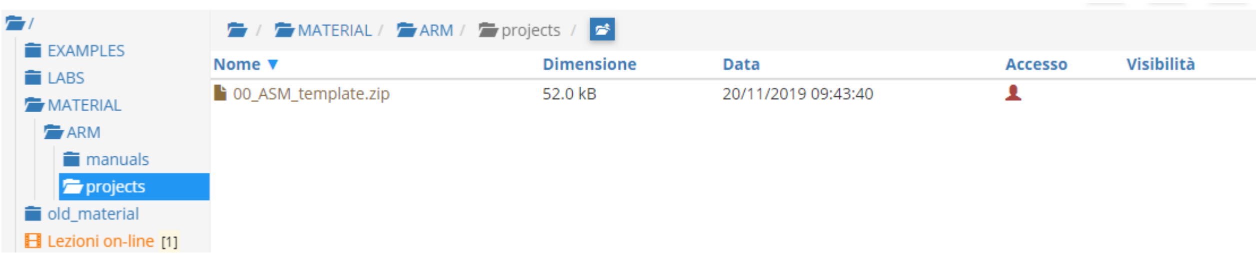
Architetture dei Sistemi di Elaborazione	Delivery date: 26/11/2021
Laboratory 6	Expected delivery of lab_06.zip must include: - Solutions of the exercises 1, 2 and 3 - this document compiled possibly in pdf format.

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.

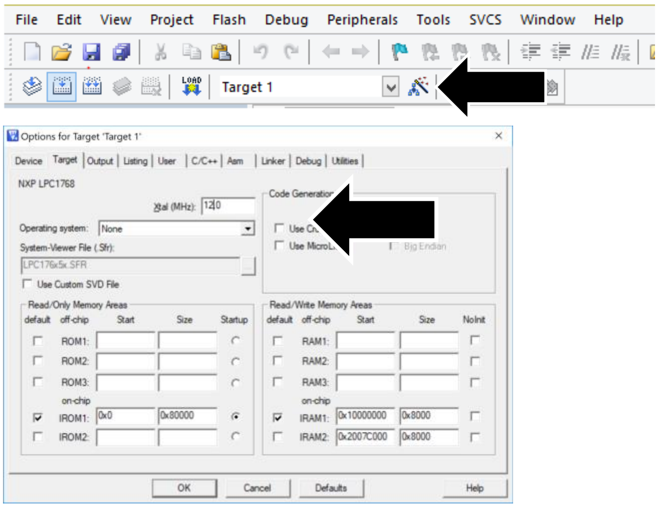
	Please, report the hexadecimal representation of the values			
Updated flag	R0 + R1		R3 - R4	
	R0	R1	R3	R4
Carry = 1	0x70000000	0xC0000000	0x00000001	0x00000000
Carry = 0	0x00000000	0x00000001	0x00000000	0xF0000000
sOverflow	0x70000000	0x70000000	0x80000000	0x00000001
Negative	0x80000000	0x00000000	0x00000000	0x00000001
Zero	0x00000000	0x00000000	0x00000000	0x00000000

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

- Quando c'è overflow nella add si setta anche il flag N a 1 in quanto 0x7 + 0x7 è la somma di due numeri positivi che danno un numero negativo. Un altro modo di farlo era mettendo due numeri negativi e sommarli (ad esempio 0x8 + 0x8), ma in questo modo si attivava anche il bit di carry.
- Nella sub per avere Zero=1 ho anche Carry=1 in quanto non ho borrow ma il flag nella sottrazione è invertito per cui è a 1.
- Nella sub per avere overflow=1 devo avere segni opposti, per cui posso scegliere di avere il sottraendo positivo ma avrò carry=1 (non ho borrow) oppure sottraendo negativo ma avrò N=1 (risultato negativo).

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'*".



	R2==R3 [cc]	R2==R3 [ms]	R2!=R3 [cc]	R2!=R3 [ms]
1) Traditional	14	0.00117	15	0.00125
2) Conditional Execution	14	0.00117	14	0.00117

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 *0b1010101* and *0b1001001* 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 11^{th} bit of R0 and sets to 1 the 6^{th} 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 [*ms*]	
		If R2 is even	Otherwise
Exercise 3) computation	564	0.00200	0.0038

ANY USEFUL COMMENT YOU WOULD LIKE TO ADD ABOUT YOUR SOLUTION:
Il risultato di tempo tra even/otherwise dipende molto dai dati ingresso, in quanto ho inserito un’ottimizzazione che esce dal ciclo principale se la XOR shiftata tra r0 ed r1 contiene solo più zeri. Per cui se le differenze di bit sono all’inizio l’esecuzione sarà più veloce, altrimenti impiegherà più tempo.