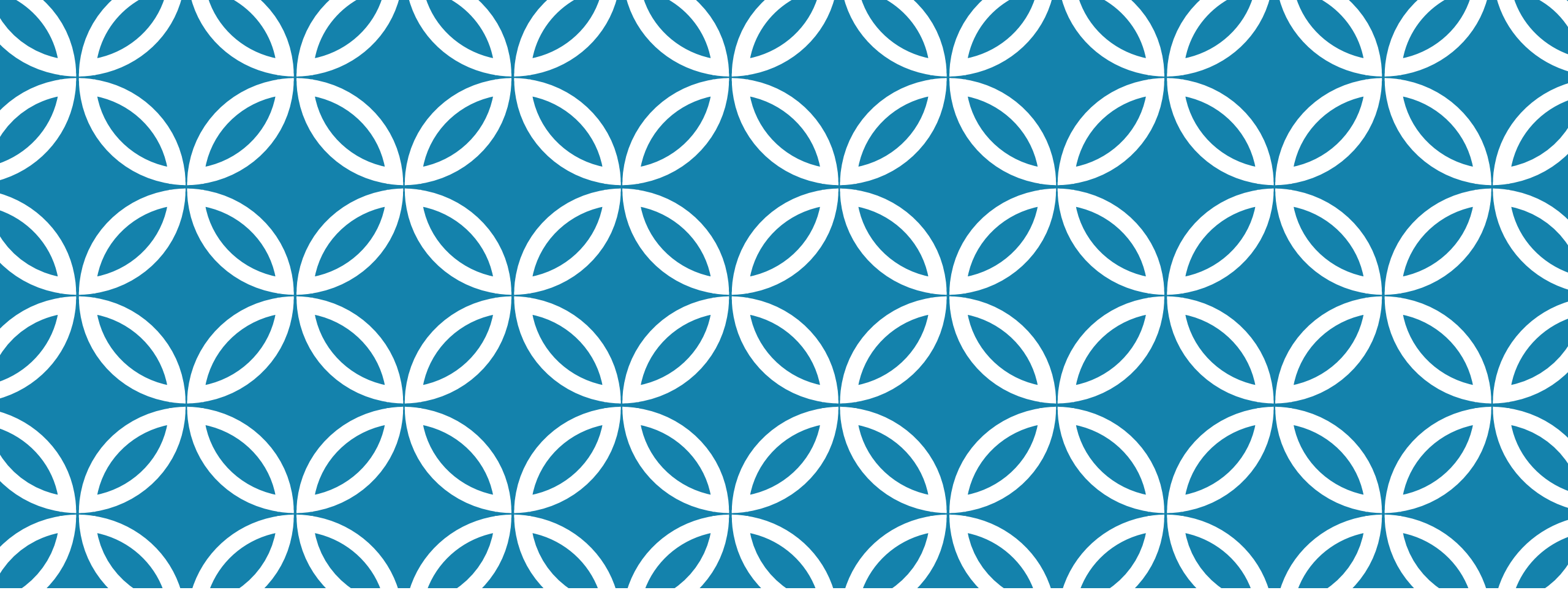


ARM – ASSEMBLY LANGUAGE PROGRAMMING

AGENDA

1. Basic Structure
2. Assembler Directives in ARM
3. Sample Programs

GENERAL STRUCTURE

```
label    operation operand      comments
main:
         LDR       R1, value    @ load value
         STR       R1, result
         SWI       #11

value:   .word     0x0000C123
result:  .word     0
```

ASSEMBLER DIRECTIVES IN ARM V4

- TTL – Stands for title, optional
- AREA – Specifies an area
  - Parameters - a {name}, type {CODE/DATA} and accessibility status [READONLY, READWRITE etc.]
- Define Data
  - DCB – Define Byte
  - DCW – Define Word
  - DCD – Define Constant Data (Half-word aligned)
- ALIGN – to word –align data
- EQU – to equate a name with data/address
- ENTRY and END

GENERAL POINTS

- Every program needs an area for CODE, which is usually READONLY
- The executable code is written between ‘ENTRY’ and ‘END’
- For a program with single function, the first statement can be prefixed with the label Main
- Usually the last statement before ‘END’ will be either ‘STOP B STOP’ or ‘SWI &11’
- When you try modifying DCD/DCB/DCW areas, you may get an error as the code area is READONLY.
- ; represents a comment
- # is used to indicate a constant

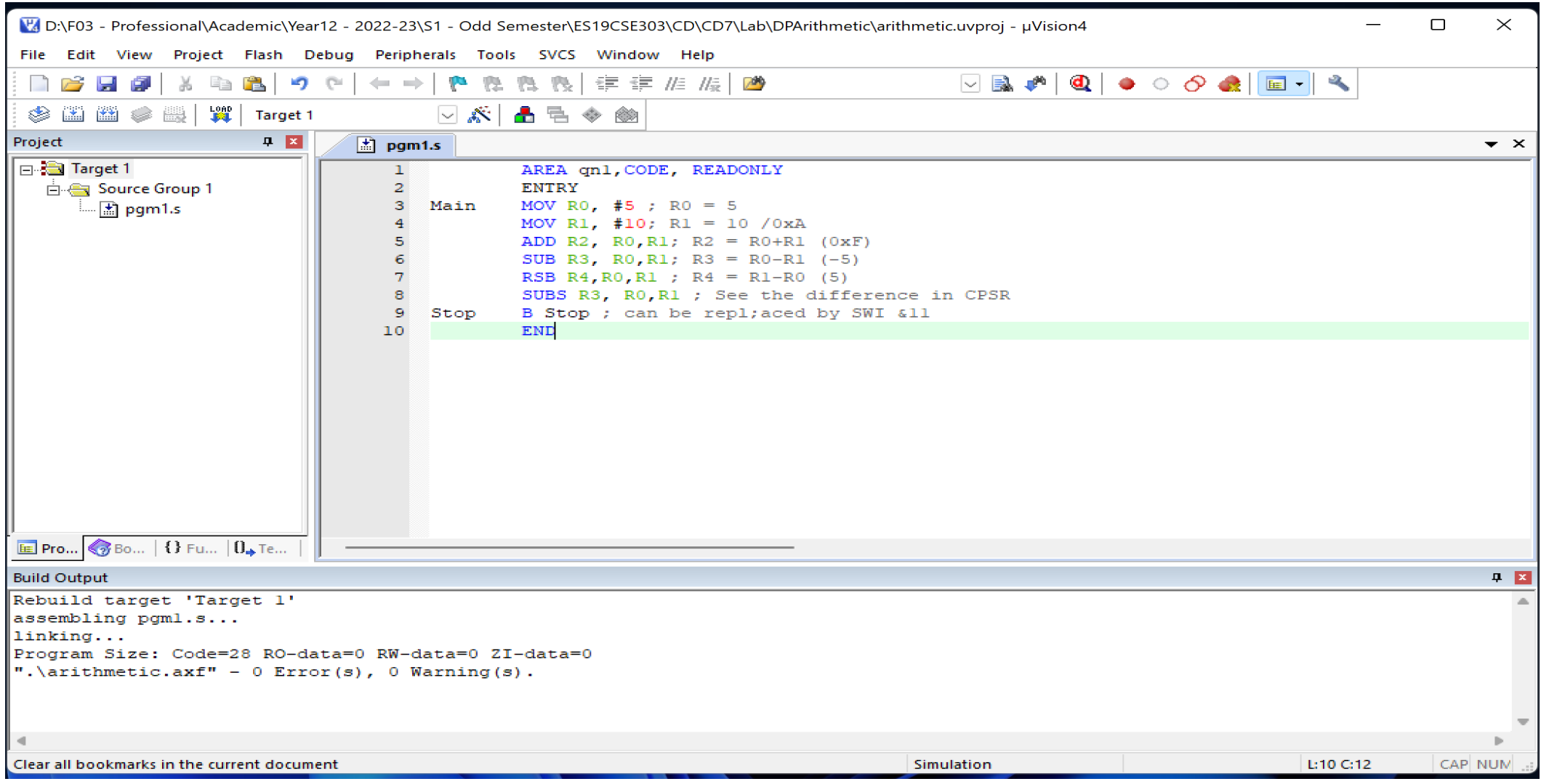
ARITHMETIC OPERATIONS

```
        AREA qn1,CODE, READONLY
        ENTRY
Main    MOV R0, #5 ; R0 = 5
        MOV R1, #10; R1 = 10 /0xA
        ADD R2, R0,R1; R2 = R0+R1 (0xF)
        SUB R3, R0,R1; R3 = R0-R1 (-5)
        RSB R4,R0,R1 ; R4 = R1-R0 (5)
        SUBS R3, R0,R1 ; See the difference in CPSR
Stop    B Stop ; can be repl;aced by SWI &11
        END
```

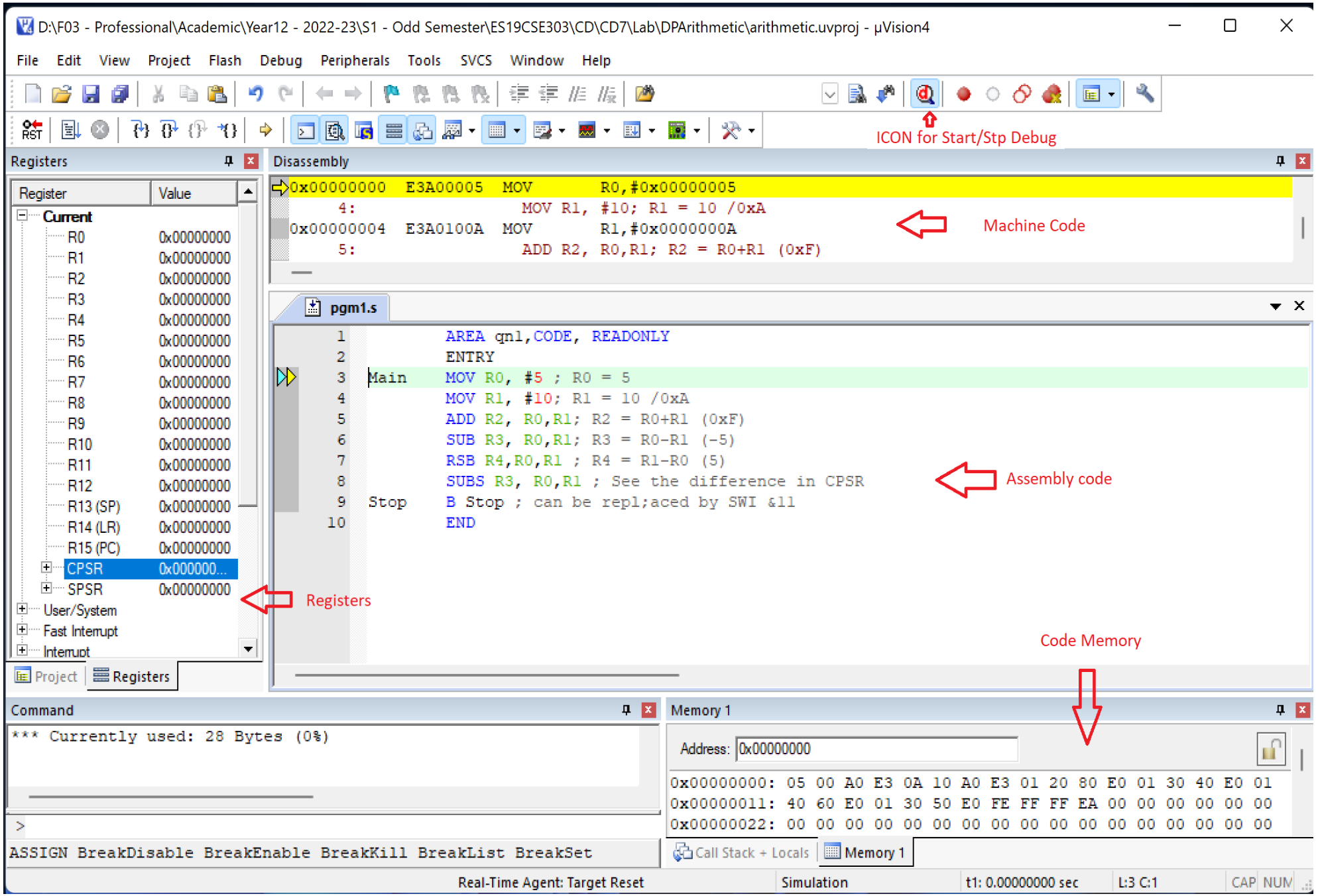
STEPS TO SEE RESULTS USING KEIL V4

1. Create a new project (Project → New Project → Give name)
2. Select device as ‘LPC2148’
3. Give ‘No’ for ‘ Copy ‘startup.s’ to Project Folder ?’ dialogue box
4. Create a new .s file (File → New → Save the file with .s extension (file type should be all files)
5. Write the code. Indentation/Spacing is very important.
6. Save all files and Build all files
7. Go to debug mode
8. Execute – either completely or go step-by-step

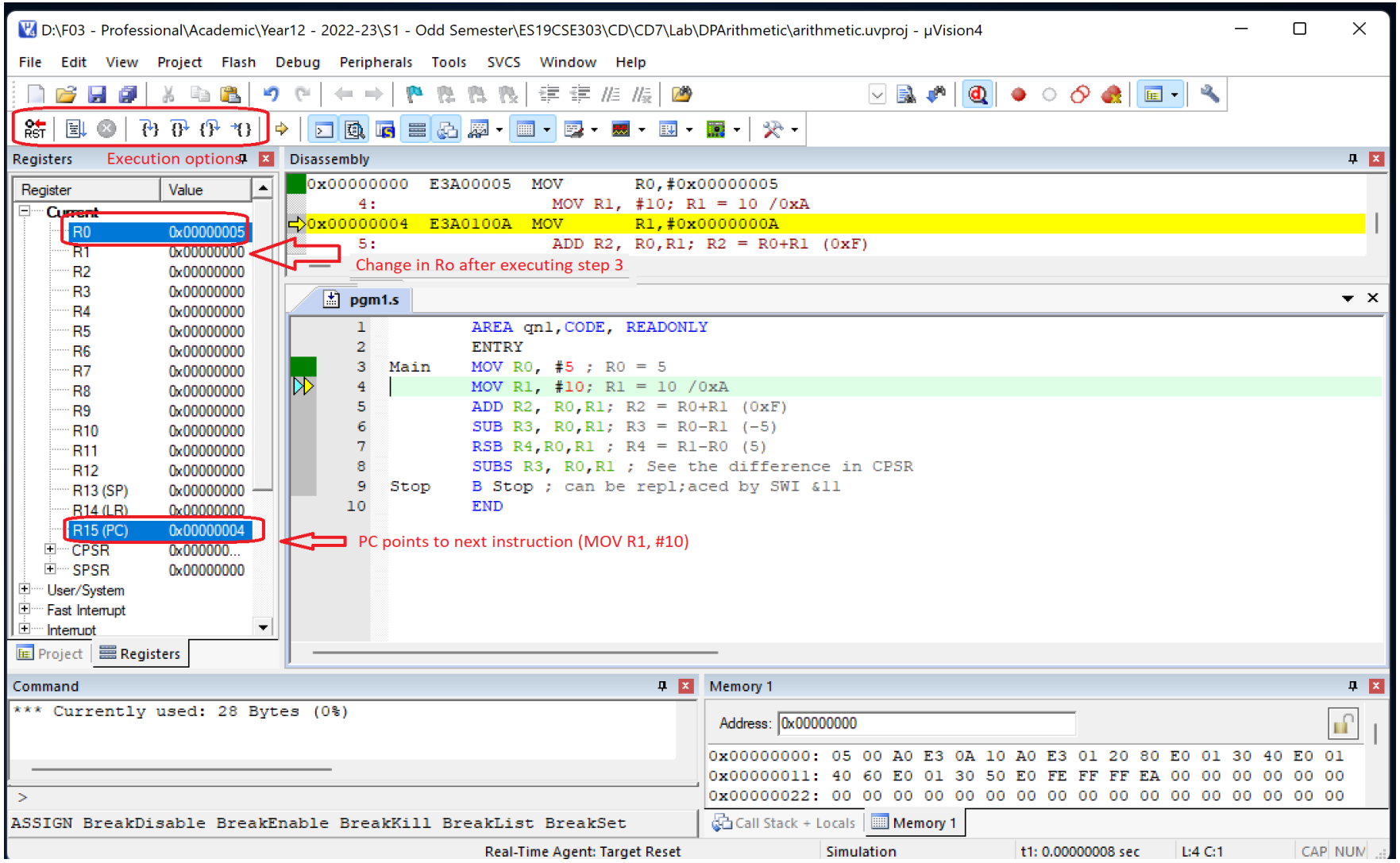
AFTER DOING TILL STEP 6



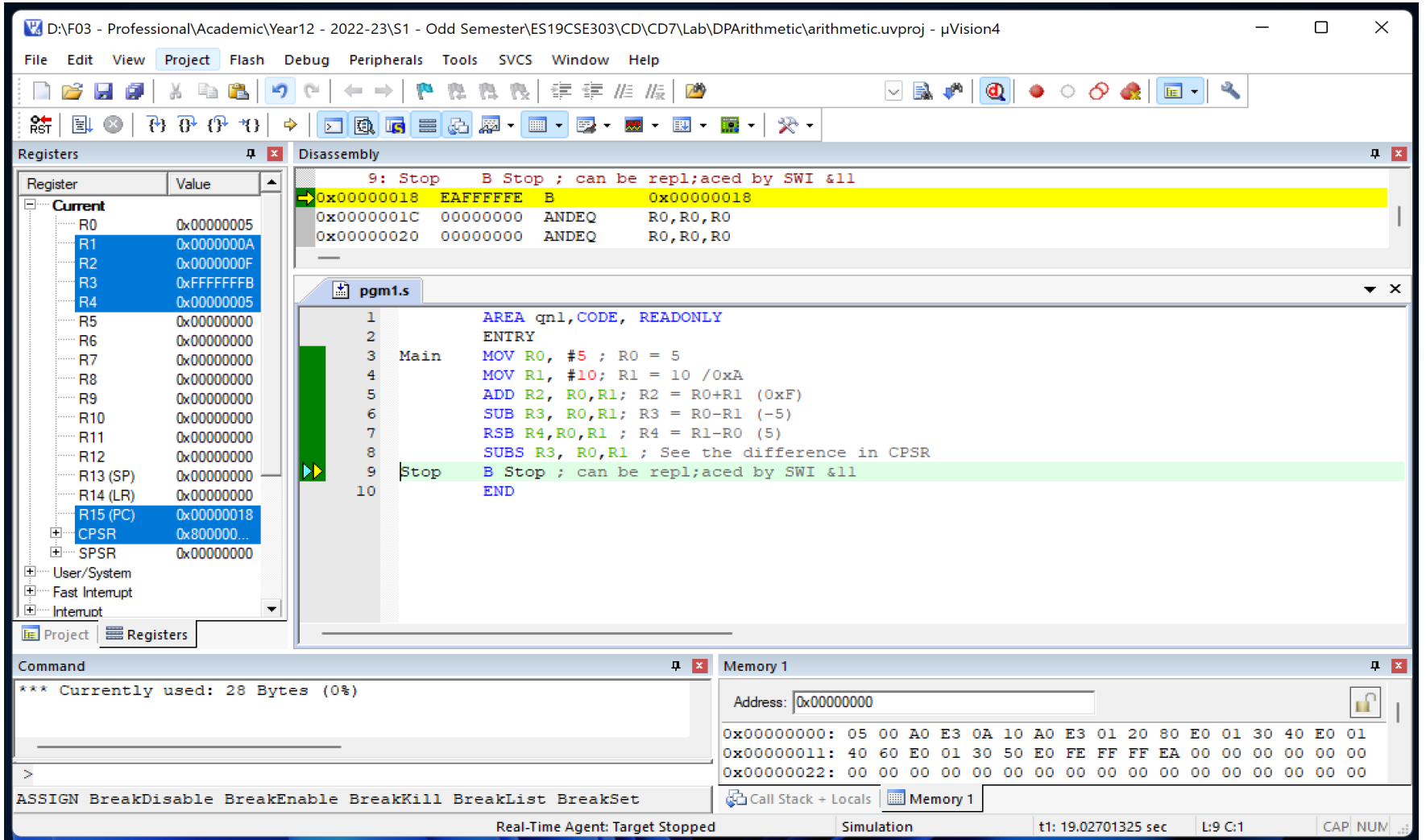
STEP 7


D:\F03 - Professional\Academic\Year12 - 2022-23\S1 - Odd Semester\ES19CSE303\CD\CD7\Lab\DPArithmetic\arithmetic.uvproj - µVision4
File Edit View Project Flash Debug Peripherals Tools SVCS Window Help
ICON for Start/Stp Debug
Registers
Register Value
Current
R0 0x00000000
R1 0x00000000
R2 0x00000000
R3 0x00000000
R4 0x00000000
R5 0x00000000
R6 0x00000000
R7 0x00000000
R8 0x00000000
R9 0x00000000
R10 0x00000000
R11 0x00000000
R12 0x00000000
R13 (SP) 0x00000000
R14 (LR) 0x00000000
R15 (PC) 0x00000000
CPSR 0x000000...
SPSR 0x00000000
User/System
Fast Interrupt
Interrupt
Registers
Disassembly
0x00000000 E3A00005 MOV R0,#0x00000005
4: MOV R1, #10; R1 = 10 /0xA
0x00000004 E3A0100A MOV R1,#0x0000000A
5: ADD R2, R0,R1; R2 = R0+R1 (0xF)
Machine Code
pgm1.s
1 AREA qn1,CODE, READONLY
2 ENTRY
3 Main MOV R0, #5 ; R0 = 5
4 MOV R1, #10; R1 = 10 /0xA
5 ADD R2, R0,R1; R2 = R0+R1 (0xF)
6 SUB R3, R0,R1; R3 = R0-R1 (-5)
7 RSB R4,R0,R1 ; R4 = R1-R0 (5)
8 SUBS R3, R0,R1 ; See the difference in CPSR
9 Stop B Stop ; can be repl;aced by SWI &11
10 END
Assembly code
Code Memory
Command
*** Currently used: 28 Bytes (0%)
ASSIGN BreakDisable BreakEnable BreakKill BreakList BreakSet
Memory 1
Address: 0x00000000
0x00000000: 05 00 A0 E3 0A 10 A0 E3 01 20 80 E0 01 30 40 E0 01
0x00000011: 40 60 E0 01 30 50 E0 FE FF FF EA 00 00 00 00 00 00
0x00000022: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
Call Stack + Locals
Memory 1
Real-Time Agent: Target Reset
Simulation
t1: 0.00000000 sec
L:3 C:1
CAP NUM

STEP 8 – EXECUTING FIRST INSTRUCTION

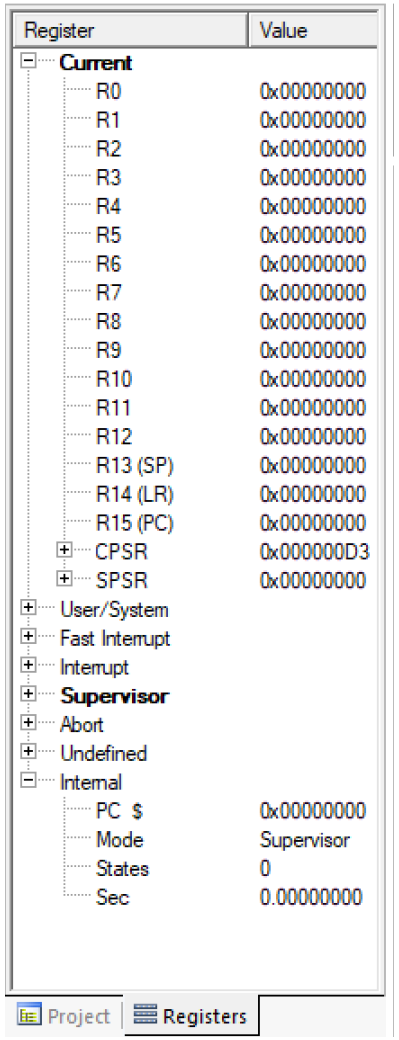


AFTER FINISHING THE EXECUTION – GETTING STUCK AT 'STOP B STOP' !!!

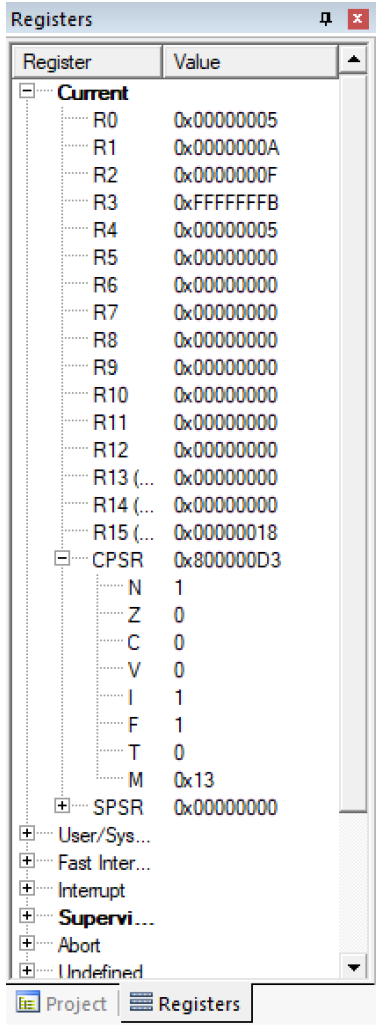


ARITHMETIC OPERATIONS WITH MULTIPLICATION

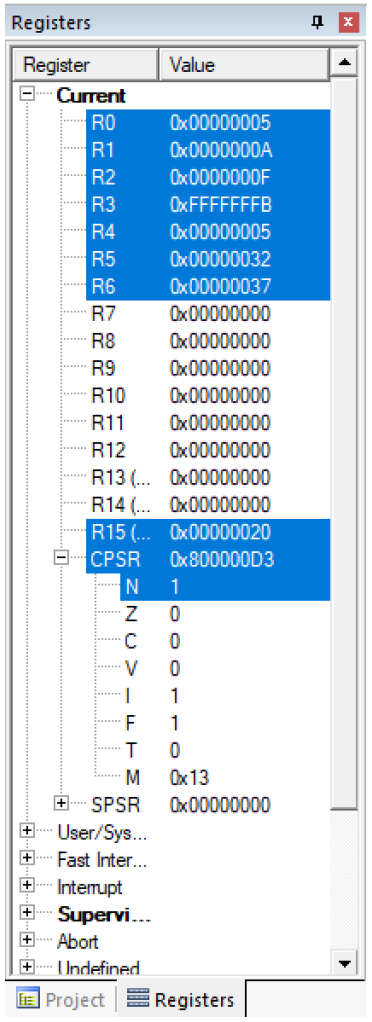
```
                AREA qn1,CODE, READONLY
                ENTRY
Main            MOV R0, #5 ; R0 = 5
                MOV R1, #10; R1 = 10 /0xA
                ADD R2, R0,R1; R2 = R0+R1 (0xF)
                SUB R3, R0,R1; R3 = R0-R1 (-5)
                RSB R4,R0,R1 ; R4 = R1-R0 (5)
                SUBS R3, R0,R1 ; See the difference in CPSR
                MUL R5,R0,R1 ; R5 = R0 * R1 = 50 (0x32)
                MLA R6,R0,R1,R0; R6 = 55 (0x37)
Stop            B Stop ; can be repl;aced by SWI &11
                END
```



Starting State



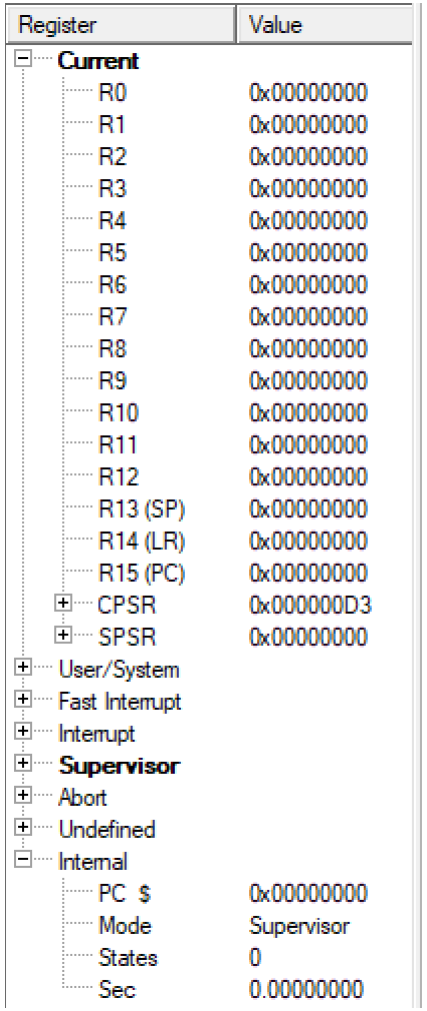
Before MUL



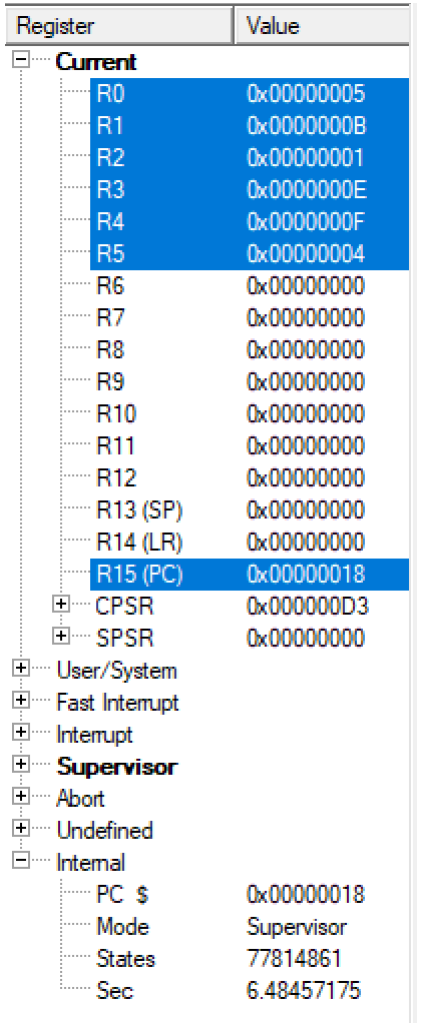
Execution Ended

LOGICAL OPERATIONS

```
        AREA qn1,CODE, READONLY
        ENTRY
Main    MOV R0, #5
        MOV R1, #11
        AND R2, R0,R1; R2 = 5 & 11 = 1
        EOR R3, R0,R1; R3 = 5 XOR 11 = 0xE
        ORR R4,R0,R1 ; R4 = 5 OR 11 = 0xF
        BIC R5, R0,R1 ; Performs (5) & not(11) = 0x4
        ;(NOT (11) = 0xFFFF FFF4
        ; NOT(11) & 5 = 0x0000 0004)
Stop    B Stop ; can be replaced by SWI &11
        END
```



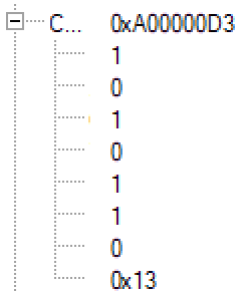
Before Execution

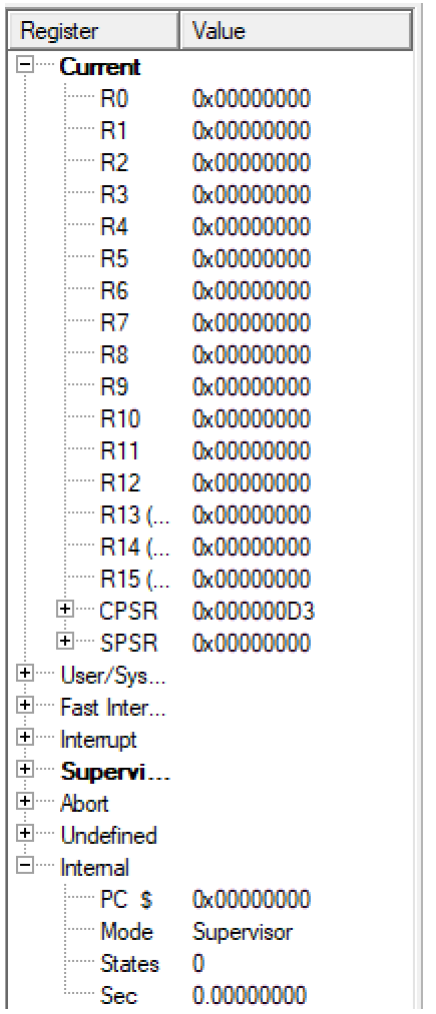


After Execution

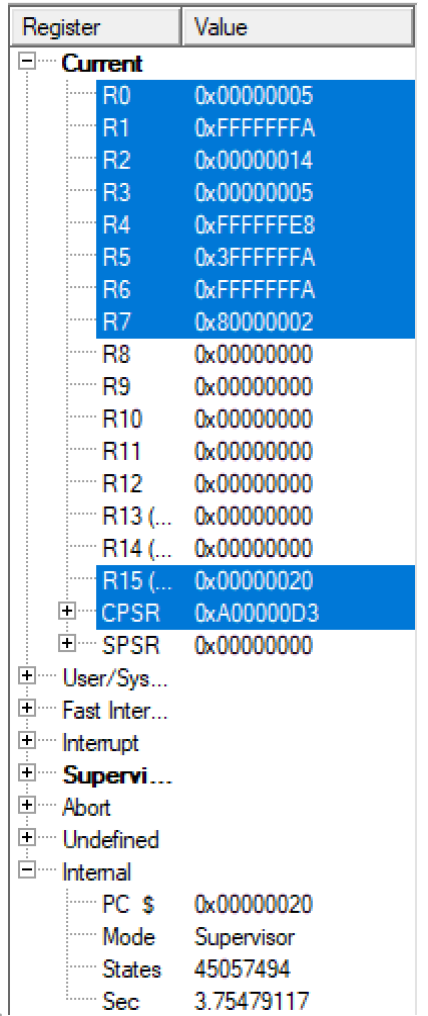
DATA MOVEMENT WITH BARREL SHIFTER OPERATIONS

```
        AREA qn1,CODE, READONLY
        ENTRY
Main    MOV R0, #5 ; R0 = 5
        MVN R1,R0 ; R1 = -(5) in 1's complement form -flipping of bits
        MOV R2, R0, LSL #2;R2 = R0 * 4 = 20 (Ox14)
        MOV R3, R2, LSR #2; R3 = R2/4 = 20/4=5
        MOV R4, R1, LSL #2 ; R4 = R1*4
        MOV R5, R4, LSR #2 ; MSB filled by 0s
        MOV R6, R4, ASR #2;MSB filled by sign bit - 0 for +ve numbers and 1 for -ve numbers
        MOVS R7, R0, ROR #1;LSb becomes MSb
Stop    B Stop ; can be replaced by SWI &11
        END
```





Before Execution



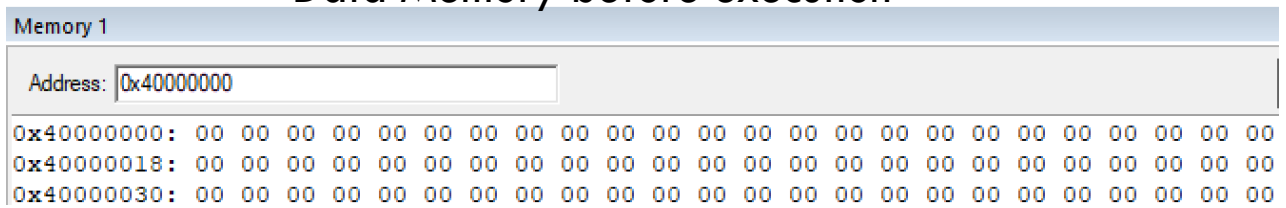
After Execution

STORE DATA IN MEMORY – STR WITH BRANCH

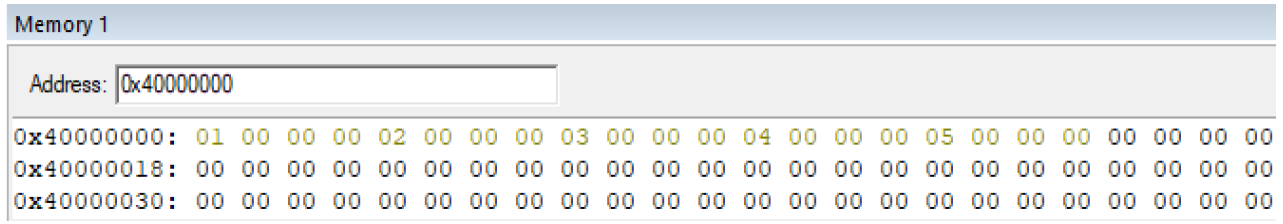
Qn. Initialize an array of size 5 with numbers from 1 to 5

```
        AREA qn1,CODE, READONLY
        ENTRY
Main    LDR R1,=0x40000000
        ; Base address of Array 0x4000 0000 is the location from which data can be written
        MOV R2,#0
        MOV R3, #5;Number of elements to be stored/ Size of Array
LOOP    CMP R2,R3     ; Checking whether index has reached the limit
        BGE Stop      ; if index >= size, stop
        ADD R4, R2,#1 ; R4 = R2+1
        STR R4, [R1],#4
        ; storing R4 value into memory pointed by R1 and update R1 as R1+4
        ADD R2, R2,#1 ; update R2 as R2+1
        BNE LOOP; Repeat search if item is not found yet
Stop    B Stop
        END
```

Data Memory before execution



Data Memory after execution



LOAD DATA FROM MEMORY

Qn. Find the sum of array elements

```
        AREA qn1,CODE, READONLY
        ENTRY
Main    LDR R1,=0x40000000
        MOV R2,#0 ;register for array index
        MOV R3, #5;Number of elements to be stored/ Size of Array
        MOV R5,#0 ;register for array sum
LOOP    CMP R2,R3 ; Checking whether index has reached the limit
        BGE Stop    ; if index >= size, stop
        LDR R4, [R1],#4 ; Fetch MEM[R1] to R4; update R1 as R1+4
        ADD R5,R5,R4 ; sum of array elements
        ADD R2, R2,#1 ; update R2 as R2+1
        BNE LOOP; Repeat search if item is not found yet
Stop    B Stop
        END
```

Before Execution – Initial Array

```
0x40000000: 01 00 00 00 02 00 00 00 03 00 00 00 04 00 00 00 05 00 00 00 00 00 00 00
0x40000018: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x40000030: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```

After Execution – All elements accessed

Memory 1

Address: 0x40000000

```
0x40000000: 01 00 00 00 02 00 00 00 03 00 00 00 04 00 00 00 05 00 00 00 00 00 00 00
0x40000018: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x40000030: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```

Registers –Initial values

Register	Value
Current	
R0	0x00000000
R1	0x00000000
R2	0x00000000
R3	0x00000000
R4	0x00000000
R5	0x00000000
R6	0x00000000
R7	0x00000000
R8	0x00000000
R9	0x00000000
R10	0x00000000
R11	0x00000000
R12	0x00000000
R13 (SP)	0x00000000
R14 (LR)	0x00000000
R15 (PC)	0x00000000
CPSR	0x000000...
SPSR	0x00000000

Registers – Final Values

Register	Value
Current	
R0	0x00000000
R1	0x40000014
R2	0x00000005
R3	0x00000005
R4	0x00000005
R5	0x0000000F
R6	0x00000000
R7	0x00000000
R8	0x00000000
R9	0x00000000
R10	0x00000000
R11	0x00000000
R12	0x00000000
R13 (SP)	0x00000000
R14 (LR)	0x00000000
R15 (PC)	0x00000028
CPSR	0x600000...
SPSR	0x00000000