0,Computer Architectures **02LSEOV 02LSEOQ [AA-LZ]**	Delivery date: Wednesday 12/12
Laboratory **7**	Expected delivery of lab_07.zip must include: - zipped project folder of the exercise 1 - this document compiled possibly in pdf format.

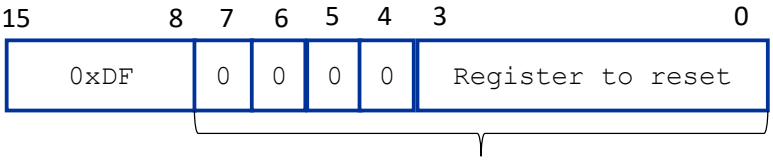
Solve the following problem by starting from the *template.s* file.

Exercise 1) Experiment the SVC instruction.

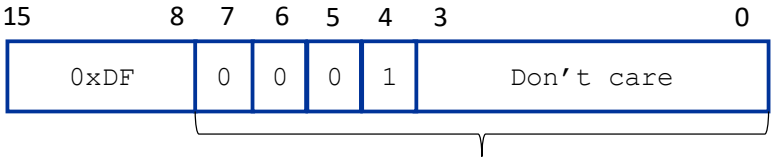
Write, compile and execute a code that invokes a `SVC` instruction when running a **user routine** with **unprivileged access level**. By means of invoking a SuperVisor Call, we want to implement a RESET, a NOP and a MEMCPY functions. The MEMCPY function is used to copy a block of data from a source address to a destination address and return information about the data transfer execution.

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

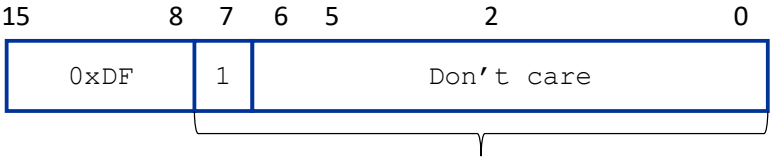
1. 0 to 7: RESET the content of register R**?**, where **?** can assume values from 0 to 7
2. 8 to 15 and >=128: NOP
3. 64 to 127: the `SVC` call have to implement a MEMCPY operation, with the following input parameters and return values:
    - the 6 least significant bits of the SVC number indicates the number of bytes to move
    - source and destination start addresses of the areas to copy are 32 bits values passed through stack
    - by again using the stack, it returns the number of transferred bytes



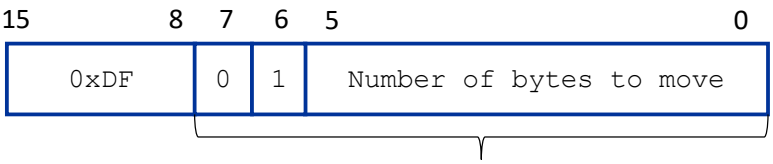
SVC number for Register RESET instruction (0 to 7)



SVC number for NOP (8 and 15)



SVC number for NOP (>= 128)



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

Example: the following SVC invokes MEMCPY from a given source to a destination

```
LDR  R0, SourceStartAddress
LDR  R1, DestinationStartAddress
PUSH R0
PUSH R1
SVC  0x48  ; 2_01001000 binary value of the SVC number
POP  R0
```

Q1: Describe how the stack structure is used by your project

(caso parametro compreso tra 64 e 127)
Prima della chiamata alla SVC viene fatta una push dei registri R1,R2,R3. I primi due registri contengono i parametri da passare all'handler ed R3 viene usato per riservare spazio nello stack per il risultato. In seguito alla chiamata alla SVC lo stack frame viene salvato nello stack PSP. A questo punto nell'handler viene recuperate l'indirizzo del PSP e tramite due load con offset 36 e 40 byte rispetto all'inizio dello stack PSP(per tenere conto della presenza dello stack frame) si possono recuperare I due parametri in ingresso. Il risultato si può invece scrivere nello stack con un offset rispetto allinizio del PSP di 44.

(caso parametro compreso tra 0 e 7)
All'inizio dell'handler SVC viene allocato uno spazio vuoto da 32 bit nello stack e viene salvato lo stato dei registri da r0 a r12 nello stack MSP.
Una volta recuperato il parametro della SVC, questo viene salvato nello spazio riservato all'inizio dello stack, in seguito si ripristinano tutti I valori dei registri.
Iinfine viene salvato il valore originale di R8(quello della funzione chiamante) nello stack.
Tramite una lettura nello stack con offset 4 viene recuperato il valore del parametro, il quale viene utilizzato per resettare il valore del registro corretto (da R1 a R7) ed infine viene ripristinato il valore originale di r8 con una pop dallo stack e viene incrementato di 4il vlore di SP per eliminare lo spazio di support utilizzato

Q2: What need to be changed in the SVC handler if the access level of the caller is privileged? Please

report code chunk that solves this request.
Se il chiamante lavora in modalità privilegiata, sia l'handler che il chiamante utilizzeranno lo stesso stack pointer(MSP)
Questo implica che nell'hadler dovranno essere modificate tutte le porzioni di codice che accedono allo stack tramite offset per tenere conto della presenza in cima allo stack dei registri salvati

Ad esempio:

```
STMFD SP!,{R8}
STMFD SP!, {R0-R12, LR} ;dimensione dati aggiuntivi=60B
.
.
.
LDR R8, [R9, #84]   ;(84= 60+24)
```

Occorrerà per cui aggiungere
Q3: Is the encoding of the SVC numbers complete? Please comment.

I valori da 16 a 63 non sono considerati

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

The function returns alternatively:

- the integer average value of the values stored in V, or
- value 0 if any significant error is encountered in the accumulation of the values.

The main C language function takes care of declare an unsigned integer vector called V and composed of N elements. At declaration time, the vector is statically filled by random values.

Please fill the table below.

F = 12MHz	Execution time (clock cycles)	Code size	Data size
Exercise 1)	291	300	1044
Exercise 2)	670	504	872