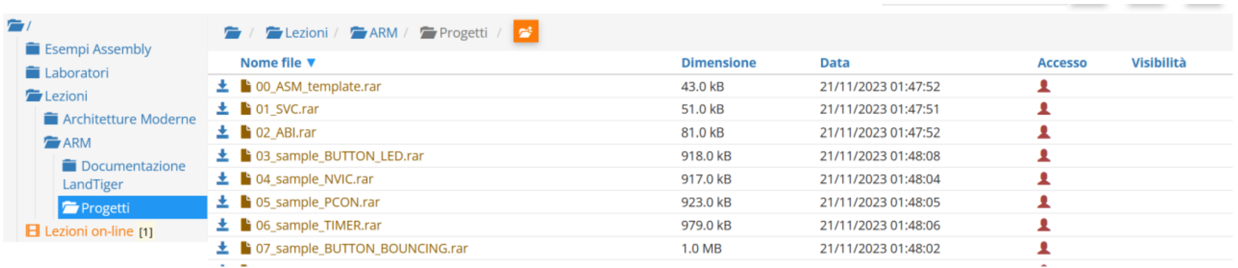
AArchitetture dei Sistemi di Elaborazione	Delivery date: 12 November 2024
Laboratory 6	Expected delivery of lab_06.zip must include: - Solutions of the exercises 1, 2, 3 - this document compiled possibly in pdf format.

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



/ / Lezioni / ARM / Progetti /

- /
  - Esempi Assembly
  - Laboratori
  - Lezioni
    - Architetture Moderne
    - ARM
      - Documentazione LandTiger
      - Progetti
  - Lezioni on-line [1]

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:
   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	2	1	2	FFFFFFFF
Carry = 0	1	2	1	1
Overflow	7FFFFFFF	FFFFFFFF	7FFFFFFF	1
Negative	1	2	FFFFFFFF	FFFFFFFF
Zero	1	1	1	FFFFFFFF

Please explain the cases where it is **not** possible to force a **single** FLAG condition: In ARM architecture, when an overflow occurs, the negative flag will also be set if the result is negative after accounting for the overflow. This typically happens in signed arithmetic operations, where an overflow can cause the result to wrap around and appear as a large negative number.

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 valBue 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	15	0.00092	19	0.00117

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 *R4*.

   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 R4 is then calculated as follows:
      - If the leading zeros are even, *R4* is the sum of *R11* and *R12*.
      - Otherwise, *R4* 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
26 bytes	0.00092 ms	0.00092 ms

4) Create two optimized versions of program 3 (where possible!)
   a. Using conditional execution.
   b. Using conditional execution in IT block.

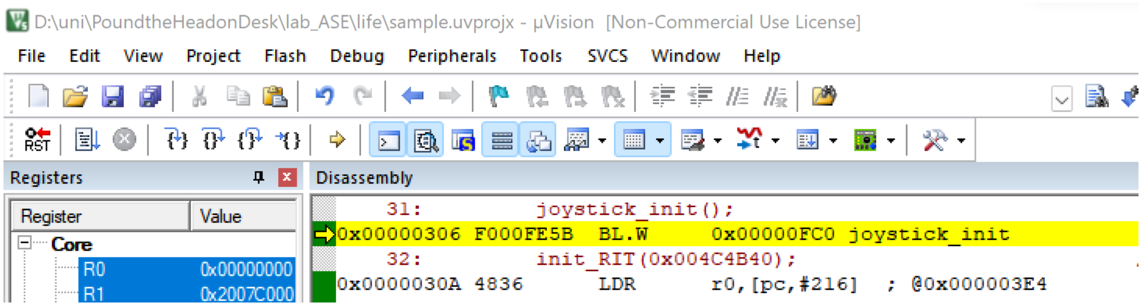
   Report and compare the execution Time

Program	Code size [Bytes]	Execution time [*replace this with the proper time measurement unit*]	
		If the leading zeroes are even	Otherwise
Program 3 (baseline)	26 bytes	0.00092 ms	0.00092 ms
Program 3.a	26 bytes	0.00092 ms	0.00092 ms
Program 3.b	26 bytes	0.00092 ms	0.00092 ms

ANY USEFUL COMMENT YOU WOULD LIKE TO ADD ABOUT YOUR SOLUTION:
In the baseline program for version 3, I included the IT conditional execution instruction. Consequently, both versions 3 and 3.b contain the same code. For version 3.b, I attempted to remove the ITE instruction; however, upon reviewing the disassembled output, I observed that the compiler reintroduced the ITE instruction automatically. As a result, versions 3, 3.a, and 3.b effectively execute the same code, with no functional differences.

How to set the CPU clock frequency in Keil

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

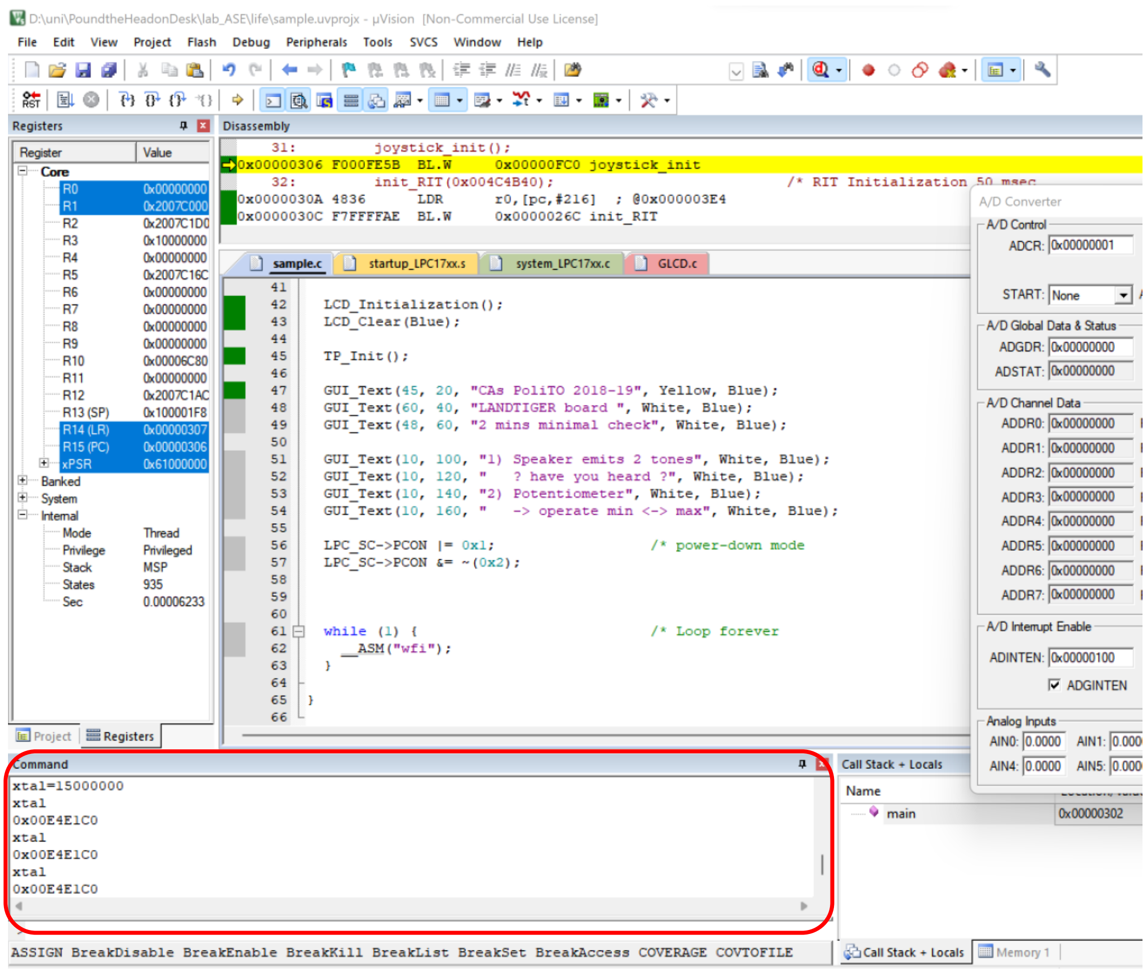

D:\uni\PoundtheHeadonDesk\lab_ASE\life\sample.uvprojx - µVision [Non-Commercial Use License]

File Edit View Project Flash Debug Peripherals Tools SVCS Window Help
Registers
Disassembly

Register	Value
Core	
R0	0x00000000
R1	0x2007C000

```
    31:         joystick_init();
0x00000306 F000FE5B  BL.W      0x00000FC0 joystick_init
    32:         init_RIT(0x004C4B40);
0x0000030A 4836      LDR       r0,[pc,#216]  ; @0x000003E4
```

2) A window will appear:


D:\uni\PoundtheHeadonDesk\lab_ASE\life\sample.uvprojx - µVision [Non-Commercial Use License]

Register	Value
Core	
R0	0x00000000
R1	0x2007C000
R2	0x2007C1D0
R3	0x10000000
R4	0x00000000
R5	0x2007C16C
R6	0x00000000
R7	0x00000000
R8	0x00000000
R9	0x00000000
R10	0x00006C80
R11	0x00000000
R12	0x2007C1AC
R13 (SP)	0x100001F8
R14 (LR)	0x00000307
R15 (PC)	0x00000306
xPSR	0x61000000
Banked	
System	
Internal	
Mode	Thread
Privilege	Privileged
Stack	MSP
States	935
Sec	0.00006233

```
    31:         joystick_init();
0x00000306 F000FE5B  BL.W      0x00000FC0 joystick_init
    32:         init_RIT(0x004C4B40);                          /* RIT Initialization 50 msec
0x0000030A 4836      LDR       r0,[pc,#216]  ; @0x000003E4
0x0000030C F7FFFFAE  BL.W      0x0000026C init_RIT
```

```c

  LCD_Initialization();
  LCD_Clear(Blue);

  TP_Init();

  GUI_Text(45, 20, "CAs PoliTO 2018-19", Yellow, Blue);
  GUI_Text(60, 40, "LANDTIGER board ", White, Blue);
  GUI_Text(48, 60, "2 mins minimal check", White, Blue);

  GUI_Text(10, 100, "1) Speaker emits 2 tones", White, Blue);
  GUI_Text(10, 120, "   ? have you heard ?", White, Blue);
  GUI_Text(10, 140, "2) Potentiometer", White, Blue);
  GUI_Text(10, 160, "   -> operate min <-> max", White, Blue);

  LPC_SC->PCON |= 0x1;                          /* power-down mode
  LPC_SC->PCON &= ~(0x2);



  while (1) {                                   /* Loop forever
    __ASM("wfi");
  }

}

```

A/D Control	
ADCR:	0x00000001
START:	None
A/D Global Data & Status	
ADGDR:	0x00000000
ADSTAT:	0x00000000
A/D Channel Data	
ADDR0:	0x00000000
ADDR1:	0x00000000
ADDR2:	0x00000000
ADDR3:	0x00000000
ADDR4:	0x00000000
ADDR5:	0x00000000
ADDR6:	0x00000000
ADDR7:	0x00000000
A/D Interrupt Enable	
ADINTEN:	0x00000100
✓ ADGINTEN	
Analog Inputs	
AIN0:	0.0000
AIN1:	0.000
AIN4:	0.0000
AIN5:	0.000

```
xtal=15000000
xtal
0x00E4E1C0
xtal
0x00E4E1C0
xtal
0x00E4E1C0
```

```
ASSIGN BreakDisable BreakEnable BreakKill BreakList BreakSet BreakAccess COVERAGE COVTOFILE
```

Name	
main	0x00000302

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*.