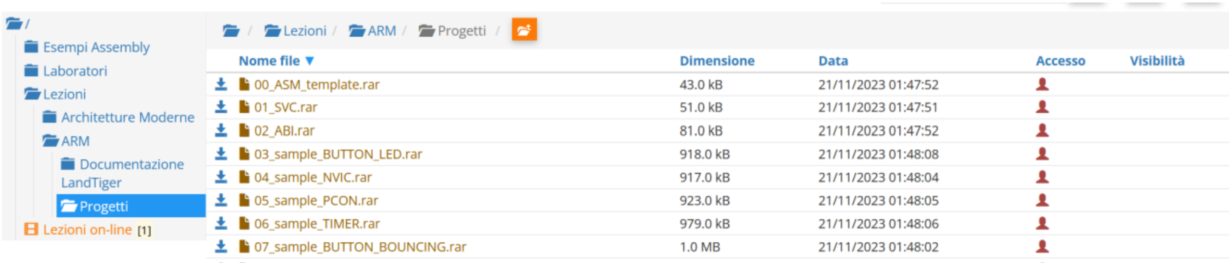
Architetture dei Sistemi di Elaborazione	Delivery date: 30th November 2023
Laboratory 6	Expected delivery of lab_06.zip must include: - Solutions of the exercises 1, 2 and 3 - this document compiled possibly in pdf format.

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. Initialize registers R1, R3 and R4 to random signed values
   b. Sum R1 to R3 (R1+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** (whenever possible) to 1:
      - carry
      - overflow
      - negative
      - zero
   e. Report the selected values in the table below.

	Please, report the hexadecimal representation of the values			
Updated flag	R1 + R3		R4 – R2	
	R1	R3	R4	R2
Carry = 1	0x1	0x2	0x4	0x3
Carry = 0	0x1	0x2	0x4	0x3
Overflow	0x7FFFFFFF	0x00000001	0x1	0x80000000
Negative	0x1	0x2	0x2	0x3
Zero	0x0	0x0	0x0	0x0

Please explain the cases when it is **not** possible to force a **single** FLAG condition:
Non è possibile forzare un singolo flag nel caso dell'overflow: infatti se questo scatta in seguito ad una somma di due numeri positivi che farebbe scattare il MSB a 1 si avrebbe anche negative flag= 1. Nel caso opposto invece, ovvero quelli di due numeri negativi che si sommano, si avrebbe oltre all'overflow anche carry=1

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 minimum among R2 and R3

- Otherwise, perform on R3 a logical left shift of 1 (is it equivalent to what?), sum R2 and store the result in R4 (i.e, r4=(r3<<1)+r2).

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), as well as the simulation time in milliseconds (ms) considering a cpu clock (clk) frequency of 16 MHz.
Refer to the guide "howto_setup_keil" to change the clock frequency in Keil.

	R2==R3 [cc]	R2==R3 [ms]	R2! =R3 [cc]	R2! =R3 [ms]
1) Traditional	11	0.00069	11	0.00069
2) Conditional Execution	15	0.00094	15	0.00094

3) Write a program that calculates the trailing zeros of a variable. The trailing zeros are computed by counting the number of zeros starting from the least significant bit and stopping at the first 1 encountered: e.g., the trailing zeros of 0b10100000 are 5. The variable to check is in R1. After the count, if the number of trailing zeros is odd, perform the sum between R2 and R3. If the number of trailing 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 trailing zeros of R1 is odd or even.
   b. As a result, the value of R4 is computed as follows:
      - If the trailing 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 [*replace this with the proper time measurement unit*]	
	If R1 is even	Otherwise
560	0.00333	0.0012

ANY USEFUL COMMENT YOU WOULD LIKE TO ADD ABOUT YOUR SOLUTION:
Ovviamente se il numero non ha zeri da controllare il programma finisce prima e il tempo d'esecuzione sarà minore; più zeri ci sono, più cicli servono e più tempo ci vorrà. Nella tabella ho inserito il tempo per R1 = 16