Architetture dei sistemi di elaborazione **02GOLOV** **Laboratory** **8**	Delivery date: Thursday 10/12 Expected delivery of lab_08.zip must include: - the source code (startup.s) for exercise 1, for exercise 2 startup.s and main.c; - this document compiled possibly in pdf format.

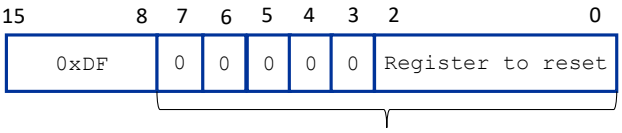
Solve the following problem by starting from the *startup.s* file in the ASM_Template project.

Exercise 1) Experiment the SVC instruction.

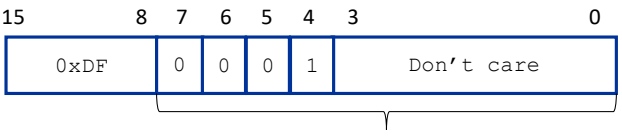
Write a testbench (i.e., a piece of code and data to fully test instruction functionalities) intended to test the following functionalities of a `SVC` instruction handler. Through this instruction it is requested to implement a RESET, a NOP and a MEMCMP functions. The MEMCMP function is used to compare two memory regions and it returns information about the execution. Assume that the SVC is called from a user routine with unprivileged access level.

In the handler of `SVC`, the following functionalities are implemented according to the `SVC` number:

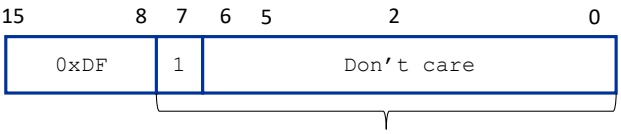
1. 0 to 7: RESET the content of register R?, where ? can assume values from 0 to 7 and it is the value specified in the SVC number.
2. 8 to 15 and >=128: NOP.
3. 64 to 127: the `SVC` call have to implement a MEMCMP operation, with the following input parameters and return values:
    - the 6 least significant bits of the SVC number indicates the number of bytes to be compared.
    - the initial addresses of the two areas to compare are 32-bit values passed through R0 and R1.
    - by again using R0, it returns:
        - 0 if all the bytes in the two areas are the same.
        - 1if the first not equal byte in the first area is greater than the second (in C language, `*ptr1+k>*ptr2+k`).
        - -1 in the other case (`*ptr1+k< *ptr2+k`).



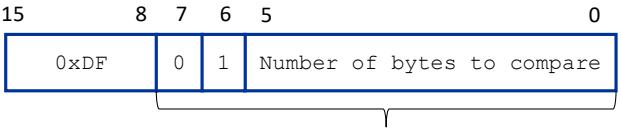
SVC number for Register RESET instruction (0 to 7)



SVC number for NOP (8 and 15)



SVC number for NOP (>= 128)



SVC number for MEMCMP (>= 64 and <=127)

NOTE: in your testbench, you should provide the most appropriate inputs values (SVC numbers, values in R0 and R1) to check that your code matches the requested behaviour. Be also aware that the SVC instruction must be called transparently to your code according to the ARM ABI.

Example: the following `SVC` invokes MEMCMP on two memory areas

```
LDR   R0, =StartAddressA
LDR   R1, =StartAddressB
SVC   0x48 ; 2_01001000 binary value of the SVC number
```

Q1: Describe how the stack structure is used by your project.
The stack is divided into two blocks of memory of 256 bytes each, for a total of 512 bytes. The first block, corresponding to the lowest memory addresses, is assigned to the user level and the stack pointer is the Process Stack Pointer (PSP). The second block, corresponding to the highest memory addresses, is assigned to the privileged level and the stack pointer is the Master Stack Pointer (MSP).
In the Interrupt Vector Table (IVT) at position 0 there is the address of the MSP. At reset, the system is in thread mode, at the privileged level and uses the MSP. After required initializations, it is possible to switch to an unprivileged level and use the PSP. A SuperVisor Call (SVC) causes the change of mode, level and Stack Pointer. In the SVC handler, the system switches to handler mode, at privileged level and uses the MSP. In the last instruction of the SVC handler, the previous system state (mode, level and Stack Pointer) is restored thanks to Link Register (LR).

Q2: What need to be changed in the SVC handler if the access level of the caller is privileged? In case report code chunk that solves this request (if any).
If the caller's access level is privileged, the caller's stack pointer is the Master Stack Pointer (MSP) instead of the Process Stack Pointer (PSP) which is used with an unprivileged access level.
At the beginning of the SVC handler there is a code chunk which, based on the value of the Link Register (LR), selects the stack pointer used by caller:

```
TST     LR, #4
MRSNE   R11, PSP        ; if LR == 0xFFFFFFFD, R11 = PSP
MRSEQ   R11, MSP        ; if LR == 0xFFFFFFF1 || LR == 0xFFFFFFF9, R11 = MSP
```

So, there is no need to change the code.

Q3: Is the encoding of the SVC numbers complete? Please comment.
No, SVC numbers from 16 to 63 are not mapped into any functionality. In these cases, SVC handler does nothing and therefore treats this range as a NOP functionality.

Exercise 2) Integrate ASM and C language functionalities

The following function, written in ASSEMBLY language, is invoked from a main C language function:

```
unsigned int variance(unsigned int* V, unsigned int n);
/* where n is the number of V elements */
```

The function returns alternatively:

- the integer truncation of the variance σ^2 of the values stored in V, according to the formula: $\sigma^2 = \frac{\sum_{i=0}^{n-1}(V_i-\mu)^2}{n}$, where μ is the integer mean of the values in V
- the value 0xFFFFFFFF if any significant error (identify the most critical ones) is encountered in the computation.

The main C language function takes care of declaring an unsigned integer vector called V composed of N elements (**N chosen by you**). At declaration time, the vector is statically filled by random values (**chosen by you)**.

Please fill the table below. For exercise 1 report the information considering the testbench you have developed. For exercise 2 replace **x** with the value chosen for N assume that the program completes without any error (thus, chose the most appropriate input values to return the variance).

F_{clk} = 12MHz	Execution time (clock cycles)	Code size	Data size
Exercise 1)	*62.17 µs × 12 MHz =* *746 cc*	*startup.o(.text) = 280 bytes*	*startup.o RO Data = 204 bytes* *startup.o RW Data = 0 bytes* *startup.o ZI Data = 512 bytes*
Exercise 2) with N=*10*	*64.25 µs × 12 MHz =* *771 cc*	*startup.o(.text) = 124 bytes* *main.o(.text.main) = 58 bytes*	*startup.o RO Data = 204 bytes* *main.o RO Data = 40 bytes* *startup.o RW Data = 0 bytes* *main.o RW Data = 0 bytes* *startup.o ZI Data = 512 bytes* *main.o ZI Data = 0 bytes*