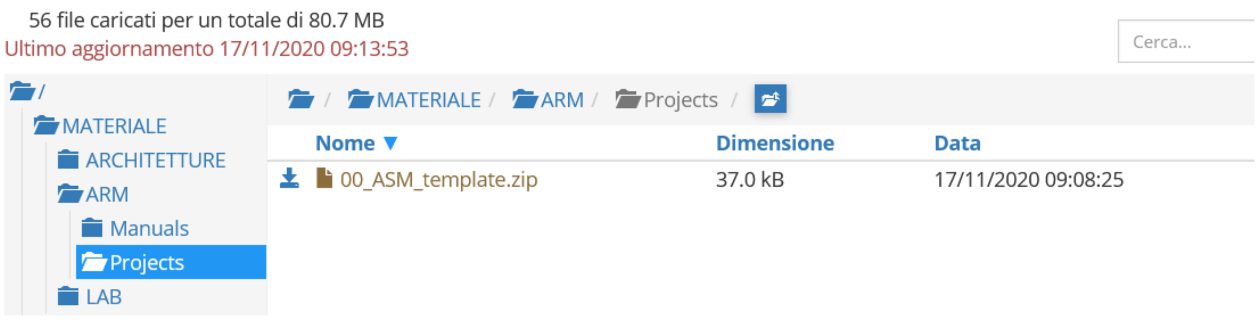
Architetture dei Sistemi di Elaborazione 02GOLOV Laboratory 6	Delivery date: 26/11/2020 Expected delivery of lab_06.zip must include: - Solutions of the exercises 1, 2 and 3 - this document compiled possibly in pdf format.

Starting from the ASM_template project (available on Portale della Didattica), solve the following exercises:


56 file caricati per un totale di 80.7 MB
Ultimo aggiornamento 17/11/2020 09:13:53

/ MATERIALE / ARM / Projects /

Nome ▼	Dimensione	Data
00_ASM_template.zip	37.0 kB	17/11/2020 09:08:25

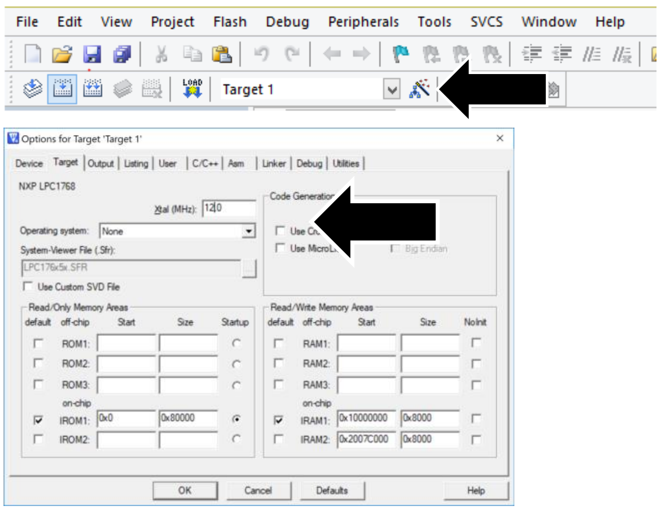
1) Write a program using the ARM assembly that performs the following operations:
   a. Sum R0 to R1 (R0+R1) and stores the result in R2
   b. Subtract R4 to R3 (R3-R4) and stores the result in R5
   c. Force, using the debug register window, a ***minimum set*** of specific values to be used in the program in order to **provoke as many flags (within the APSR) as possible** 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	
	R0	R1	R3	R4
Carry = 1	0x80000000	0x80000000	0x00000001	0x00000001
Carry = 0	0x60000000	0x20000000	0x60000000	0xE0000000
Overflow	0x60000000	0x20000000	0x60000000	0xE0000000
Negative	0x60000000	0x20000000	0x60000000	0xE0000000
Zero	0x80000000	0x80000000	0x00000001	0x00000001

2) Write two versions of a program that performs the following operations described below:
   a. Initialize registers R0 and R1 to signed values chosen by you;
   b. Compare the content of the two registers:
      - If the stored values are equal, then check if
        - such value perfectly divides by 3, then store value 0xA in R2
        - if not, store value 0x5 in R2
      - If R0 and R1 values differ, store in the R2 the minimum among R0 and R1

   The first version must be implemented resorting to a traditional assembly programming approach using conditional branches. Instead, the second version must be implemented using the conditional instructions execution approach. In order to compare exhaustively both versions

report the execution time in the table that follows. **To completely fill each column in the table, select the most appropriate data (i.e., the values to initialize R0 and R1) to explore all the program flow possibilities.**
<u>NOTE</u>, 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'*" .



NXP LPC1768

Xtal (MHz): 12.0

Operating system: None

System-Viewer File (.Sfr): LPC176x5x.SFR

Read/Only Memory Areas

default	off-chip	Start	Size	Startup
☐	ROM1:			○
☐	ROM2:			○
☐	ROM3:			○
	on-chip			
☑	IROM1:	0x0	0x80000	●
☐	IROM2:			○

Read/Write Memory Areas

default	off-chip	Start	Size	NoInit
☐	RAM1:			☐
☐	RAM2:			☐
☐	RAM3:			☐
	on-chip			
☑	IRAM1:	0x10000000	0x8000	☐
☐	IRAM2:	0x2007C000	0x8000	☐

CC=t1*cclk_frequency

Program flow possibilities	R0==R1 and %3==0	R0==R1 and %3!=0	R0!=R1 and R0>R1	R0!=R1 and R0<R1
Programming style	*Elapsed time depending on data*, *measured in clock cycles* [cc]			
Traditional	17	19	12	11
Conditional Execution	17	17	12	12

3) Write a program able to indicate whether a register contains a value that shows "even" or "odd" parity. The parity refers to the total number of 1-bits in a binary string. For example, the decimal number 4 is showing an **<u>odd parity</u>** (0100 ← a single 1-bit), while the decimal number 5 has an **<u>even parity</u>** (0101← two 1-bits).

Implement the ASM code that performs the following operations:

a. It determines whether the registers R0 and R1 are showing the same parity,
b. As a result, the values of R0 and R1 are updated as following:
    - If R0 and R1 have the same parity (both even or both odd): the program clears (to binary value 0b) the 8 Most Significant Bits – MSB - and sets (to value 1b) the 8 Least Significant Bits – LSB - of R0. **<u>All other bits must remain unchanged</u>**.
    - If R0 and R1 have different parity: the program copies in R1 the values of the flags.
c. Report code size (look for the .text section in the map file generated by Keil) and execution time (with 12MHz cclk) in the following table:

	Code size [Bytes]	Execution time [cc]	
		if both Odd or Even	Otherwise
Exercise 3) computation	100	21	21

ANY USEFUL COMMENT YOU WOULD LIKE TO ADD ABOUT YOUR SOLUTIONS:
Per il parity check dell’esercizio 3 ho deciso di usare un algoritmo di complessità logaritmica, anche se di più difficile comprensione. Esso si basa sul dividere il numero di partenza in due parti e fare lo xor della prima con la seconda, ricorsivamente, fino a raggiungere un unico bit, che indicherà 1 se il numero ha parity odd, 0 se ha parity even. L’equivalente C del codice sarebbe:

```
private static short computeParityMostEfficient(long no) {
        no ^= (no >>> 32);
        no ^= (no >>> 16);
        no ^= (no >>> 8);
        no ^= (no >>> 4);
        no ^= (no >>> 2);
        no ^= (no >>> 1);

        return (short) (no & 1);
    }
```