Starting from the ASM_template project (available on Portale della Didattica), solve the following exercises:
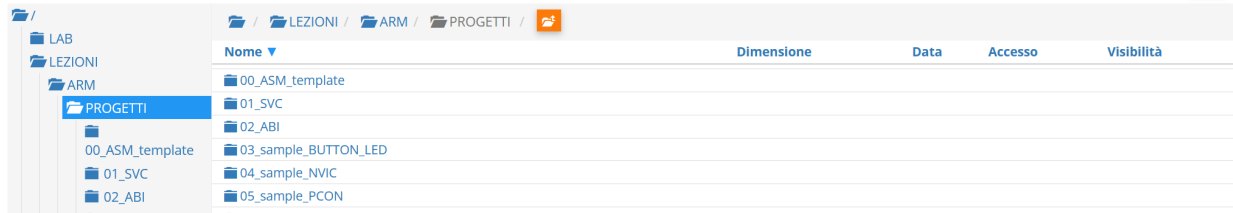
1) Write a program using the ARM assembly that performs the following operations:
   a. Initialize registers R3 and R4 to random signed values
   b. Sum R0 to R3 (R0+R3) and store the result in R2
   c. Subtract R4 to R2 (R4-R2) and store the result in R5
   d. Force, using the debug register window, a set of specific values to be used in the program to provoke the following flag to be updated **once at a time** (underline possible) to 1:
      - carry
      - overflow
      - negative
      - zero
   e. Report the selected values in the table below.

Updated flag	Please, report the hexadecimal representation of the values			
	R0 + R3		R4 – R2	
	R0	R3	R4	R2
Carry = 1	0xCD	0xFFFFFFFA	0xFFFFF	0xC7
Carry = 0	0xCD	0x00000FFA	0xFFFFFF	0x10C7
Overflow	0x7FFFFFFF	0x1	0xFFFFFFFF	0x80000000
Negative	0x0FFFFFFF	0x1	0x1	0x10000000
Zero	0x0FFFFFFF	0x1	0x10000000	0x10000000

Please explain the cases when it is **not** possible to force a **single** FLAG condition:

1) Non è possibile forzare il flag NEGATIVE ad un valore prefissato nel caso in cui si verifichi un OVERFLOW in quanto l'OVERFLOW determina la corruzione del bit di segno.
2) Non è possibile forzare il flag CARRY nell'operazione di sottrazione di un valore più piccolo ad uno più grande in quanto l'architettura realizza un preset del CARRY a 1 in questo tipo di operazioni di sottrazione.

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 R5 the maximum among R2 and R3

• Otherwise, perform a logical right shift of R3, sum R2 and store the result in R4

First, solve it by 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 16 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'*".
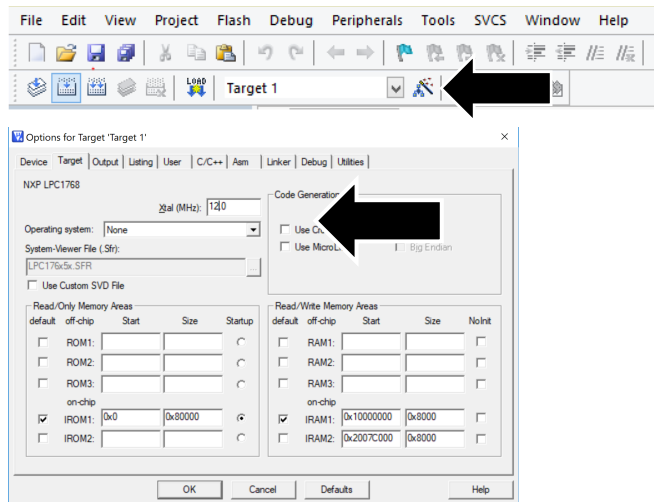
	R2==R3 [cc]	R2==R3 [ms]	R2! =R3 [cc]	R2! =R3 [ms]
1) Traditional	13	0.00108	14	0.00117
2) Conditional Execution	12	0.00100	12	0.00100

3) Write a program that calculates the leading zeros of a variable. The leading zeros are computed by counting the number of zeros starting from the most significant bit and stopping at the first 1 encountered: e.g., the leading zeros of 0b00000101 are 5. The variable to check is in R1. After the count, if the number of leading zeros is odd, perform the sum between R2 and R3. If the number of leading zeros is even, perform the difference between R2 and R3. In both cases the result is placed in R4.

Implement the ASM code that performs the following operations:
   a. Determines whether the number of leading zeros of R1 is odd or even.
   b. As a result, the value of R4 is computed as follows:
       • If the leading zeros are even, R4 is the difference between R2 and R3
       • Else, R4 is the sum of R2 and R3
   c. Report code size and execution time (with 15MHz clk) in the following table.

	Code size [Bytes]	Execution time [μs]	
		If R2 is even	Otherwise
Exercise 3) computation	564	3.17	2.67

Il numero di "leading zeros" è conteggiato realizzando ciclicamente uno shift verso sinistra del registro R1 valutando volta per volta il flag CARRY al fine di incrementare il contatore oppure uscire dal loop di shifting. Per valutare la parità/disparità del numero di "leading zeros" è stato effettuato uno shift verso destra del registro contatore valutando ancora una volta il flag CARRY (0 = contatore PARI, 1 = contatoreDISPARI).