Computer Architectures **Laboratory** **6** **REVALOR RICCARDO** **s339423**	Delivery date: 13 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.

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:

DISCLAIMER: if the SUBS raises a flag and the following ADDS sets it back to zero, I cont that flag as not set to 1 at the end of the execution of the program.

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

Please explain the cases where it is **not** possible to force a **single** FLAG condition:
Overflow (V) flag → If an overflow is generated (ex by summing 0xA1000000 and 0xB5000000), the V flag is set to 1 but the C flag is also set set to 1.

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? >> a division by 2, but ONLY for unsigned numbers [for signed ones I'd have to use an arithmetic right shift]), 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	15 (14.72)	0.00092	18 (17.28)	0.00108

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 R13 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 [μs]	
	If the leading zeroes are even	Otherwise
32	1	1.08

DISCLAIMER: for the code size I don't consider the whole startup.s file (in that case it would be 564 bytes)but just the size of the lines I've written in the Reset Handler portion of code.

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 [μs]	
		If the leading zeroes are even	Otherwise
Program 4 (baseline)	32	1	1.08
Program 4.a	30	0.92	0.92
Program 4.b	30	0.92	0.92

DISCLAIMER: for the code size I don't consider the whole startup.s file (in that case it would be 564 bytes)but just the size of the lines I've written in the Reset Handler portion of code.

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

In the base program I've implemented the logic through standard branch instructions. The two enhanced solution aim to **decrease both the code size and the execution time**. However, they are very similar as, as a matter of fact, when in Program4.a I write the conditional instructions, the compiler **autonomously places them inside an ITE block**, so at the end the first solution becomes equivalent to the second one, it's just less explicit (image below).
The advantage of these enhancements is that all branches are eliminated, resulting in a more linear execution flow. For example, the execution time is now independent of whether the leading zeros are even or odd. In theory, without using the ITE blocks (Program 4.a) each

conditional instruction would rely on its own independent and specific condition and the overall execution would be less efficient because the processor would have to test each condition independently. In this case, however, the code written is so simple and small that the difference would be minimal in terms of performance.

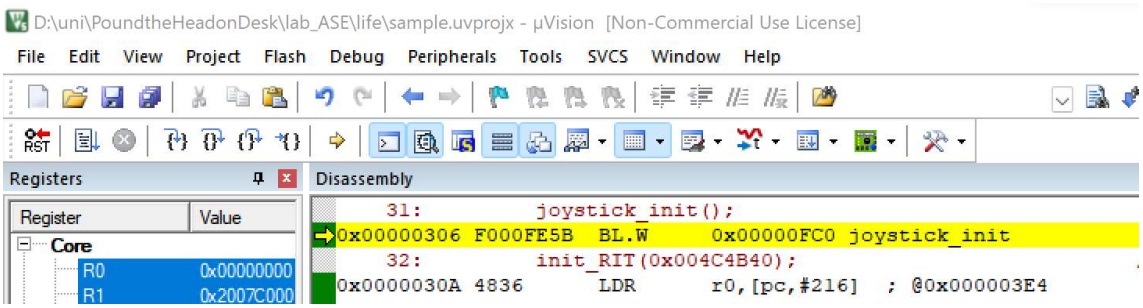
```
   134:                                 ;check if the last bit is 1 (odd) or 0 (even)
0x000000D8 FABAF08A  CLZ      r0,r10
   135:                                 tst r0, #1 ;it sets Z flag to 1 if so
   136:
   137:                                 ;case 1: last bit is 1 --> odd
0x000000DC F0100F01  TST      r0,#0x01
   138:                                 subne r8, r11, r12
   139:
   140:                                 ;case 2: last bit is 0 --> even
0x000000E0 BF14      ITE      NE
0x000000E2 EBAB080C  SUBNE    r8,r11,r12
   141:                                 addeq r8, r11, r12
0x000000E6 EB0B080C  ADDEQ    r8,r11,r12
   142:                 b .
   143:                 ENDP
   144:
```

startup_LPC17xx.s

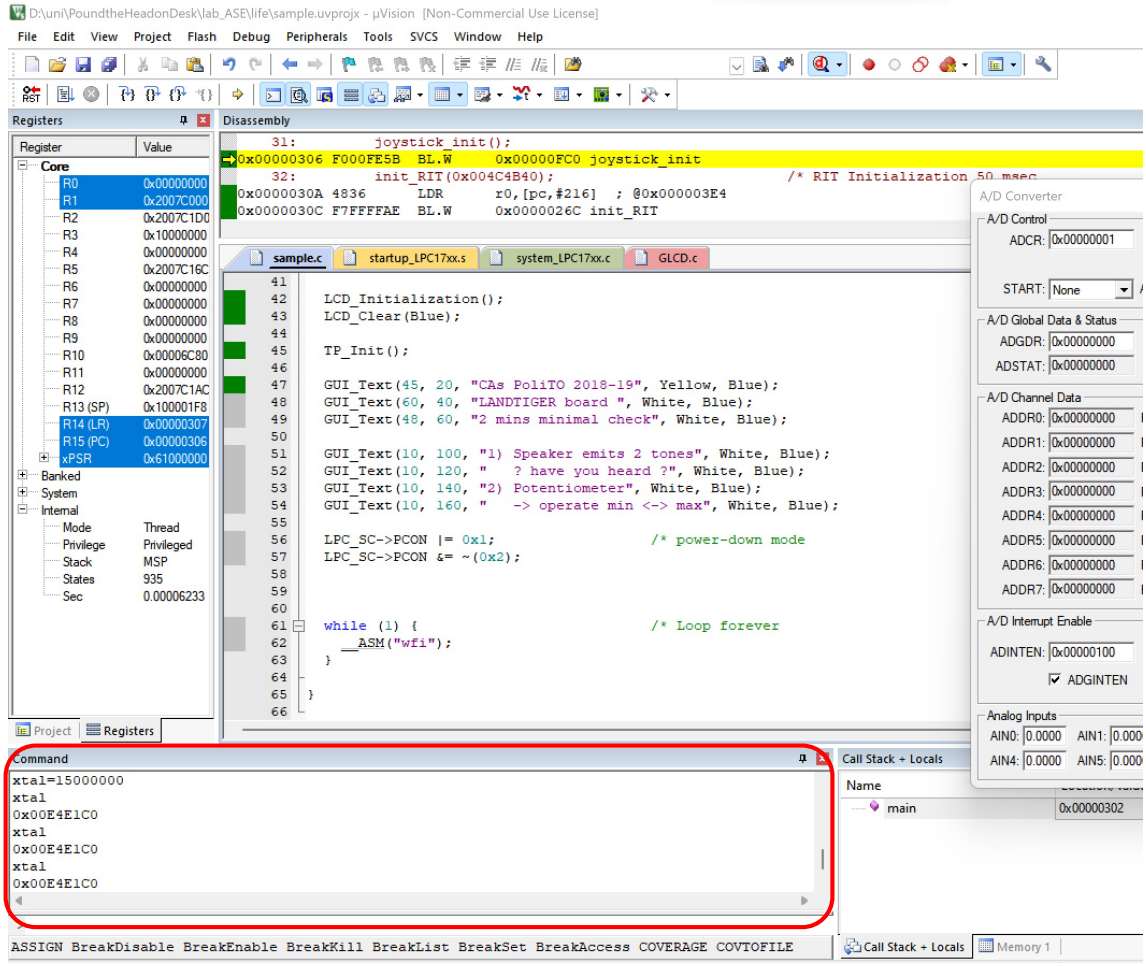
```
131                 clz r0, r10
132
133                 ;check if the number of leading zeros is odd or even
134                 ;check if the last bit is 1 (odd) or 0 (even)
135                 tst r0, #1 ;it sets Z flag to 1 if so
136
137                 ;case 1: last bit is 1 --> odd
138                 subne r8, r11, r12
139
140                 ;case 2: last bit is 0 --> even
141                 addeq r8, r11, r12
142                 b .
143                 ENDP
144
```

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*.