Computer Architectures **02LSEOV**	Delivery date: November 6th 2024, 11.59 PM
Laboratory **5** **SILVIA BONENTI** **S339232**	Expected delivery of lab_06.zip must include: • The solutions of exercises 1, 2, 3, 4, 5, and 6.

This lab will begin to delve into ARM programming, which was introduced this week. During these labs, you will use the Keil µVision IDE, which helps develop embedded applications.

- ***Defining register names***
  - Rename register *r1* to single_value, *r2* to double_value, *r3* to triple_value, *r4* to quadruple_value and *r5* to quintuple_value. Assign some value to single_value.
  - By only using MOV and ADD, assign these values to the registers:

*double_value = single_value *2*
*triple_value = single_value *3*
*quadruple_value = single_value * 4*
*quintuple_value = single_value * 5*
Hint: exploit Inline Barrel Shifter with MOV.

- ***Addressing***
  - Allocate 26 bytes into a memory area DATA READWRITE, without initializing them.
  - Initialize *r0* and *r1* to 1.
  - Assign the Fibonacci sequence elements to registers from r2 to r12. For example:

r2 = r1 + r0
r3 = r2 + r1

  - Assign to r14 the address of the first byte of the memory area allocated before.
  - Using pre-indexed addressing, save the least significant byte of registers *r0 - r12*, incrementing *r14* at each assignment.
  - Using post-indexed addressing mode, save the least significant byte of registers *r12 - r0* (reverse order), incrementing *r14* at each assignment.
  - At the end, check that the content of the memory is the following:

01 01 02 03 05 08 0D 15 22 37 59 90 E9 E9 90 59 37 22 15 0D 08 05 03 02 01 01

- ***Readonly and readwrite data areas***

- Define the following constants in a readonly data area:

myConstants DCW 57721,56649, 15328, 60606, 51209, 8240, 24310, 42159

- Allocate 16 byte (4 word) in a readwrite data area:
- Write a program in ARM assembly language that sums the halfword values in the readonly area two by two (i.e., first value + second value, third value + fourth value, etc.). Then, the program stores the 4 results (expressed as words) in the readwrite data area.

- ***Implementation of UADD8 instruction***
  - Cortex-M4 instruction set contains the following instruction:

UADD8 <Rd>, <Rn>, <Rm>

  - *UADD8* sums corresponding bytes of Rn and Rm, storing the result in Rd. Example:

Rn = 0x 7A 30 45 8D
Rm = 0x C3 15 9E AA
Rd = 0x 3D 45 E3 37
Note: the carry between bytes in Rd is absent.

  - *UADD8* is not present in the Cortex-M3 instruction set. Write the equivalent instructions to execute *UADD8 r4, r0, r1* in a Cortex-M3.
  - *Hint: You can gradually get the bytes of a number by doing, for example, a shift operation. Alternatively, you can create a mask and do an AND operation (not yet explained in class) that allows you to cancel out the other bytes and leave only the ones you need to perform the operation.*
- ***Try another board***
  - Create a new project by selecting a card with the Cortex M4 core, for example NXP LPC4072. Execute the instruction *UADD8 r4, r0, r1* with *r0 = 0x 7A 30 45 8D* and *r1 = 0x C3 15 9E AA*. Verify that the content of *r4* is the same as computed in the previous exercise.
  - *Note: You can install the package of the NXP LPC4072 board from the Keil Pack Installer. You can also choose another Cortex M4 core board of your choice.*
- ***Assigning an immediate to a register***
  - Write at least 3 different pieces of code (each one with a single or multiple instructions/pseudo-instructions) to move the value *0x00F4C400* to register r1.
  - Add a comment to the following question in your solution: which is the lowest immediate that the assembler does not accept in a MOV instruction?
    THE LOWEST IMMEDIATE THAT THE ASSEMBLE DOES NOT ACCEPT IN A MOV INSTRUCTION IS 0x00010002 (65538_{10}), because its digits are not obtained by shifting left an 8 bit constant and it also doesn't comply with any of the allowed patterns