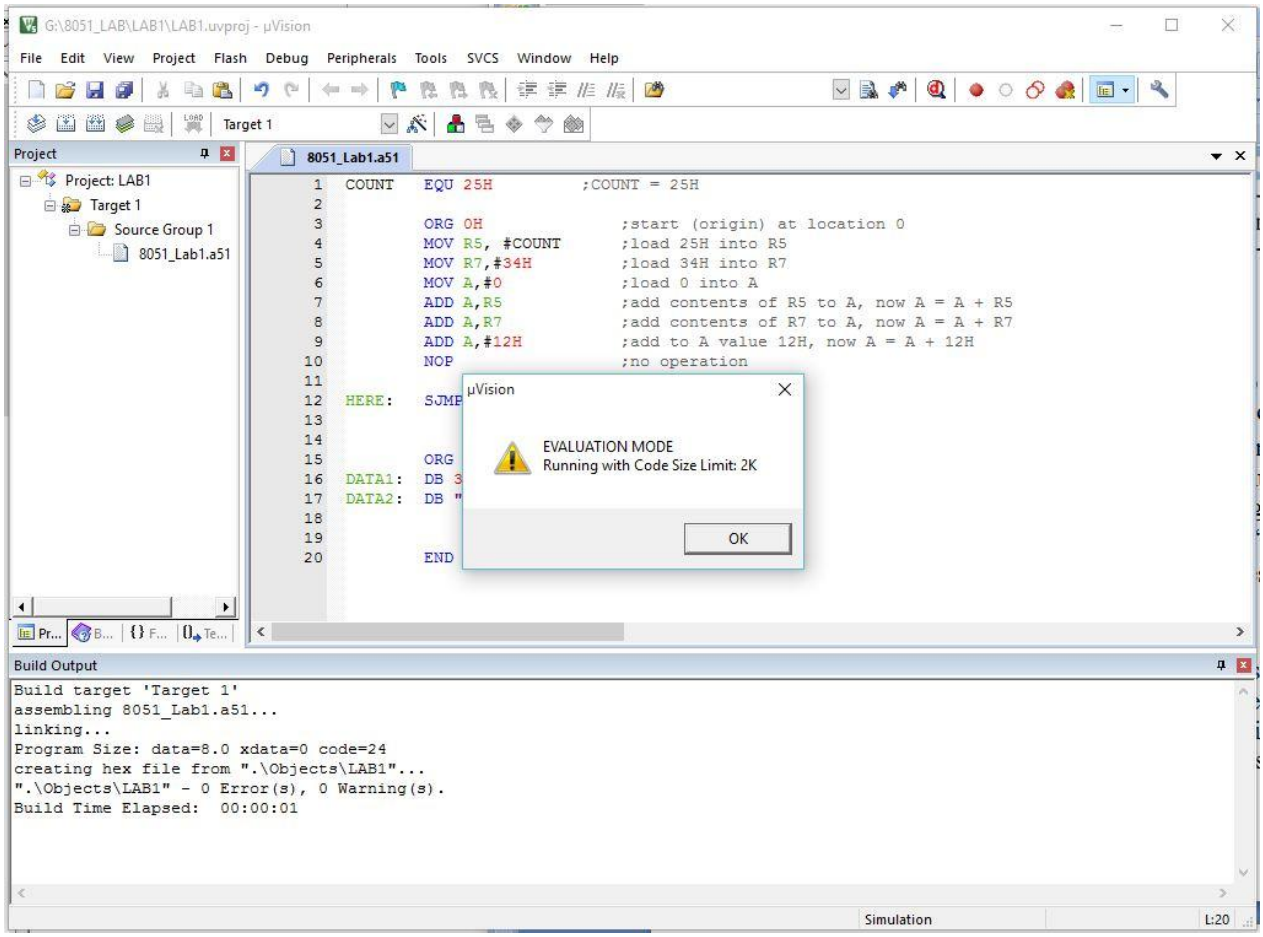
B.3 Assemble the source program

- With the source program in the editor window. Press 'F7' or use “Build Target” command in the “Project” pull-down menu to assembly the source program into machine executable instructions. This single command will produce all necessary files, included the List file and Hex file and others are used by "µVision". A window is then popped up at the bottom of the screen to show out the assembly result.
- If error or warning message was shown in the window, there should be incorrect or undefined statements being found in the source program. e.g. syntax errors or undefined variables. The assembly process could not be completed. You should correct the errors and repeat the assembly procedures again until no such message appears.



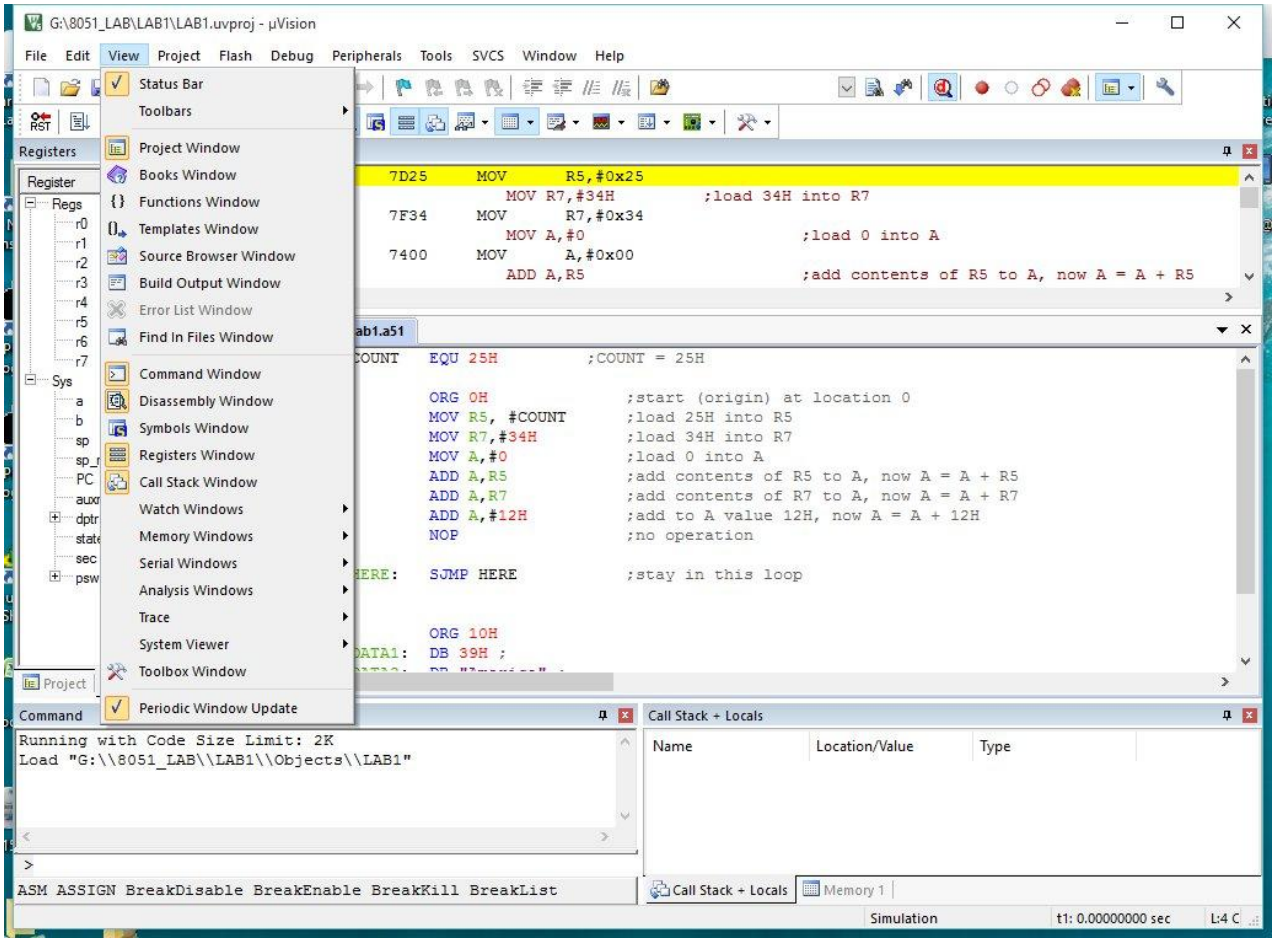
B.4 Monitoring functions of the simulation software

Start debugging by pressing F7, a message dialog box will appear indicating that this an evaluation version and code is limited to 2K bytes of memory, press OK.

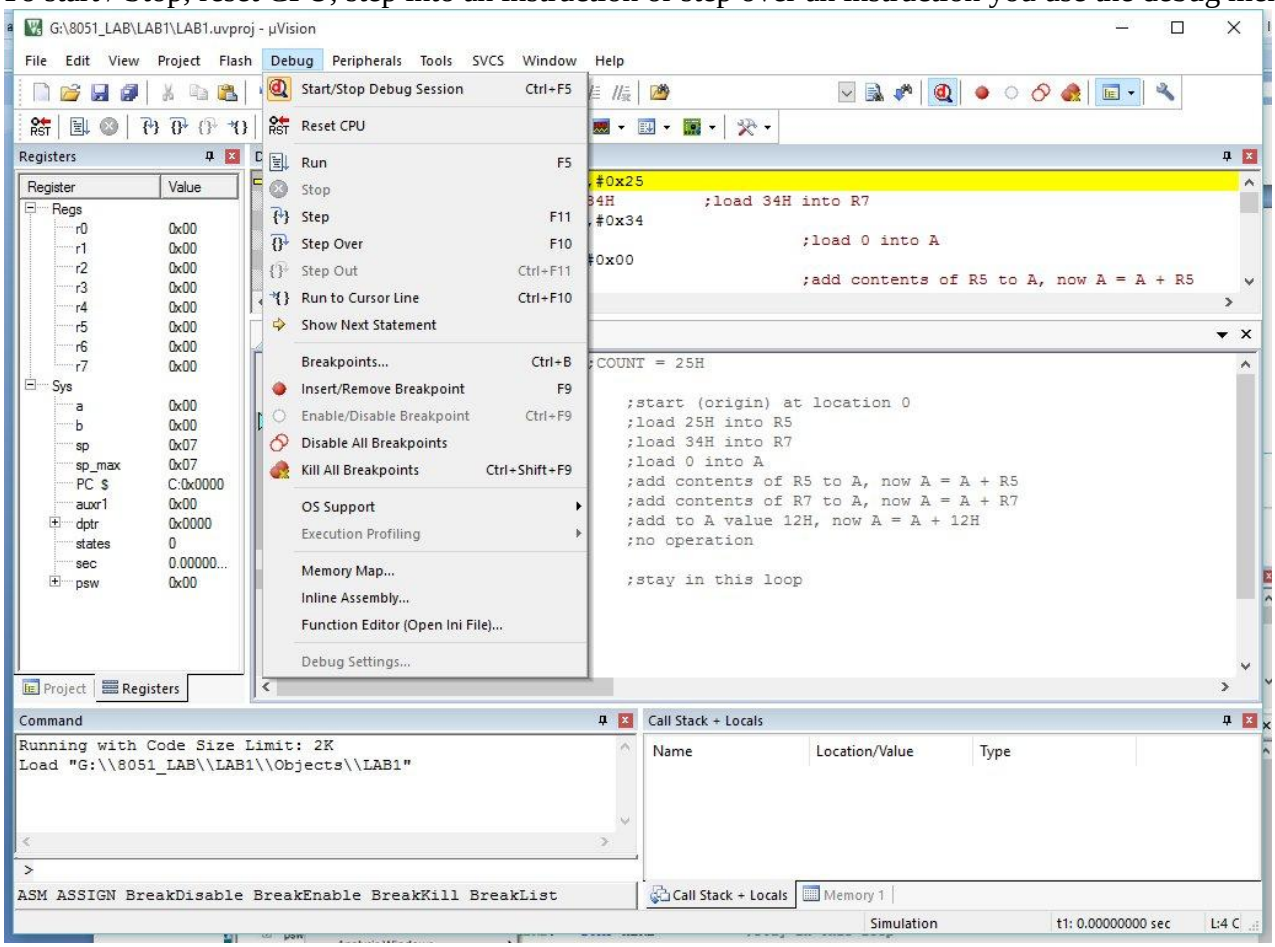


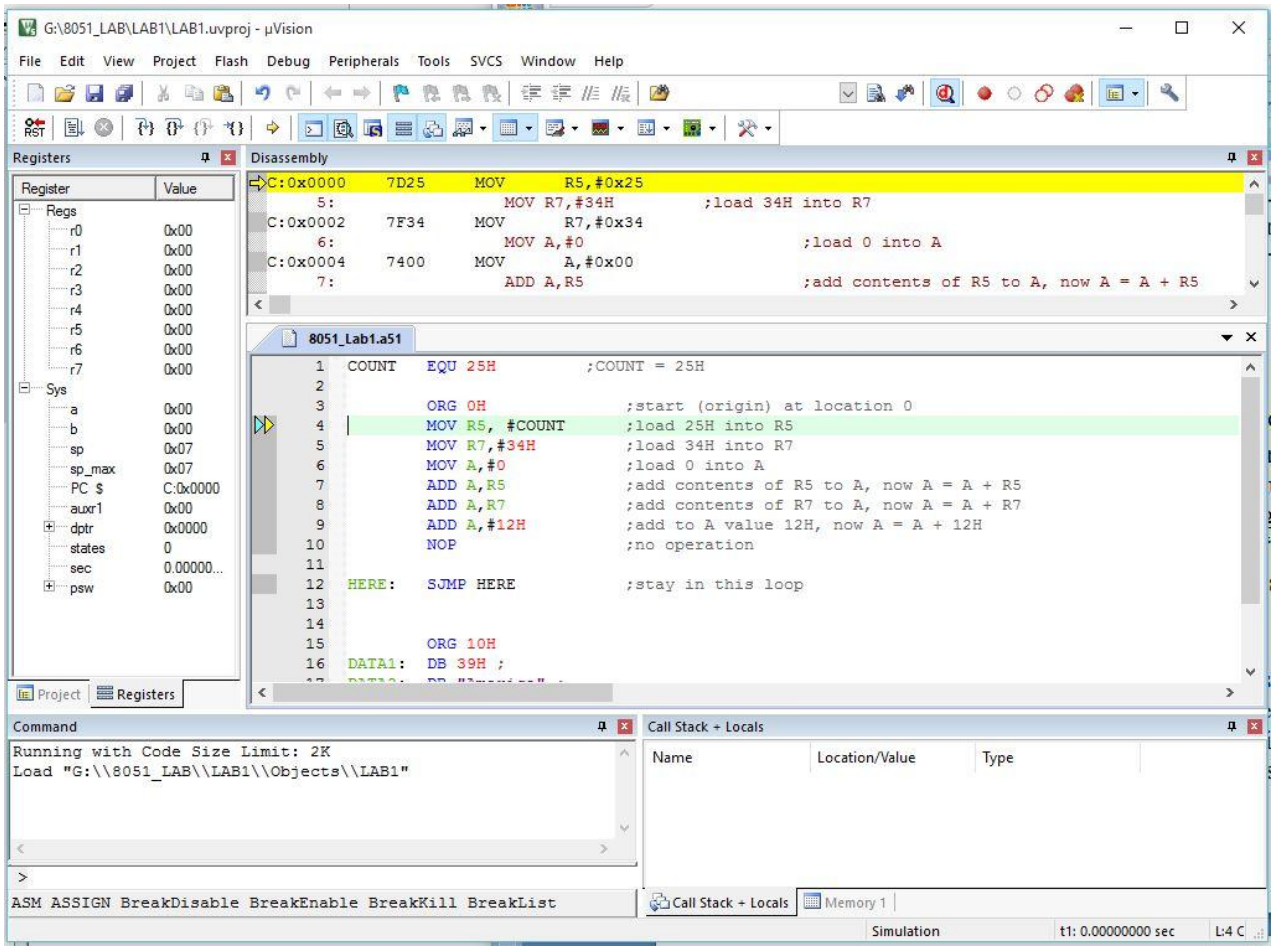
In debug mode, the software has monitoring functions which are activated for use to trace the code during debug mode.

1. Command Window
2. Disassembly window
3. Symbols Window
4. Registers Window
5. Call Stack Window
6. Watch Windows
7. Memory Windows
8. Serial Windows
9. Analysis Windows



To start / Stop, reset CPU, step into an instruction or step over an instruction you use the debug menu




G:\8051_LAB\LAB1\LAB1.uvproj - µVision
File Edit View Project Flash Debug Peripherals Tools SVCS Window Help
Registers
Register Value
Regs
r0 0x00
r1 0x00
r2 0x00
r3 0x00
r4 0x00
r5 0x00
r6 0x00
r7 0x00
Sys
a 0x00
b 0x00
sp 0x07
sp_max 0x07
PC $ C:0x0000
auxr1 0x00
dptr 0x0000
states 0
sec 0.00000...
psw 0x00
Project Registers
Disassembly
C:0x0000 7D25 MOV R5,#0x25
5: MOV R7,#34H ;load 34H into R7
C:0x0002 7F34 MOV R7,#0x34
6: MOV A,#0 ;load 0 into A
C:0x0004 7400 MOV A,#0x00
7: ADD A,R5 ;add contents of R5 to A, now A = A + R5
8051_Lab1.a51
1 COUNT EQU 25H ;COUNT = 25H
2
3 ORG 0H ;start (origin) at location 0
4 MOV R5, #COUNT ;load 25H into R5
5 MOV R7,#34H ;load 34H into R7
6 MOV A,#0 ;load 0 into A
7 ADD A,R5 ;add contents of R5 to A, now A = A + R5
8 ADD A,R7 ;add contents of R7 to A, now A = A + R7
9 ADD A,#12H ;add to A value 12H, now A = A + 12H
10 NOP ;no operation
11
12 HERE: SJMP HERE ;stay in this loop
13
14
15 ORG 10H
16 DATA1: DB 39H ;
Command
Running with Code Size Limit: 2K
Load "G:\\8051_LAB\\LAB1\\Objects\\LAB1"
ASM ASSIGN BreakDisable BreakEnable BreakKill BreakList
Call Stack + Locals
Name Location/Value Type
Call Stack + Locals Memory 1
Simulation t1: 0.00000000 sec L:4 C

C. Programming exercise

C.1 Familiarisation

(1) Open the 8051 simulation software in Part B. Create a new file with filename "Lab02_asm" in the editor window.

(2) Type in the assembly program in table A.2 into the editor window.

(3) Assemble the source program until no compilation error or warning message occurs.

(4) Issue suitable commands to show the following windows on screen

4.1 Disassembly Code Window
4.2 The Registers window

(5) Complete the following table by writing down the addresses and opcodes corresponding to the instructions inside the program.

Code Address	Op Codes				Mnemonics	Operands
					ORG	0H
0000					MOV	R5, #25H
					MOV	R7,#34H
					MOV	A,#0
					ADD	A,R5
					ADD	A,R7
					ADD	A,#12H
					NOP	
				HERE:	SJMP	HERE
					ORG	10H
				DATA1:	DB	39H
				DATA2:	DB	"America"
					END	

(6) Execute the program steps by steps until end. Note the change of values in the registers (Use the reset command to restart the program execution at the beginning)

(7) Record down the values in accumulator A, registers R5 & R7 at the start of the execution

Value of the Accumulator A = _____ R5 = _____ R7 =_____

(8) Record down all change values in Accumulator A when program is executed

Change of values in the Accumulator A = _____

C.2 Data Movement

C.2.1 Immediate Addressing

The source operand is a constant in immediate addressing mode. The immediate data must be preceded by the pound sign "#". This addressing mode is used to load data into any of the registers.

(1) Type in the following assembly program into the editor window.

Table C.2.1

line	Mnemonics	Operands
1.	ORG	0h
2.	MOV	A, #25h
3.	MOV	B, #18h
4.	MOV	R1, #0CDh
5.	MOV	R2, #0A1h
6.	MOV	R3, #36h
7.	MOV	DPTR, #4521h
8.	ADD	A, #5Bh
9.	ADD	A, #28h
10.	END	

(2) Assemble the source program and observe the values in the registers during program execution.

(3) Record the value of the Accumulator A at the end of execution. _____

C.2.2 Register Addressing

Register addressing mode involves the move of data between registers and use them to hold the data to be manipulated.

(1) Replace line 10 in the assembly program in table C.2.1 by the following 5 instructions.

Table C.2.2

line	Mnemonics	Operands
10.	MOV	A, R2
11.	MOV	R3, A
12.	ADD	A, R1
13.	MOV	A, R3
14.	MOV	R2, A
15.	END	

(2) Assemble the source program and observe the change of values in the registers during program execution.
(3) Record the value of the Accumulator A at the end of execution. _____

C.2.3 Direct Addressing

Direct addressing mode is used to transfer data to and from internal memory locations directly between the registers of the 8051.

(1) Replace instruction lines 10 to 14 of the assembly program in part C.2.2 by the following 5 instructions.

Table C.2.3

line	Mnemonics	Operands
10.	MOV	20h, R1
11.	ADD	A, 20h
12.	MOV	21h, A
13.	MOV	R3, 20h
14.	ADD	A, R3

(2) Assemble the source program and observe the change of values in the registers during program execution.

(3) Write down the contents of the following at the end of execution

Accumulator A = _____ R1 = _____ R3 = _____

Memory locations 20h = _____ 21h = _____

C.2.4 Register Indirect Addressing

In the indirect addressing mode, a register is used as a pointer to the data. The content of the register stored the address of the memory location to be accessed.

C.2.4.1 Transfer from internal RAM (8-bit memory address)

A register (8-bit length) can work as a pointer to store the RAM address to be accessed.
1) Use the 8051 simulation software, create and carry out simulated execution of the assembly program in table C.2.4.1
Table C.2.3

line	Mnemonics	Operands
1	ORG	0H
2	MOV	20H, #38H

3	MOV	21H, #30H
4	MOV	22H, #35H
5	MOV	23H, #31H
6	MOV	R0, #20H
7	MOV	R1, #30H
8	MOV	R2, #40H
9	MOV	A, @R0
10	MOV	@R1, A
11	INC	R0
12	INC	R1
13	MOV	A, @R0
14	MOV	@R1, A
15	INC	R0
16	INC	R1
17	MOV	A, @R0
18	MOV	@R1, A
19	INC	R0
20	INC	R1
21	XCH	A, @R0
22	MOV	A, @R0
23	MOV	@R1, A
24	END	

(2) Open suitable windows to observe the change of values in the registers and internal RAM during program execution.

(3) Write down the contents of the following registers at the end of execution.

Accumulator A = _____ R0 = _____ R1 = _____ R2 = _____

Internal Memory location	Content	Internal Memory location	Content
20H		30H	
21H		31H	
22H		32H	

C.2.4.2 Transfer from code memory (16-bit memory address)

A 16-bit Special Function Register (SFR), DPTR register, which is used as a 16-bit pointer to external connected RAM.

(1) Use the 8051 simulation software, create and carry out simulated execution of the assembly program in table C.2.4.2

line	Mnemonics	Operands
1	ORG	200H
2	DB	"Alex"
3	;	
4	ORG	0H
5	MOV	DPTR, #200H
6	CLR	A
7	MOVC	A, @A+DPTR
8	MOV	R3, A
9	MOV	R0, A
10	PUSH	0
11	INC	DPTR
12	CLR	A
13	MOVC	A, @A + DPTR
14	MOV	R1, A
15	INC	DPTR
16	CLR	A
17	MOVC	A, @A + DPTR
18	XCH	A , R3
19	POP	3
20	END	

(2) Open suitable windows to observe the change of values in the registers and internal RAM during program execution.

(3) Write down the contents of the following registers at the end of execution.

Register	Content	Internal Memory location	Content
R0		200H	
R1		201H	
R2		202H	
R3			
SP			
DPTR			