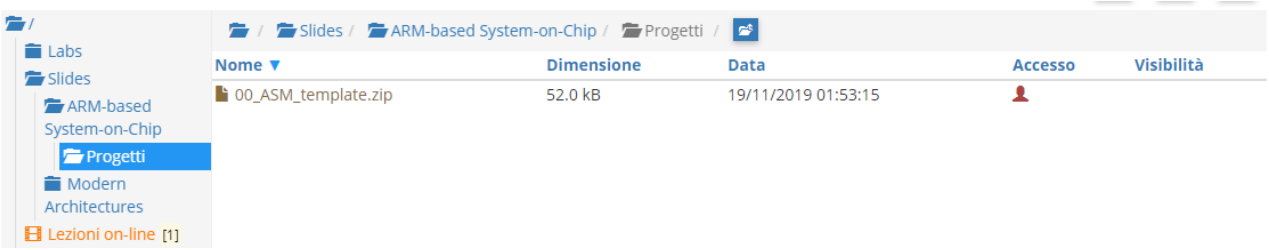
Architetture dei Sistemi di Elaborazione **02GOLOV [AA-LZ]** **Laboratory** **5**	Delivery date: Wednesday, November 27, 2019 Expected delivery of lab_05.zip must include: - Solutions of the exercises 1, 2 and 3 - this document compiled possibly in pdf format.

Solve the following problems by starting from the ASM_template project. Download from the Portale della Didattica.

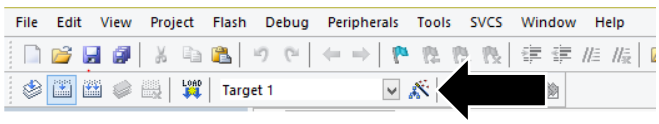


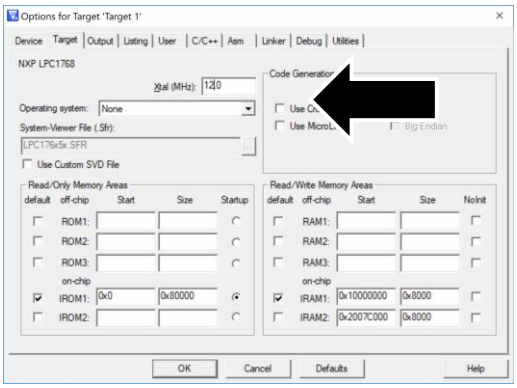
1) Write a program using the ARM assembly that makes the following simple operations:
   a. Sum R0 to R1 (R0+R1) and stores the result in R2
   b. Subtract R4 to R3 (R4-R3) and stores the result in R5
   c. Select and force, using debug register window, a set of specific values to be used in the program in order to provoke the following flag to be updated 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
Carry = 1	0000 00CD	FFFF FF32	FFFF FF32	FFFF FFFF
Carry = 0	0000 00CD	0000 00AE	FFFF FFFF	FFFF FF32
Overflow	A111 1111	B555 5555	4AAA AAAB	A111 1111
Negative	0000 00CD	F000 0000	0010 0000	FFFF FF32
Zero	0000 00CD	0000 FFFF	FF00 0000	FF00 0000

2) Write two versions of a program that performs the following operations:
   a. Setup registers R0 and R1 to random signed values
   b. Compare the registers:
      - If they differ, find and store in the register R2 the maximum among R0 and R1
      - Otherwise, rise R0 to the square, sum R1 and store the result in R3

   Solve the problem by adopting a 1) conditional execution approach and compare the execution time with a 2) a traditional assembly programming approach. Report the execution time in the two cases in the table that follows: please, report the number of clock cycles (cc) considering a cpu clock (cclk) frequency of 12 MHz.
   Notice that the processor clock frequency is setup in the menu "*Options for Target: 'Target 1'*" .





# cc	R0==R1	R0!=R1 && R0>R1	R0!=R1 && R0<R1
1) Conditional execution	12	12	12
2) Traditional	12	9.96	12

3) Write a program able to indicate whether a register contains a value that shows “even” or “odd” parity. In mathematics, an integer is “even” if it is evenly divisible by two, and “odd” if it is not even. In computer science, the parity concept is different and usually indicates whether the total number of 1-bits in the string is even or odd. For example, the number 4 is showing to be odd (0100 ← just a single 1-bit), while the number 5 is even (0101← two 1-bits).

   Please implement the ASM code that performs the following checks and actions:
   a. It determines if the register R0 and R1 are showing the same c.s. parity,
   b. As a result, the value of R0 and R1 is updated as following
      - If R0 and R1 are both even or odd, then it clears the 16 MSB of R0 and sets to 1s the 8LSB of R1 (all other bits must remain unchanged)
      - Otherwise, it copies in R1 the values of the flags.
   c. Report code size and execution time (with 12MHz cclk) in the following table.

		Execution time microseconds	
	Code size	if both Odd or Even	Otherwise
Exercize 3) computation	564	28	27.83

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

Nell’esercizio 2 i cicli di clock (calcolati come tempo d’esecuzione*frequenza di clock) risultano 12 in tutti i casi tranne in quello con R0>R1 con implementazione tradizionale.

Sperimentalmente si osserva che, effettivamente, la combinazione delle istruzioni eseguite è tale da essere eseguita sempre in 12 cicli di clock, tranne per il caso prima citato.

Si riportano le tabelle di verifica del calcolo dei cicli di clock: si è calcolato, istruzione per istruzione, il tempo di esecuzione ed i rispettivi cicli di clock. Si sono sommati ed i valori ottenuti sono stati verificati con quelli precedentemente ottenuti.

CONDITIONAL EXECUTION									
R1>R0				R1<R0				R1=R0		
	ns	cc			ns	cc			ns	cc
	170	2,04			170	2,04			170	2,04
MOV	80	0,96		MOV	80	0,96		MOV	80	0,96
MOV	80	0,96		MOV	80	0,96		MOV	80	0,96
CMP	90	1,08		CMP	90	1,08		CMP	90	1,08
MOVGT	160	1,92		MOVGT	160	1,92		MOVGT	160	1,92
MOVLT	170	2,04		MOVLT	170	2,04		MOVLT	170	2,04
MULEQ	170	2,04		MULEQ	170	2,04		MULEQ	170	2,04
ADDEQ	80	0,96		ADDEQ	80	0,96		ADDEQ	80	0,96
tot	1000	12		**tot**	1000	12		**tot**	1000	12

TRADITIONAL									
R1>R0				R1>R0				R1>R0		
	ns	cc			ns	cc			ns	cc
	170	2,04			170	2,04			170	2,04
MOV	80	0,96		MOV	80	0,96		MOV	80	0,96
MOV	80	0,96		MOV	80	0,96		MOV	80	0,96
CMP	90	1,08		CMP	90	1,08		CMP	90	1,08
BGT	250	3		BGT	80	0,96		BGT	80	0,96
MOV	80	0,96		BLT	250	3		BLT	80	0,96
B	250	3		MOV	80	0,96		MUL	90	1,08
								ADD	80	0,96
								B	250	3
tot	1000	12		**tot**	830	9,96		**tot**	1000	12

ns: nanosecondi
cc: cicli di clock