

Architetture dei Sistemi di Elaborazione

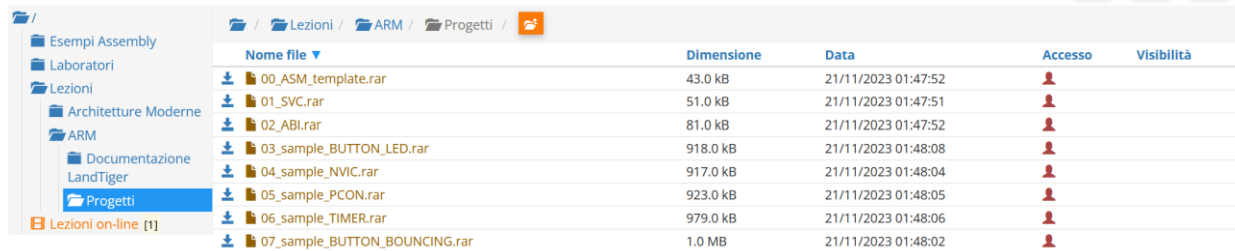
Delivery date:
30th November 2023

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.



Nome file	Dimensione	Data	Accesso	Visibilità
00_ASM_template.rar	43.0 kB	21/11/2023 01:47:52		
01_SVC.rar	51.0 kB	21/11/2023 01:47:51		
02_ABI.rar	81.0 kB	21/11/2023 01:47:52		
03_sample_BUTTON_LED.rar	918.0 kB	21/11/2023 01:48:08		
04_sample_NVIC.rar	917.0 kB	21/11/2023 01:48:04		
05_sample_PCON.rar	923.0 kB	21/11/2023 01:48:05		
06_sample_TIMER.rar	979.0 kB	21/11/2023 01:48:06		
07_sample_BUTTON_BOUNCING.rar	1.0 MB	21/11/2023 01:48:02		

- 1) Write a program using the ARM assembly that performs the following operations:
- Initialize registers R3 and R4 to random signed values
 - Sum R1 to R3 ($R1+R3$) and store the result in R2
 - Subtract R4 to R2 ($R4-R2$) 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	R1 + R3		R4 - R2	
	R1	R3	R4	R2
Carry = 1	0xFFFFFFFF	0x00000002	0x7FFFFFFF	0x00000001
Carry = 0	0x00000008	0x00000008	0x00000000	0xFFFFFFFF
Overflow				
Negative	0x00000005	0xFFFFFFFF	0x00000001	0x00000004
Zero	0x00000000	0x00000000		

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

ADDS

OverFlow ad 1: viene ottenuto sia negative a 1 che overflow a 1 con $r1=0x7FFFFFFF$ e $r3=0x00000001$, siccome causa un overflow nell'aritmetica con segno, poiché supera il valore massimo rappresentabile per un intero a 32bit con segno. Il risultato dovrebbe essere $0x80000000$ che è interpretato come il numero negativo più piccolo in complemento a 2 a 32 bit.

SUBS

-non è possibile ottenere $zero=1$ con flag del carry a 0, siccome è necessario che $r4$ e $r2$ siano uguali(nessun borrow), il che automaticamente imposta il carry a 1.

-non è possibile ottenere solo $overflow=1$ perché si verifica quando si sottrai un numero positivo da uno negativo (o viceversa) e il risultato supera i limiti di rappresentazione di un intero a 32 bit con segno. Nel mio caso provoca anche $carry=1$, siccome nella sottrazione il carry ad 1 indica il "no borrow", e nei casi di overflow c'è sempre un borrow. Quindi è impossibile avere solo il flag di overflow uguale ad 1.

- 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 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	0.000915	15	0.00125
2) Conditional Execution	13	0.001083	13	0.001083

- 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:

- 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 [<i>replace this with the proper time measurement unit</i>]	
	If R1 is even	Otherwise
56	0.001499ms	0.00217ms

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