

Architetture dei Sistemi di Elaborazione

Delivery date:

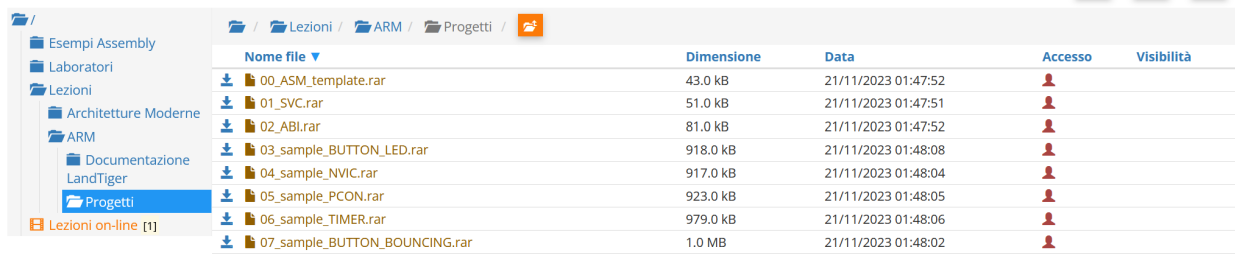
30th November 2023

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 R1, 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	0xFFFFFFFFE	0x00000003	0x00000002	0x00000001
Carry = 0	0x00000010	0x00000000	0x00000005	0x00000010
Overflow	0x7FFFFFFF	0x00000001	0x00000005	0x80000000
Negative	0xFFFFFFFFB	0x00000001	0xFFFFFFFF0	0xFFFFFFFFC
Zero	0x00000000	0x00000000	0x00000000	0x00000000

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

FLAG C=0 (carry) risulta impossibile impostarlo indipendentemente durante la SUB a causa del suo metodo implementativo. Poichè la sottrazione avviene come somma di un valore messo a complemento a due il carry sarà zero solo se il primo elemento è più grande, quindi creando un risultato negativo.

FLAG V (overflow) non può essere modificato in modo indipendente senza modificare altri flag quali il carry, zero o negativo. Tale risultato si dimostra facilmente in modo empirico considerando tutte le combinazioni di somma tra due registri a 2 bit con segno possibili (per semplicità non esplicito le ridondanze associative e le somme superflue che non provocano modifiche del flag di overflow):

10+	11+	01+	10+	11+
10	10	11	11	11
===	===	=====	=====	=====
100	101	100	100	110

Non è difficile a questo punto vedere come ogni operazione produca un risultato con valore pari a zero e/o con bit di segno pari a uno o riportando un carry, comportando così un evidente modifica dei rispettivi flag.

FLAG Z (zero) non può essere impostato singolarmente durante la SUBS in quanto per risultare a zero il valore sottratto trasformato in complemento a due porterà sempre un carry.

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	8	0.000665	11	0.000915
2) Conditional Execution	10	0.000833	10	0.000833

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 [<i>replace this with the proper time measurement unit</i>]	
	If R1 is even	Otherwise
560	1.75[μ s]	1.17[μ s]

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

Il programma avrà sempre tempi di esecuzioni diversi in quando la parità e disparità avverrà sempre su bit diversi e quindi ci sarà come minimo le tempistiche di un ciclo di controllo prima di uscire dal loop. A priori non vi è una fondamentale differenza di tempistiche causate da algoritmi più complessi se legati al numero di zeri pari o dispari, l'unico elemento da notare è che nel caso in cui R1 sia pari si farà sempre almeno un ciclo di controllo, mentre se dispari significa che il lsb=1 e quindi il programma non farà nemmeno uno shift. In questo caso avendo impostato R1 even come 2_xxx10 (x valori irrilevanti) si noti come la differenza dei due valori $1.75 - 1.17 = 0.58[\mu\text{s}]$ rappresenta la durata del ciclo, per cui per ogni "valore non 1" ci costa $0.58[\mu\text{s}]$ in termini di tempi di esecuzioni.