B.M.S. College of Engineering, Bangalore – 19

(Autonomous & Affiliated College under VTU)

Department of Electronics & Communication Engineering

Course Title	ARM Processor & Programming LAB				
Course Code	21ES4CCAPP	Credits	4	L-T-P	3:0:1

Course Objectives:

The student should be made to:

- **Learn the working of ARM processor**
- **Understand the Building Blocks of Embedded Systems**
- **Learn the concept of memory map and memory interface**
- **Write programs to interface memory, I / Os with processor**
- **Study the interrupt performance.**

Lab requisites:

Software and hardware tools used for ARM PROGRAMMING:

- **Keil µVision4 IDE (ASSEMBLY PROGRAMS – SIMULATION - PART - A)**
- **ARM7TDMI - LPC 2148 EVALUATION BOARD FOR INTERFACING EXPERIMENTS (PART - B)**

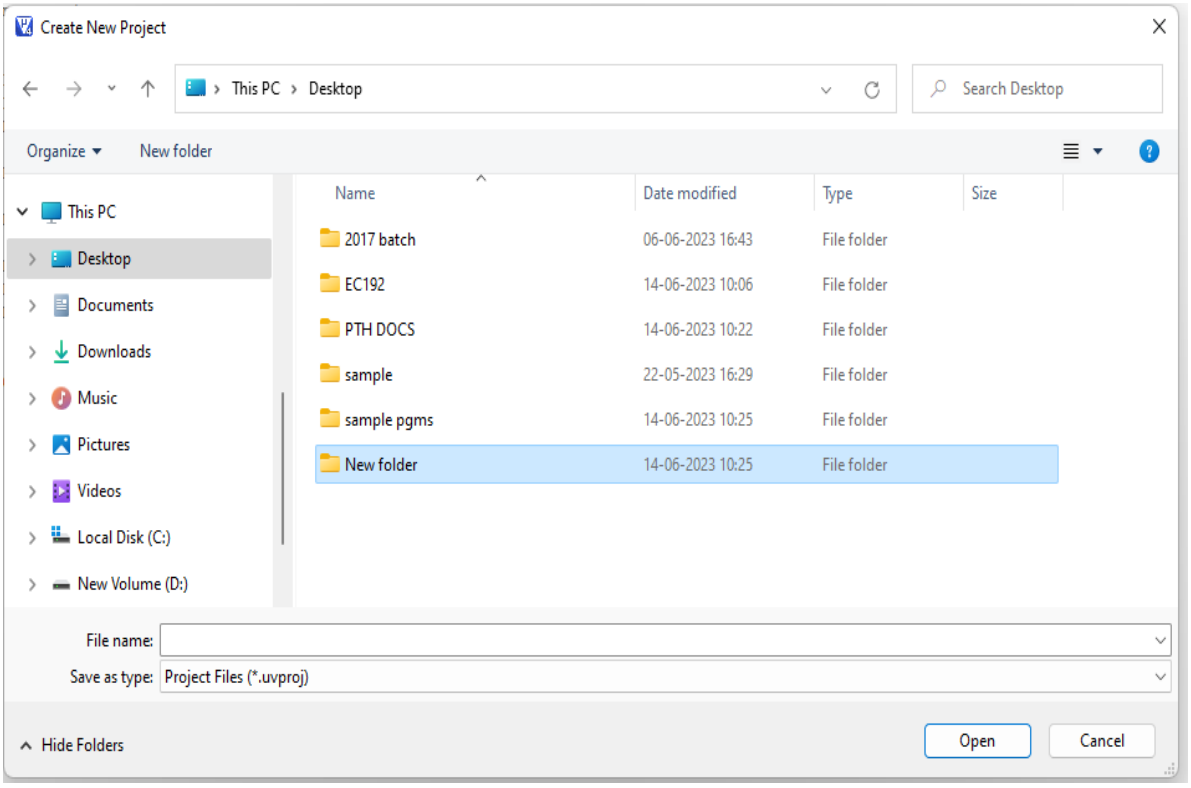
PART-A EXPERIMENTS USING Keil µVision4 IDE (SIMULATION)

1. Add a series of 16-bit numbers stored in sequential location in the memory (called Table) and store the result in memory.
2. Add two 64-bit numbers and store the result in a memory location.
3. Sum of first 10 integer numbers.
4. Multiply two 16-bit binary numbers.
5. Find the factorial of a given number.
6. Divide an 8-bit variable into two 4-bit nibbles and store one nibble in each byte of a 16-bit variable. Store the disassembled byte in memory location (pointed by result).
7. Compare 2 values stored in memory location and store the higher value in a memory location (pointed by result).
8. Find the largest in a series of numbers stored in memory.

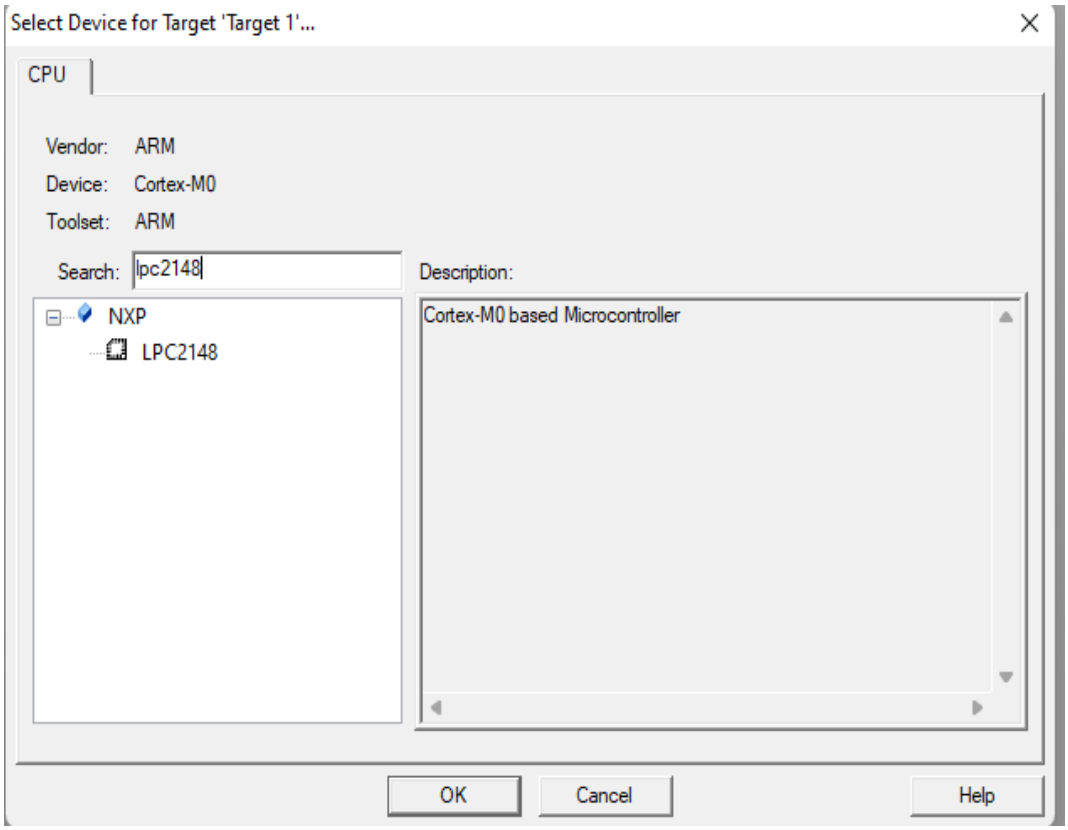
PART A - ASSEMBLY PROGRAMS EXECUTION STEPS - Keil µVision4 IDE

- Open **Keil µVision4 IDE** software by double clicking on **Keil µVision4** icon:

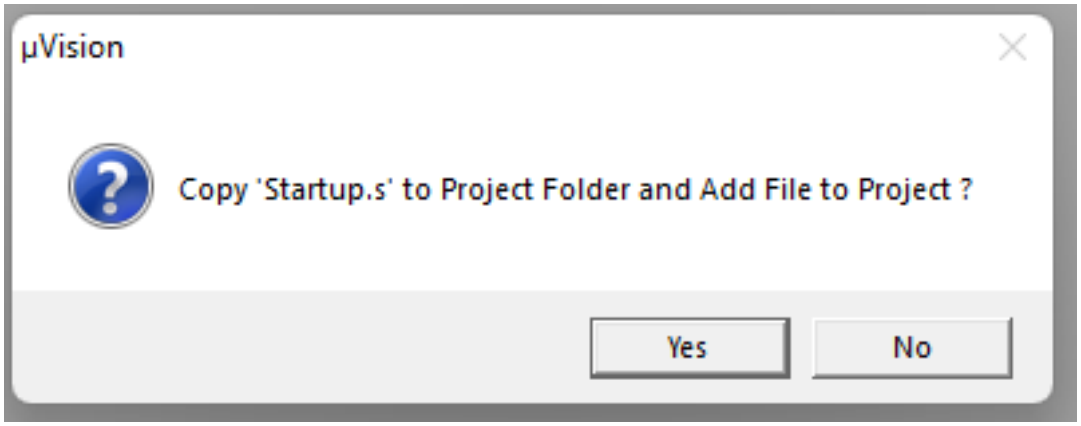
- Go to "**Project**", select "**New µvision Project**" and save the project:



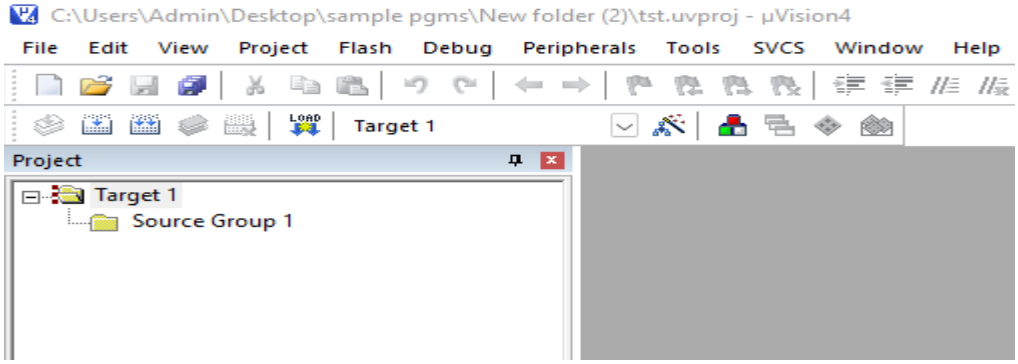
❖ Once the project is created, select the **target device** - select **LPC2148**, click on OK



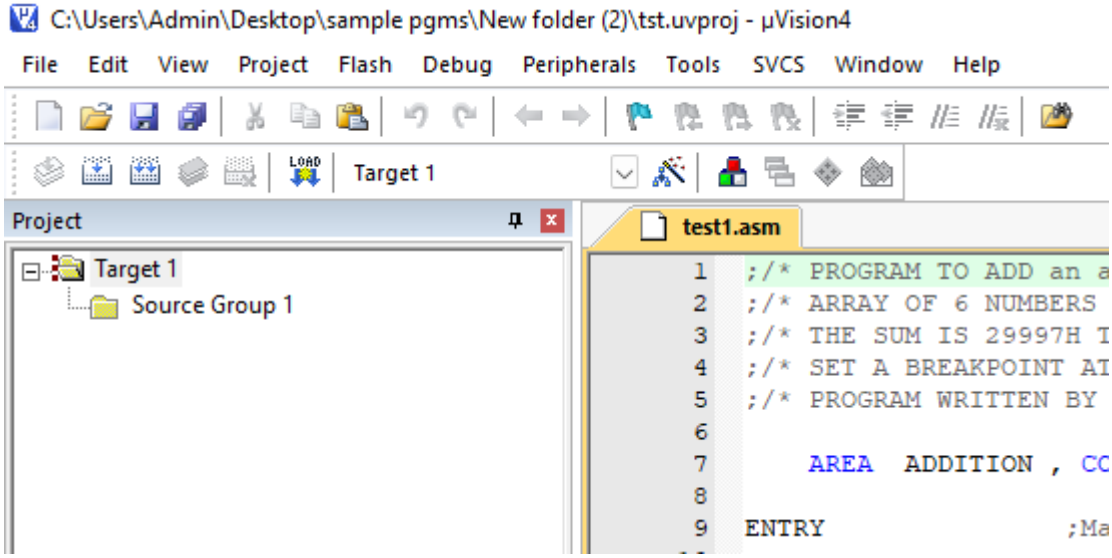
❖ A dialog box with **"copy startup. s to project folder and add file to project?"** will pop up, select **NO**.



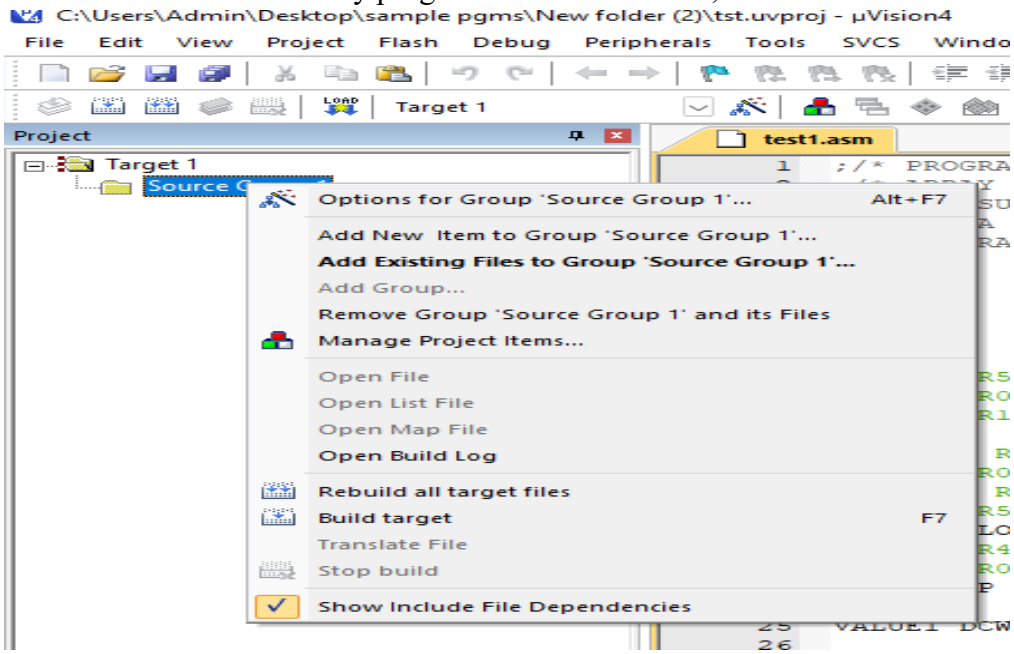
❖ Thus, the target is created along with source group1 as shown below:

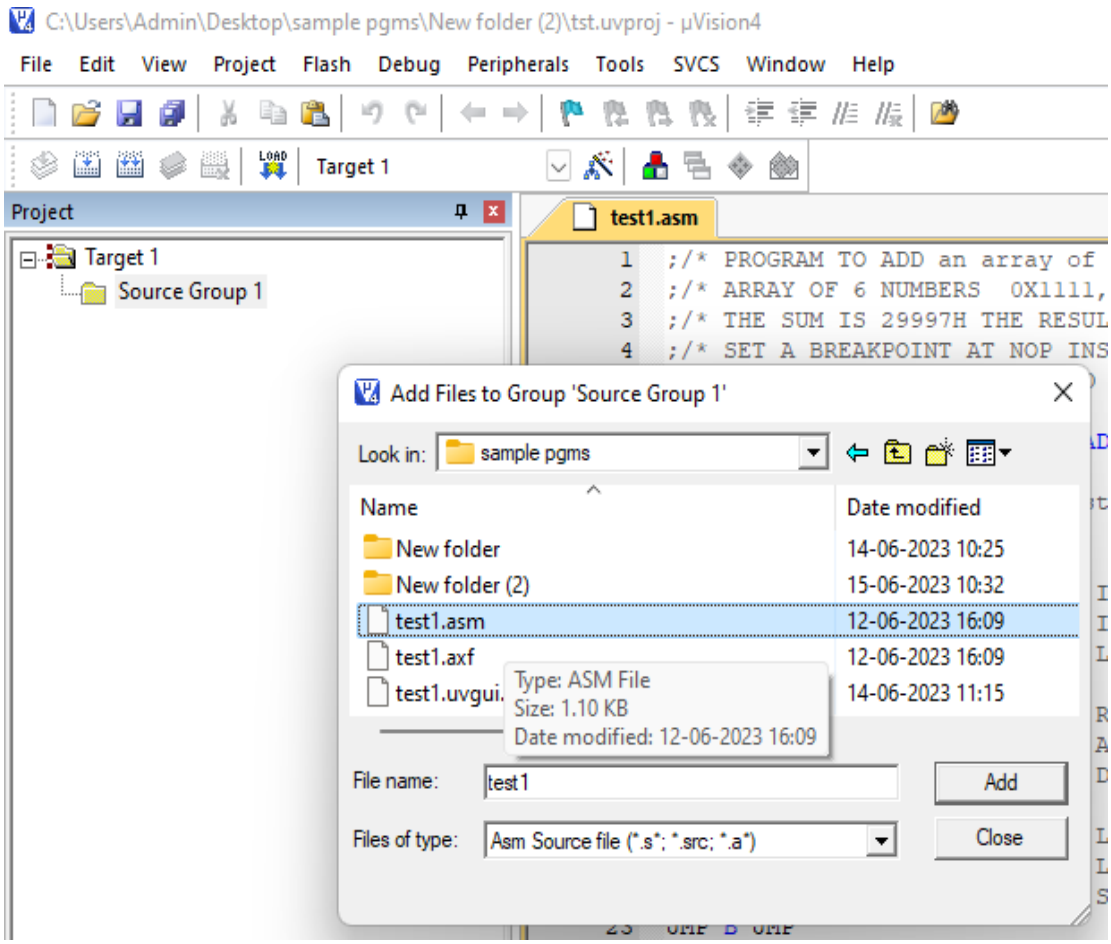


❖ Open the text editor. Write the program and save the program with the extension **.asm**

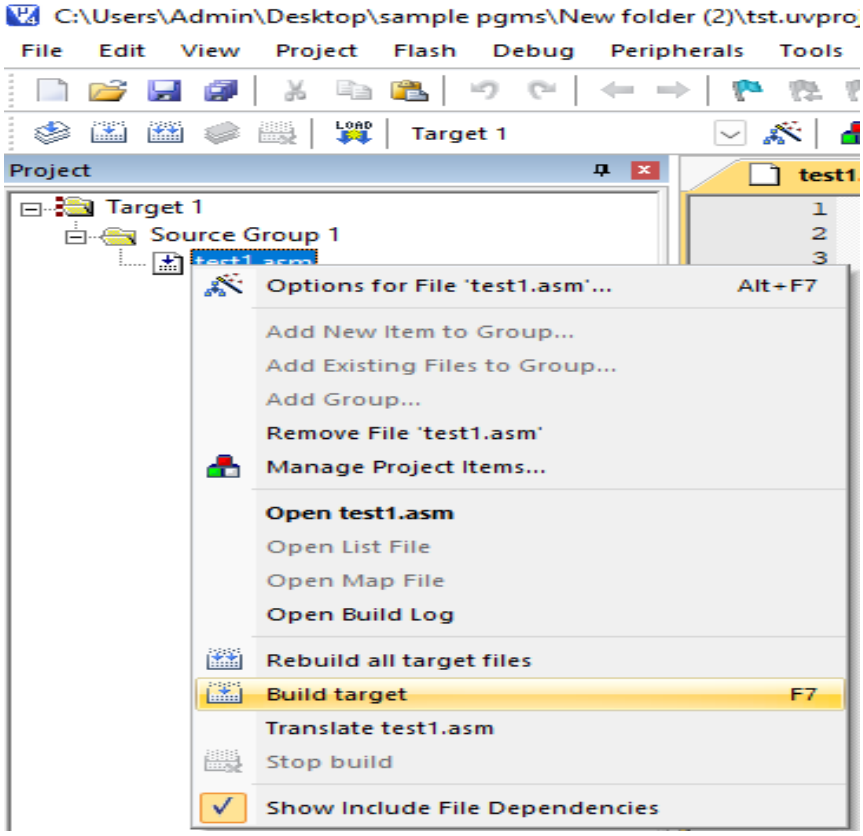


❖ Once the program is saved, the program file must be added to the **target.**
Right click on the "Source Group 1", click on **Add existing files to group "source group 1"** and then select the assembly program and click on Add, Close.

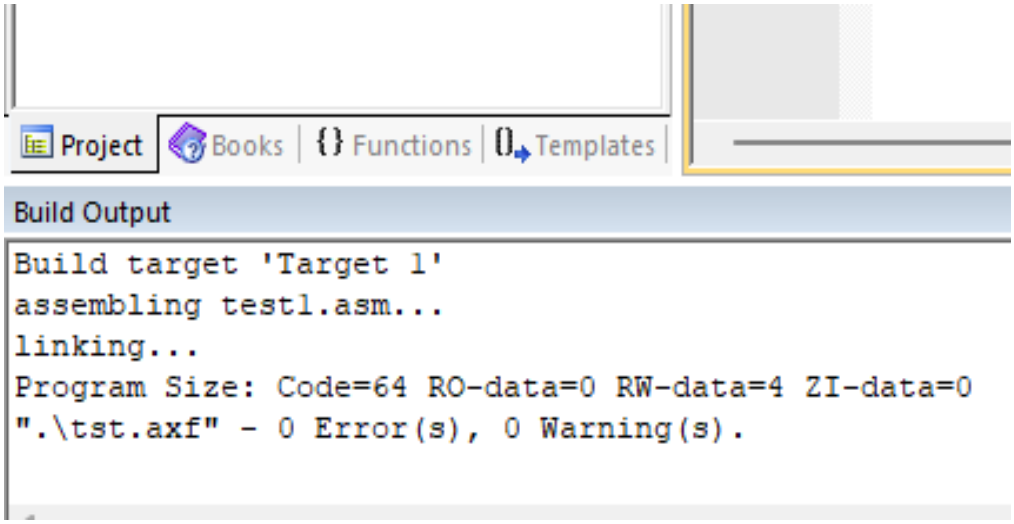




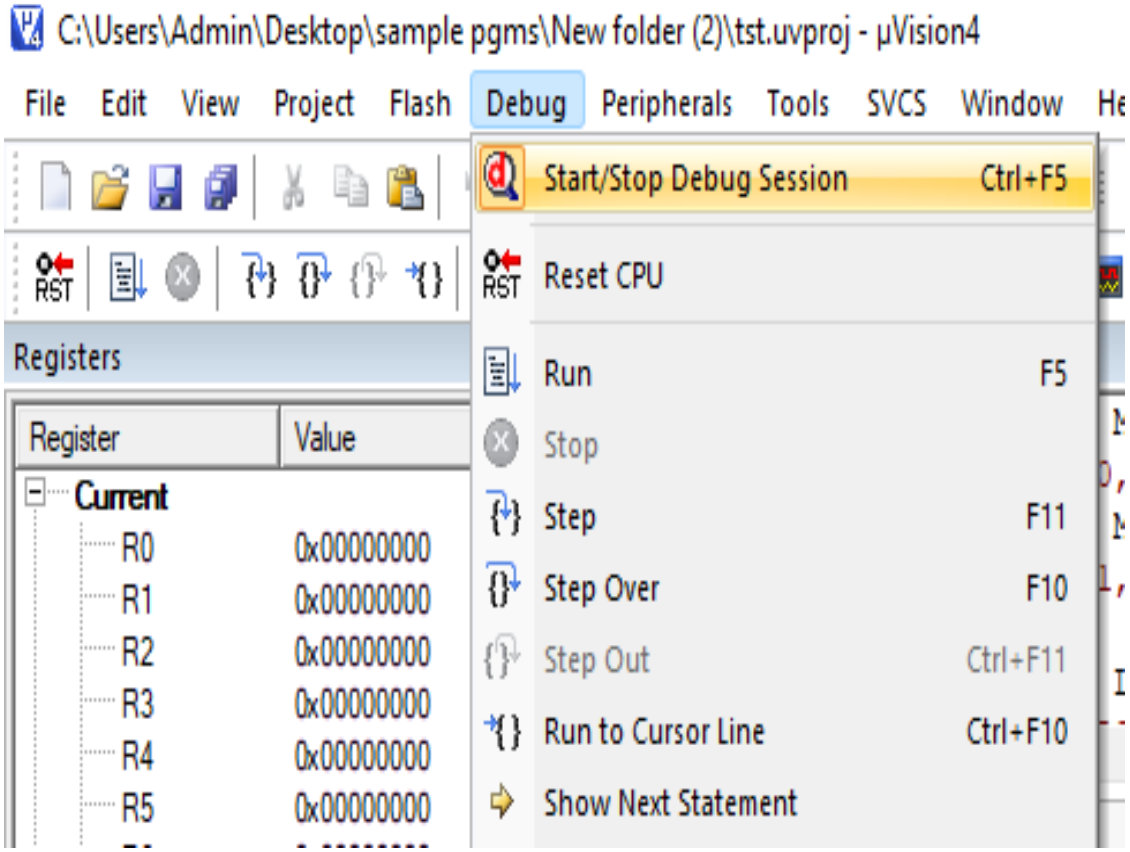
❖ Once the program is added to the source group, right click, select Build target or press F7



❖ Once the target is built, it displays the information with any errors or zero errors.



❖ If no errors, we can proceed for debug, select start /stop debugging option.

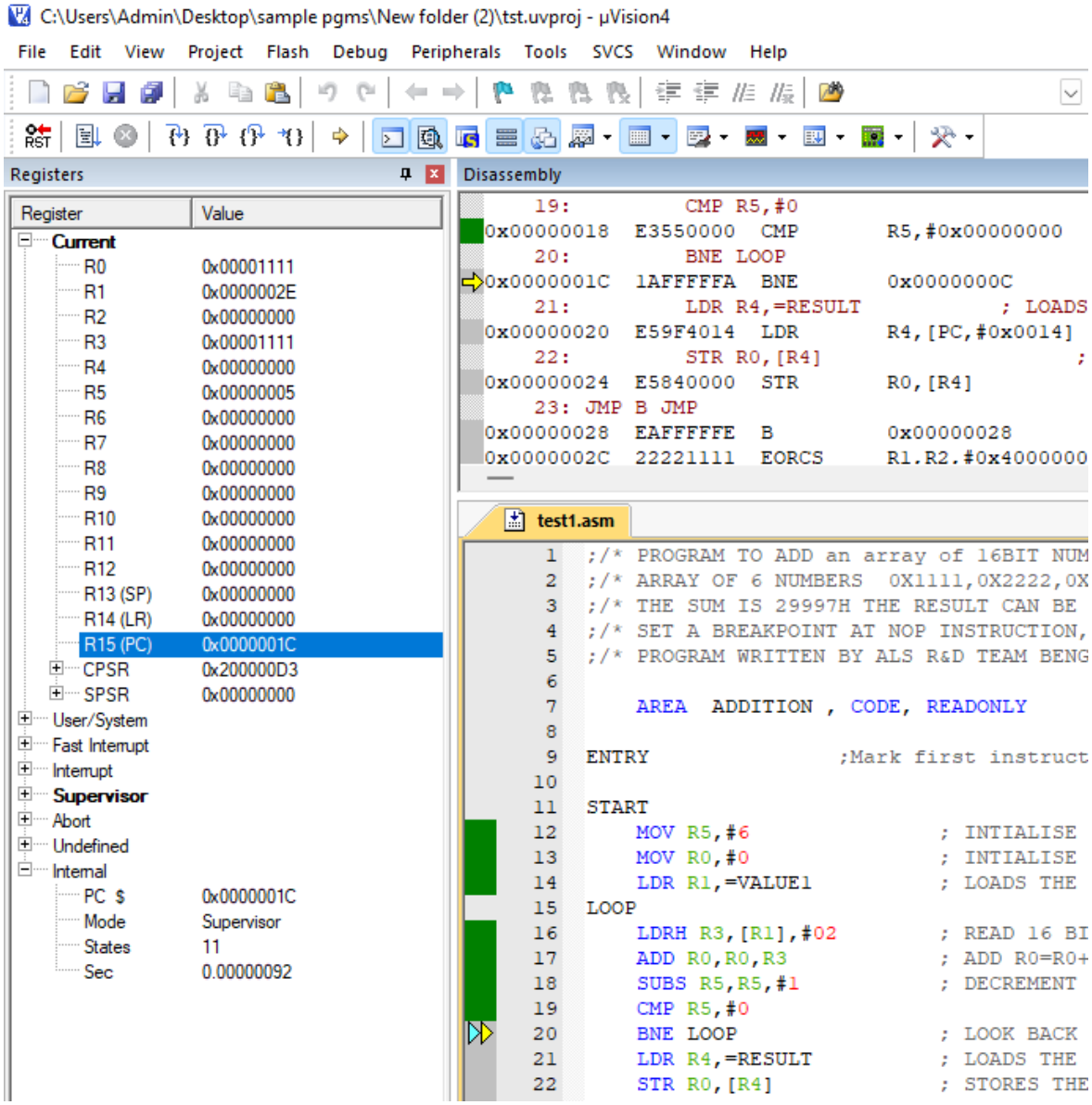


❖ Then select Run option from the debug menu or can use the key F5.

If F5 key - continuous run

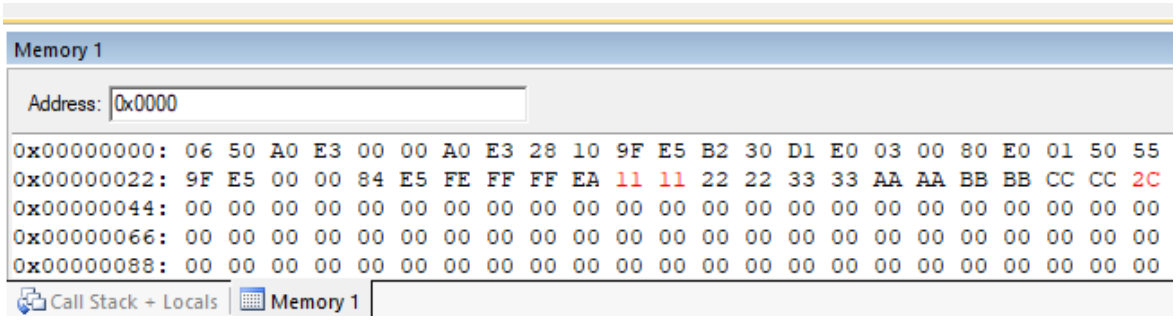
If F11key - step one line

If F10key – step over line

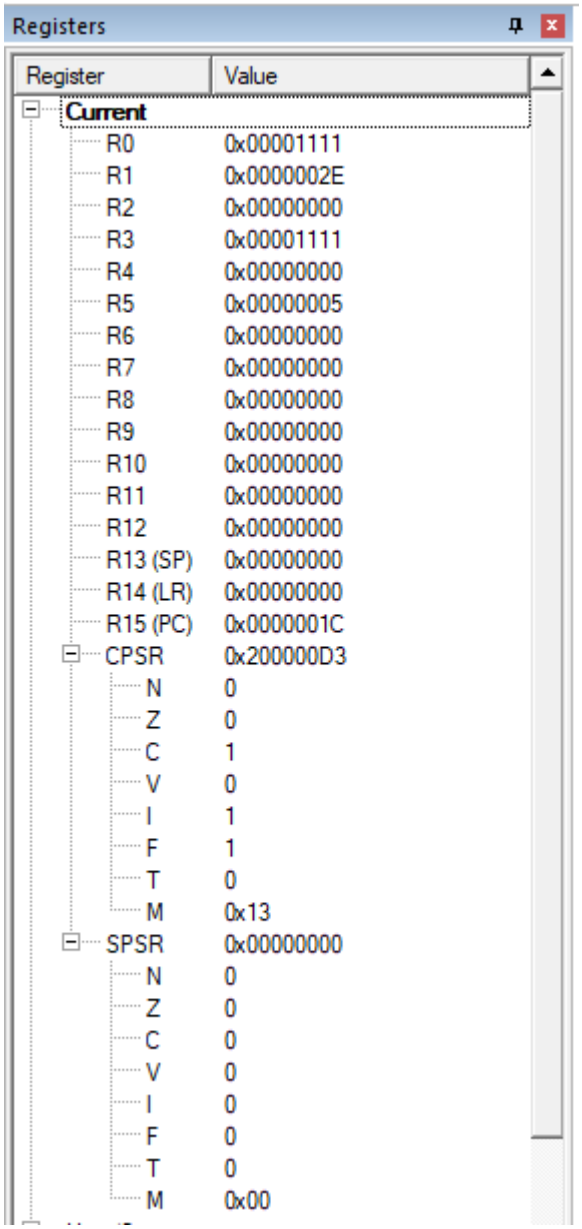


❖ The results obtained will be verified in the respective registers and in memory window as shown below:

Memory window:



Registers window:



****

Instruction set (for reference)

Mnemonic	Instruction	Action	See Section:
ADC	Add with carry	Rd := Rn + Op2 + Carry	4.5
ADD	Add	Rd := Rn + Op2	4.5
AND	AND	Rd := Rn AND Op2	4.5
B	Branch	R15 := address	4.4
BIC	Bit Clear	Rd := Rn AND NOT Op2	4.5
BL	Branch with Link	R14 := R15, R15 := address	4.4
BX	Branch and Exchange	R15 := Rn, T bit := Rn[0]	4.3
CDP	Coprocesor Data Processing	(Coprocessor-specific)	4.14
CMN	Compare Negative	CPSR flags := Rn + Op2	4.5
CMP	Compare	CPSR flags := Rn - Op2	4.5
EOR	Exclusive OR	Rd := (Rn AND NOT Op2) OR (op2 AND NOT Rn)	4.5
LDC	Load coprocessor from memory	Coprocessor load	4.15
LDM	Load multiple registers	Stack manipulation (Pop)	4.11
LDR	Load register from memory	Rd := (address)	4.9, 4.10
MCR	Move CPU register to coprocessor register	cRn := rRn {<op>cRm}	4.16
MLA	Multiply Accumulate	Rd := (Rm * Rs) + Rn	4.7, 4.8
MOV	Move register or constant	Rd : = Op2	4.5
MRC	Move from coprocessor register to CPU register	Rn := cRn {<op>cRm}	4.16
MRS	Move PSR status/flags to register	Rn := PSR	4.6
MSR	Move register to PSR status/flags	PSR := Rm	4.6
MUL	Multiply	Rd := Rm * Rs	4.7, 4.8
MVN	Move negative register	Rd := 0xFFFFFFFF EOR Op2	4.5
ORR	OR	Rd := Rn OR Op2	4.5

1. Write an ALP to Add a series of 16-bit numbers stored in sequential location in the memory (called Table) and store the result in memory.

```
;/*ARRAY OF 6 NUMBERS 0X1111,0X2222,0X3333,0XAAAA,0XBBBB,0XCCCC*/
;/* THE SUM IS 29997H THE RESULT CAN BE VIEWED IN LOCATION 0X40000000 &
ALSO IN R0  */

        AREA ADDITION, CODE, READONLY

ENTRY                                   ;Mark first instruction to execute

START
        MOV R5,#6                       ; INTIALISE COUNTER TO 6(i.e. N=6)
        MOV R0,#0                       ; INTIALISE SUM TO ZERO
        LDR R1,=VALUE1                  ; LOADS THE ADDRESS OF FIRST VALUE
LOOP
        LDR R2,[R1],#2                   ; WORD ALIGN T0 ARRAY ELEMENT
        LDR R3,MASK                     ; MASK TO GET 16 BIT
        AND R2,R2,R3                    ; MASK MSB
        ADD R0,R0,R2                     ; ADD THE ELEMENTS
        SUBS R5,R5,#1                   ; DECREMENT COUNTER
        CMP R5,#0
        BNE LOOP                        ;  LOOK BACK TILL ARRAY ENDS
        LDR R4,=RESULT                  ; LOADS THE ADDRESS OF RESULT
        STR R0,[R4]                     ; STORES THE RESULT IN R1
        NOP
        NOP
        NOP
here B here

MASK DCD 0X0000FFFF                             ;  MASK MSB

VALUE1 DCW      0X1111,0X2222,0X3333,0XAAAA,0XBBBB,0XCCCC   ; ARRAY OF
16 BIT NUMBERS(N=6)

        AREA DATA2,DATA,READWRITE ; TO STORE RESULT IN GIVEN ADDRESS

RESULT DCD 0X0

        END                                     ; Mark end of file
```

2. Write an ALP to Add two 64-bit numbers and store the result in a memory location.

```
;/*     VALUE1      0X1234E640 0X43210010 (R0,R1)*/
;/*      VALUE2     0X12348900 0X43212102 (R2,R3)*/
;/*     RESULT                0X24696F40 0X86422112 (R5,R4)*/

       AREA  ADDITION , CODE, READONLY

ENTRY                                  ;Mark first instruction to execute

START

       LDR R0,=0X1234E640      ;LOAD THE FIRST VALUE IN R0,R1
       LDR R1,=0X43210010
       LDR R2,=0X12348900      ;LOAD THE SECOND VALUE IN R2,R3
       LDR R3,=0X43212102
       ADDS R4,R1,R3           ;RESULT IS STORED IN R4,R5
       ADC R5,R0,R2

       NOP
       NOP
       NOP

       END                             ;Mark end of file
```

---

3. Write an ALP to find sum of first 10 integer numbers.

```
            AREA SUM1, CODE, READONLY
START
            MOV R1, #10;
            MOV R2, #00;
LOOP
          ADDS R2, R2, R1;
          SUBS R1, R1, #01;
          BNE LOOP;
L         B L
          NOP
          END
```

4. Write an ALP to Multiply two 16-bit binary numbers.

```
;/*     VALUE1:     1900H (6400)          (IN R1)      */
;/*     VALUE2:     0C80H (3200)          (IN R2)      */
;/*     RESULT:     1388000H(20480000)(IN R3)      */

AREA multiply, CODE, READONLY

ENTRY                          ;Mark first instruction to execute

START

        MOV r1,#6400           ; STORE FIRST NUMBER IN R0
        MOV r2,#3200           ; STORE SECOND NUMBER IN R1
        MUL r3,r1,r2           ; MULTIPLICATION

        NOP
        NOP

        END                    ;Mark end of file
```

5. Find the factorial of a given number

```
        AREA FACTORIAL, CODE, READONLY
ENTRY                                   ;Mark first instruction to execute
START
        MOV r0, #7                      ; STORE FACTORIAL NUMBER IN R0
        MOV r1, r0                      ; MOVE THE SAME NUMBER IN R1 FACT
FACT SUBS r1, r1, #1                    ; SUBTRACTION
        CMP r1, #1                      ; COMPARISON
STOP    BEQ STOP
        MUL r3, r0, r1                  ; MULTIPLICATION
        MOV r0, r3                      ; Result
        BNE FACT                        ; BRANCH TO THE LOOP IF NOT EQUAL STOP
        NOP
        NOP
        NOP
        END                             ;Mark end of file
```

6. Divide an 8-bit variable into two 4-bit nibbles and store one nibble in each byte of a 16-bit variable. Store the disassembled byte in memory location (pointed by result).

```
        TTL     splitbyte
        AREA    Program, CODE, READONLY
        ENTRY

Main
        LDR     R1, Value               ;Load the value to be disassembled
        LDR     R2, Mask                ;Load the bitmask
        MOV     R3, R1, LSR#0x4         ;Copy just the high order nibble into R3
        MOV     R3, R3, LSL#0x8         ;now left shift it one byte
        AND     R1, R1, R2              ;AND the original number with the bitmask
        ADD     R1, R1, R3              ;Add the result of that to
                                        ;what we moved into R3
        STR     R1, Result              ;Store the result
        SWI     &11

Value   DCB     &FB                     ;Value to be shifted
        ALIGN                           ;keep the memory boundaries
Mask    DCW     &000F                   ;bitmask = %0000000000001111
        ALIGN
Result  DCD     0                       ;Space to store result
        END
```

7. Compare 2 values stored in memory location and store the higher value in a memory location (pointed by result).

```
        TTL     comparenum
        AREA    Program, CODE, READONLY
        ENTRY

Main
        LDR     R1, Value1              ;Load the first value to be compared
        LDR     R2, Value2              ;Load the second value to be compared
        CMP     R1, R2                  ;Compare them
        BHI     Done                    ;if R1 contains the highest
        MOV     R1, R2                  ;otherwise overwrite R1
Done
        STR     R1, Result              ;Store the result
        SWI     &11

Value1  DCD     &FEDCA987               ;Value to be compared
Value2  DCD     &12345678               ;Value to be compared
Result  DCD     0                       ;Space to store result
        END
```

8. Find the largest in a series of numbers stored in memory.

```
;/*ARRAY OF 7 NUMBERS 0X44444444
,0X22222222,0X11111111,0X33333333,0XAAAAAAAA*/
;/*0X88888888 ,0X99999999
;/* RESULT CAN BE VIEWED IN LOCATION 0X40000000 & ALSO IN R2
*/

        AREA  LARGEST , CODE, READONLY

ENTRY                           ;Mark first instruction to execute

START
        MOV R5,#6               ; INTIALISE COUNTER TO 6(i.e. N=7)
        LDR R1,=VALUE1          ; LOADS THE ADDRESS OF FIRST VALUE
        LDR R2,[R1],#4          ; WORD ALIGN T0 ARRAY ELEMENT
LOOP
        LDR R4,[R1],#4          ; WORD ALIGN T0 ARRAY ELEMENT

        CMP R2,R4               ; COMPARE NUMBERS
        BHI LOOP1               ; IF THE FIRST NUMBER IS > THEN GOTO LOOP1

        MOV R2,R4   ; IF THE FIRST NUMBER IS < THEN MOV CONTENT R4 TO R2
  LOOP1
        SUBS R5,R5,#1           ; DECREMENT COUNTER
        CMP R5,#0               ; COMPARE COUNTER TO 0
        BNE LOOP                ; LOOP BACK TILL ARRAY ENDS

        LDR R4,=RESULT          ; LOADS THE ADDRESS OF RESULT
        STR R2,[R4]             ; STORES THE RESULT IN R1

        NOP
        NOP
        NOP

   ARRAY OF 32 BIT NUMBERS(N=7)

   VALUE1
                DCD 0X44444444          ;
                DCD   0X22222222        ;
                DCD   0X11111111        ;
                DCD   0X33333333        ;
                DCD   0XAAAAAAAA        ;
                DCD   0X88888888        ;
                DCD   0X99999999        ;



        AREA DATA2,DATA,READWRITE               ; TO STORE RESULT IN GIVEN
  ADDRESS
  RESULT DCD 0X0

        END                     ; Mark end of file
```

****