Architetture dei Sistemi di Elaborazione **Laboratory 6**	Delivery date: 12 November 2024 Expected delivery of lab_06.zip must include: - Solutions of the exercises 1, 2, 3 and 4 - 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:
   a. Initialize registers *R1*, *R2*, and *R3* to random signed values.
   b. Subtract *R2* to *R1* (*R2* – *R1*) and store the result in *R4*.
   c. Sum *R2* to *R3* (*R2* + *R3*) and store the result in *R5*.

   Using the debug log window, change the values of the written program in order to set the following flags to 1, one at a time and when possible:
   - carry
   - overflow
   - negative
   - zero

Report the selected values in the table below:

Updated flag	Hexadecimal representation of the obtained values			
	R2 – R1		R2 + R3	
	R2	R1	R2	R3
Carry = 1	3	2	3	0xFFFFFFFF
Carry = 0	7	8	7	8
Overflow	0x7FFFFFFF	0xFFFFFFFD	0x7FFFFFFF	1
Negative	2	5	6	0xFFFFFFF8
Zero	9	9	9	0xFFFFFFF7

Please explain the cases where it is **not** possible to force a **single** FLAG condition:
Nei calcoli con numeri signed, può verificarsi un overflow quando il risultato di un'operazione supera i limiti di rappresentazione del valore positivo o negativo. In particolare, quando il valore è troppo grande per essere rappresentato come positivo, si "ripiega" su un valore negativo a causa del sistema a complemento a due. In questi casi, il flag di overflow (V) viene attivato, poiché indica che l'operazione ha ecceduto i limiti, mentre anche il flag negativo (N) si attiva, poiché il risultato è negativo. Pertanto, ogni volta che si verifica un overflow con numeri signed, non è possibile isolare un singolo flag per indicare solo una condizione, poiché i flag di overflow e negativo sono entrambi influenzati.

2) Write a program that performs the following operations:
   a. Initialize registers *R6* and *R7* to random signed values.
   b. Compare the two registers:
      - If they differ, store in register *R8* the maximum among *R6* and *R7*.
      - Otherwise, perform a logical right shift of 1 on *R6* (is it equivalent to what?), then subtract this value from *R7* and store the result in *R4* (i.e., *R4 = R7 – (R6 >> 1)*).

   Considering a CPU clock frequency (clk) of *16 MHz*, report the number of clock cycles (cc) and the simulation time in milliseconds (ms) in the following table:

	R6 == R7 [cc]	R6 == R7 [ms]	R6 != R7 [cc]	R6 != R7 [ms]
Program 3	14,72	0,00092	22,72	0,00142

Note: you can change the CPU clock frequency by following the brief guide at the end of the document.

3) Write a program that calculates the leading zeros of a variable. Leading zeros are calculated by counting the zeros starting from the most significant bit and stopping at the first 1 encountered: for example, there are five leading zeros in *2_00000*101. The variable to be checked is in *R10*. After counting, if the number of leading zeros is odd, subtract *R11* from *R12*. If the number of leading zeros is even, add *R11* to *R12*. In both cases, the result is placed in *R13*.

   Implement ASM code that does the following:
   a. Determine whether the number of leading zeros of *R1* is odd or even (with conditional/test instructions!).
   b. The value of R13 is then calculated as follows:
      - If the leading zeros are even, *R13* is the sum of *R11* and *R12*.
      - Otherwise, *R13* is the subtraction of *R11* and *R12*.
   a) Assuming a *15 MHz* clk, report the code size and execution time in the following table:

Code size [Bytes]	Execution time [mS]	
	If the leading zeroes are even	Otherwise
0x000000EC−0x000000CC=0x20 = 32 BYTE	0,00108	0,00108

4) Create two optimized versions of program 4 (where possible!)
   a. Using conditional execution.
   b. Using conditional execution in IT block.

   Report and compare the execution Time

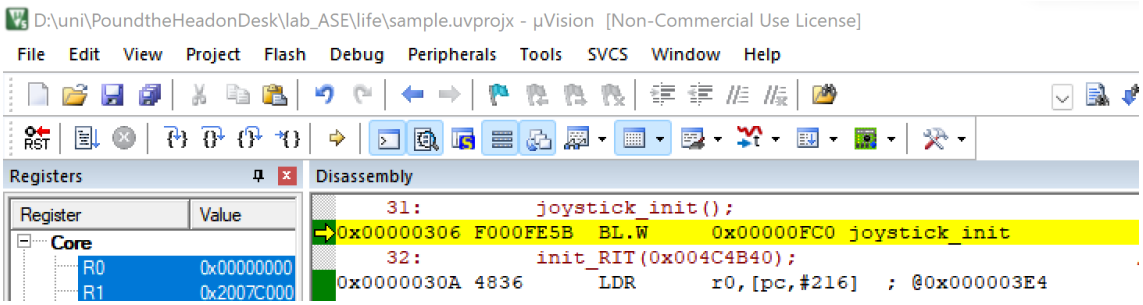
Program	Code size [Bytes]	Execution time [*mS*]	
		If the leading zeroes are even	Otherwise
Program 4 (baseline)	32 BYTE	0,00108	0,00108
Program 4.a	32 BYTE	0,00108	0,00108
Program 4.b	32 BYTE	0,00108	0,00108

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

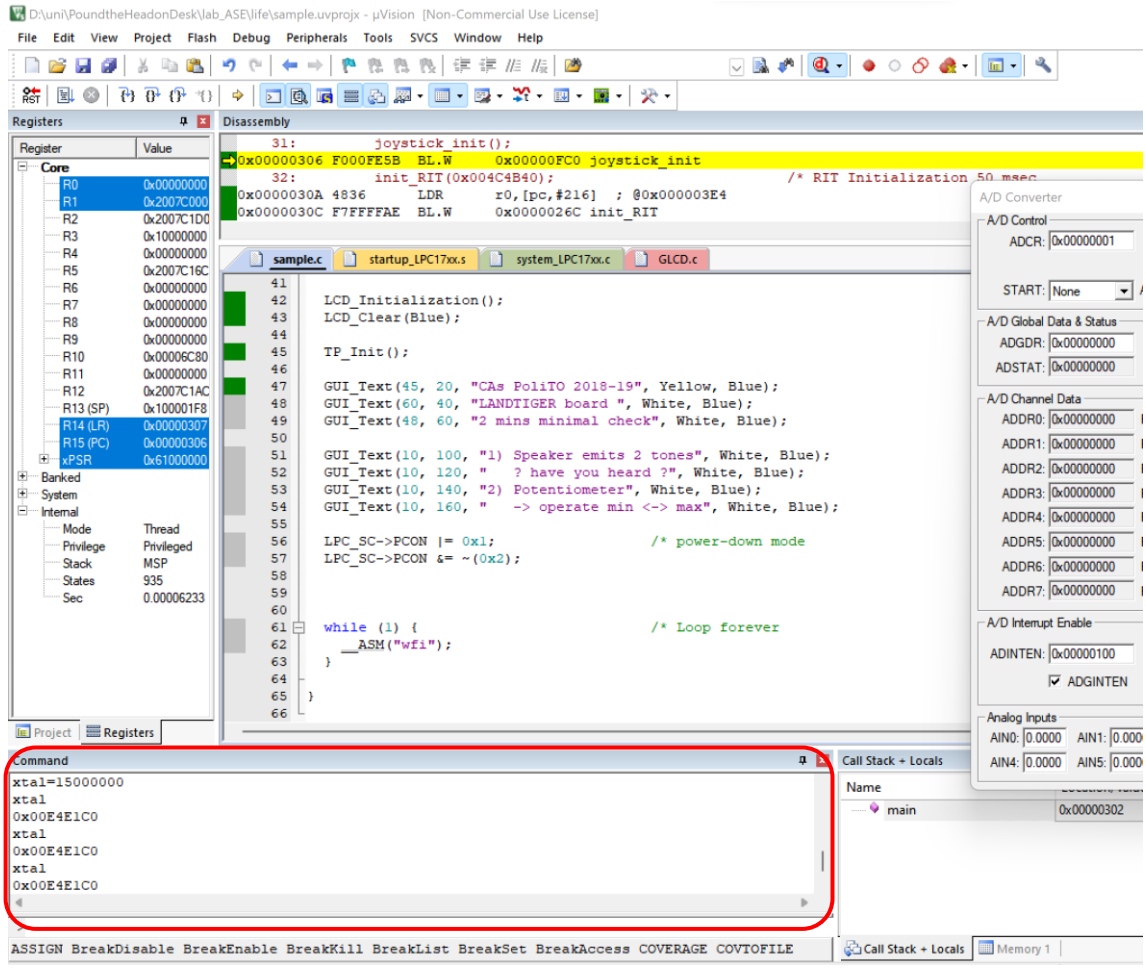
Non ci sono differenze tra l'esercizio 3 e il 4b, poiché in entrambi i casi utilizzo l'istruzione ITE già presente nel programma 3. Nel caso del programma 4a, il compilatore inserisce automaticamente l'istruzione ITE, rendendo il codice equivalente al precedente sia in termini di tempo di esecuzione, sia in termini di dimensione in byte.

How to set the CPU clock frequency in Keil

1) Launch the debug mode and activate the command console.



2) A window will appear:



You can type *xtal* to check its value. To change its value, make a routine assignment, i.e., *xtal=frequency*, keeping in mind that frequency is in Hz must be entered. To set a frequency of *15 MHz*, you must write as follows: *xtal=15000000*.