Computer Architectures	Delivery date: 13 November 2024
Laboratory 6	Expected delivery of lab_06.zip must include: - Solutions of the exercises 1, 2, 3 and 4 - this document compiled possibly in pdf format.

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:

	Hexadecimal representation of the obtained values			
Updated flag	R2 – R1		R2 + R3	
	R2	R1	R2	R3
Carry = 1	0x0000000F	0x00000000	0x0000000F	0xFFFFFFFF
Carry = 0	0x00000002	0x00000001	0x00000002	0x00000003
Overflow				
Negative	0x00000002	0x00000001	0x00000002	0x8FFFFFFF
Zero	0x00000000	0x00000000	0x00000000	0x00000000

We cannot have an overflow without a carry
We cannot have a zero without a carry, unless all the operands are already zero

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 2	11	0.00069	13	0.00081

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 *R8*.

   Implement ASM code that does the following:

   a. Determine whether the number of leading zeros of *R10* is odd or even (with conditional/test instructions!).

   b. The value of R8 is then calculated as follows:
      - If the leading zeros are even, *R8* is the sum of *R11* and *R12*.
      - Otherwise, *R8* 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 [*replace this with the proper time measurement unit*]	
	If the leading zeroes are even	Otherwise
32	0.00000067s	0.00000073s

4) Create two optimized versions of program 3 (where possible!)

   a. Using conditional execution.

   b. Using conditional execution in IT block.

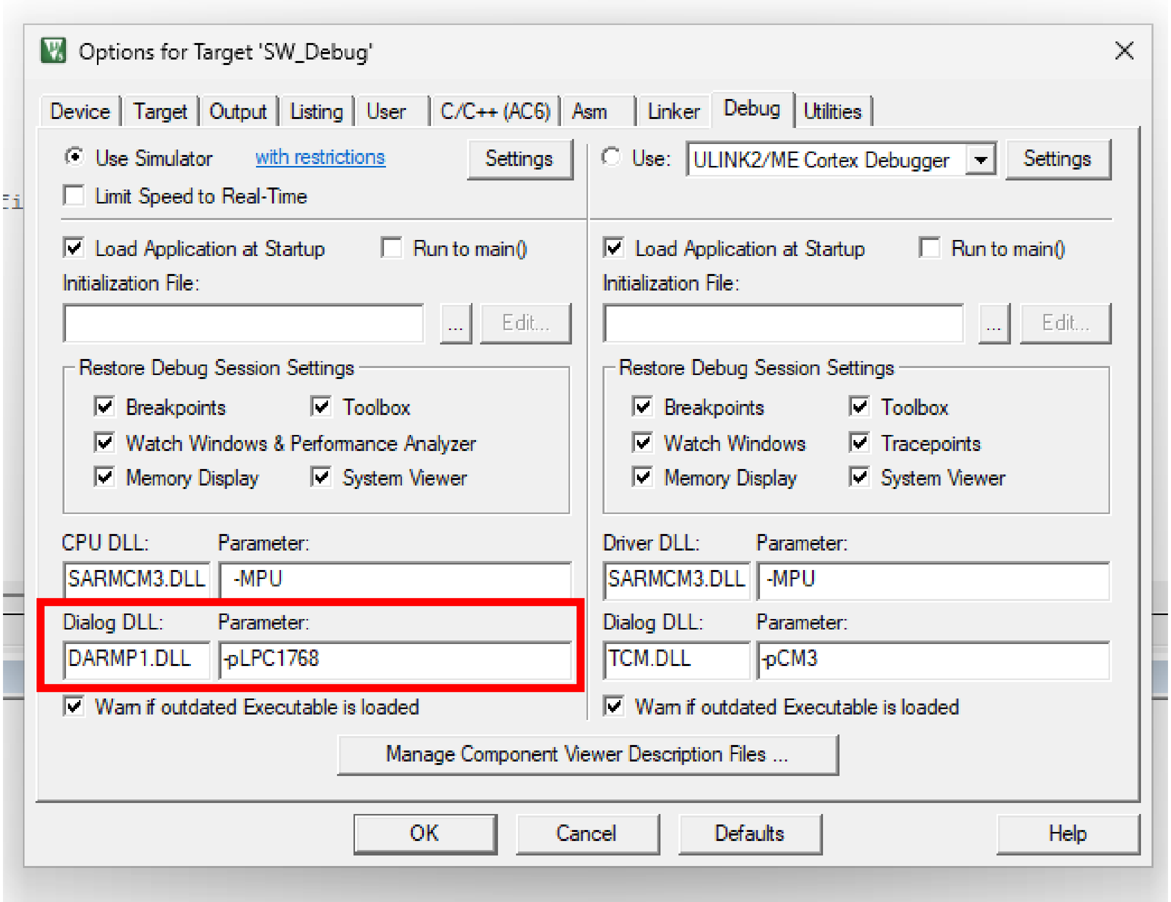
   Report and compare the execution Time

Program	Code size [Bytes]	Execution time [*replace this with the proper time measurement unit*]	
		If the leading zeroes are even	Otherwise
Program 4 (baseline)	32	0.00000067s	0.00000073s
Program 4.a	30	0.00000060s	0.00000060s
Program 4.b	30	0.00000060s	0.00000060s

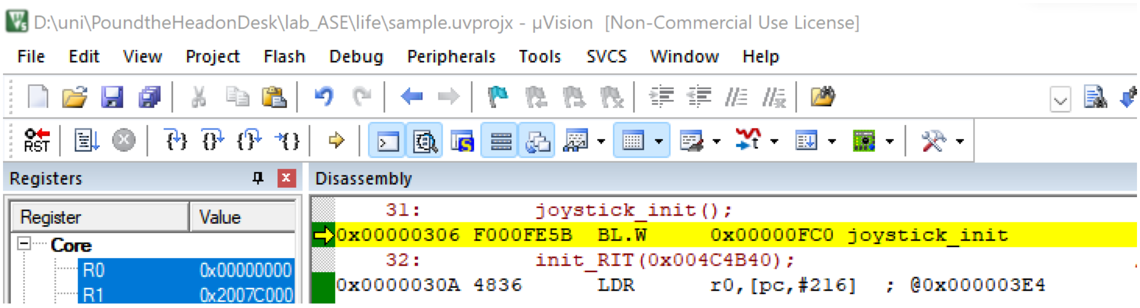
When writing the version program4.a, even if not explicitly writing the instruction ITE EQ, the assembler adds it automatically. This means that program4.a and program4.b are compiled to the same machine code, resulting in the same code size

How to set the CPU clock frequency in Keil

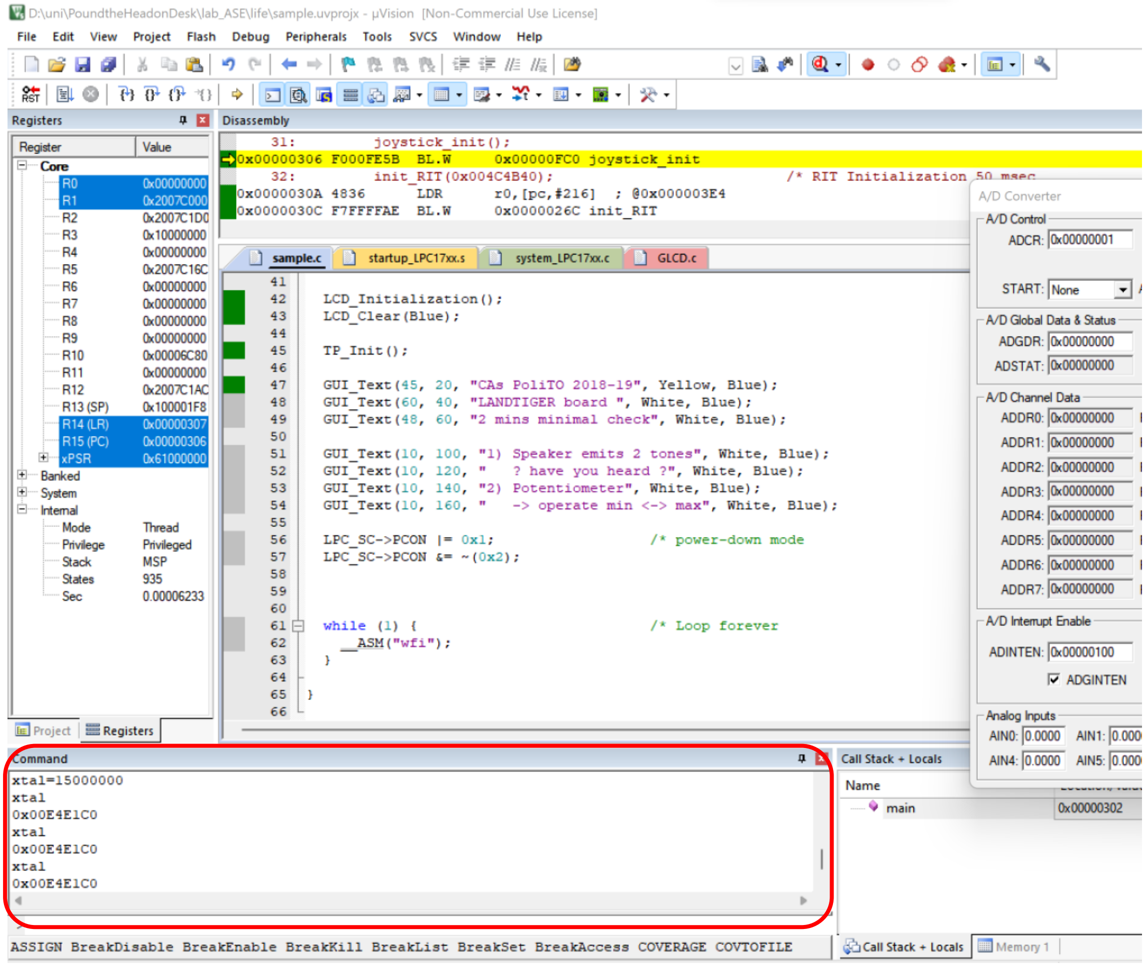
1) Verify that debug parameters ('Options for Target' window -> 'Debug' tab) are correctly set to match the target device (LPC1768). Modify the 'Dialog DLL' field to DARMP1.DLL and the 'Parameter' field to -pLPC1768, as shown in the image below



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



3) 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*.

4) After having set the frequency, reset the CPU by clicking on the RST button in the top left, above the 'Registers' window. The debug session will restart and the simulation will be executed using the desired clock frequency