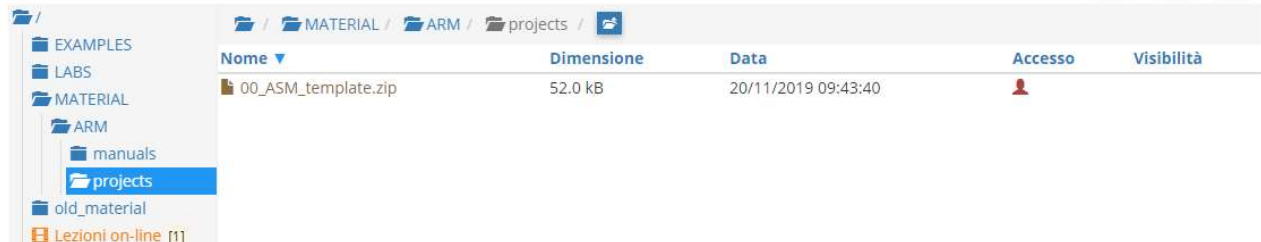


Expected delivery of **lab_06.zip** must include:

- 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:
- Sum R0 to R1 ($R0+R1$) and store the result in R2
 - Subtract R4 to R3 ($R3-R4$) and store the result in R5
 - 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
 - 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	0x000000CC	0xFFFFFFF35	0x000000D5	0x000000D4
Carry = 0	0x00000003	0x00000005	0x0000000A	0xFFFFFFFEC
Overflow	0xA1111111	0xB5555555	0x7FFFFFFF	0x80000001
Negative	0xFFFFFFFDD	0x00000002	0x00000002	0x00000014
Zero	0x00000000	0x00000000	0x00000000	0x00000000

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

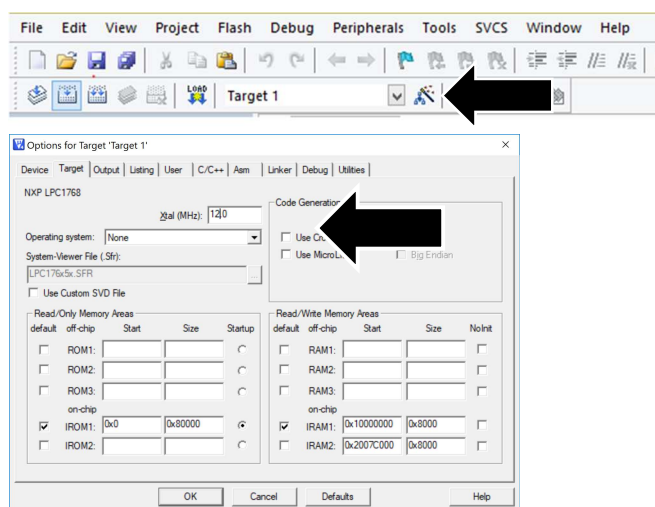
- ➔ Il flag Overflow (V) non può essere settato senza settare anche altri flag;
ADDS:
Se entrambi gli operandi sono positivi e $V=1$, allora anche $N=1$.
Se entrambi gli operandi sono negativi e $V=1$, allora anche $C=1$.
SUBS:
Se sottraendo +, minuendo -, e $V=1$, allora anche $C=1$
Se sottraendo -, minuendo +, e $V=1$, allora anche $N=1$
- ➔ Il flag Zero non è impostabile singolarmente. In complemento a due il negativo di un numero si ottiene invertendo tutti i bit e sommando 1. Ciò significa che appena si ha una somma tra 1, il carry si propagherà fino alla fine della somma. L'unico caso possibile è quello in tabella (entrambi operandi a 0).

- 2) Write two versions of a program that performs the following operations:
- Initialize registers R2 and R3 to random signed values
 - 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	12 cc	0.001 ms	12 cc	0.001 ms
2) Conditional Execution	12 cc	0.001 ms	12 cc	0.001 ms

- 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:

- It determines whether the content of R2 is odd or even.
- 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.
- 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	0.00327 ms	0.00327 ms

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