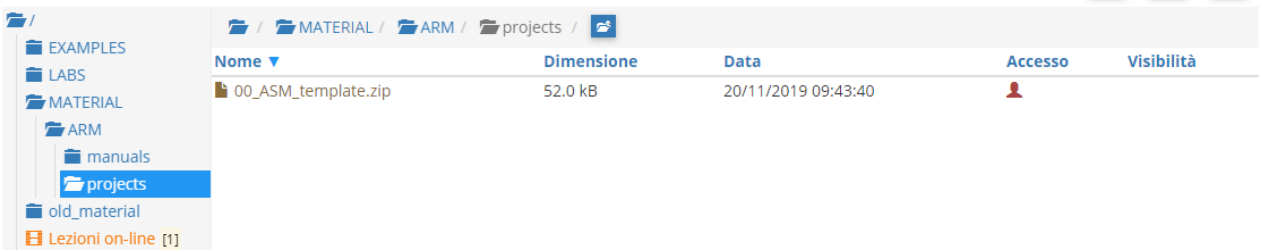
Architetture dei Sistemi di Elaborazione **Laboratory 6**	Delivery date: 26/11/2021 Expected delivery of lab_06.zip must include: - Solutions of the exercises 1, 2 and 3 - this document compiled possibly in pdf format.

Starting from the ASM_template project (available on Portale della Didattica), solve the following exercises:



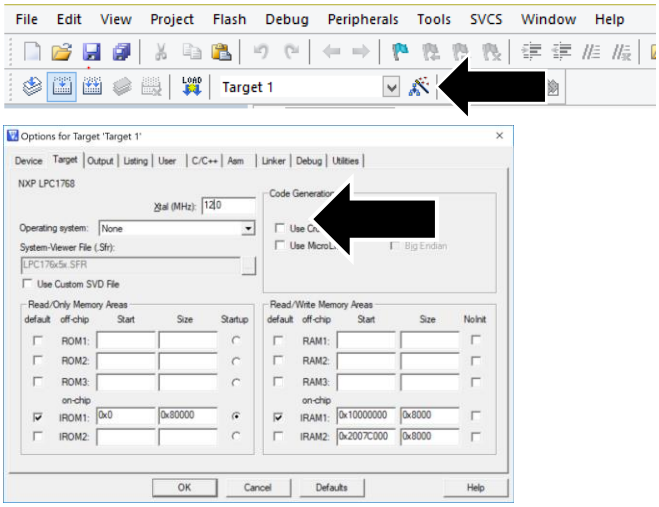
1) Write a program using the ARM assembly that performs the following operations:
   a. Sum R0 to R1 (R0+R1) and store the result in R2
   b. Subtract R4 to R3 (R3-R4) and store the result in R5
   c. Force, using the debug register window, a set of specific values to be used in the program to provoke the following flag to be updated **once at a time** (whenever possible) to 1:
      - carry
      - overflow
      - negative
      - zero
   d. Report the selected values in the table below.

	Please, report the hexadecimal representation of the values			
Updated flag	R0 + R1		R3 - R4	
	R0	R1	R3	R4
Carry = 1	0xFFFFFFFF	0x00000001	0x00000001	0x00000000
Carry = 0	0x00000000	0x00000001	0x00000000	0x00000001
Overflow	0x7FFFFFFF	0x00000001	0x80000000	0x7FFFFFFF
Negative	0x7FFFFFFF	0x00000001	0x00000000	0x00000001
Zero	0x00000000	0x00000000	0x00000001	0x00000001

Please explain the cases when it is **not** possible to force a **single** FLAG condition:
con la add non puoi settare solo il flag overflow perché se c'è overflow il risultato è negativo in complemento a 2.
È impossibile settare lo zero flag in una sub senza settare il carry flag perché se il risultato della sub è zero allora i due valori sottratti sono uguali quindi non c'è bisogno di borrow durante l'operazione e quindi verrà per forza settato anche il carry flag a 1 (arm implemaenta un carry flag inverso nel caso della sub, cioè se c'è il borrow non setta il carry)

2) Write two versions of a program that performs the following operations:
   a. Initialize registers R2 and R3 to random signed values
   b. Compare the two registers:
      - If they differ, store in the register R4 the minimum among R2 and R3
      - Otherwise, perform an arithmetic right shift of R3, sum R2 and store the result in R5

First, solve it resorting to 1) a traditional assembly programming approach using conditional branches and then compare the execution time with a 2) conditional instructions execution approach.
Report the execution time in the two cases in the table that follows: **NOTE**, report the number of clock cycles (cc) considering a cpu clock (clk) frequency of 12 MHz, as well as the simulation time in milliseconds (ms).
Notice that the processor clock frequency is setup in the menu "*Options for Target: 'Target 1'*".



	R0==R1 [cc]	R0==R1 [ms]	R0!=R1 [cc]	R0!=R1 [ms]
1) Traditional	8	0.00067	7	0.00058
2) Conditional Execution	10	0.00083	10	0.00083

3) Write a program that calculates the **Hamming distance** between two values. The Hamming distance is defined as the number of positions at which the corresponding values are different: e.g., the Hamming distance between the values *0b10101101* and *0b10010101* is 3. The initial values are stored in R0 and R1, while the resulting Hamming distance must be stored in R2.

Implement the ASM code that performs the following operations:

a. It determines whether the content of R2 is odd or even.
b. As a result, the values of R0 and R1 are updated as follows:
   - If R2 is even, the program clears the 11th bit of R0 and sets to 1 the 6th bit of R1 (all other bits must remain unchanged)
   - Else, the program copies in R1 the values of the flags.
c. Report code size and execution time (with 15MHz clk) in the following table.

	Code size [Bytes]	Execution time [*replace this with the proper time measurement unit*]	
		If R2 is even	Otherwise
Exercise 3) computation	564	0.00001080	0.00001067

ANY USEFUL COMMENT YOU WOULD LIKE TO ADD ABOUT YOUR SOLUTION: