| Architetture dei Sistemi<br>Di Elaborazione | Delivery date:<br>14th December 2023 |
|---|---|
| **Laboratory<br>8** | Expected delivery of lab_08.zip must include:<br>- zipped project folders for Exercise1, Exercise2<br>- this lab track completed and converted to pdf format. |

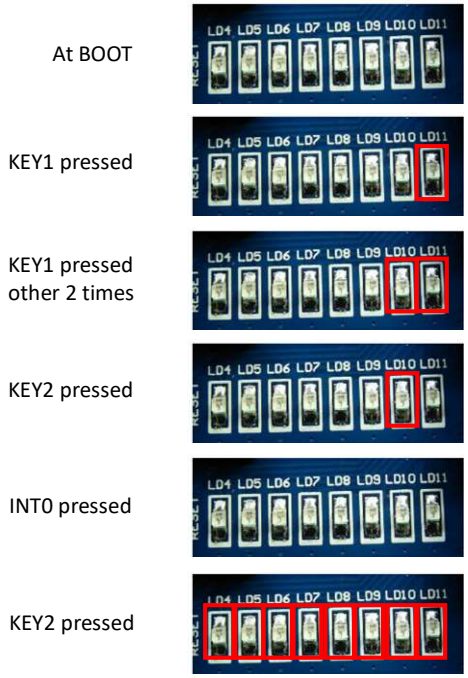
**Exercise 1)**

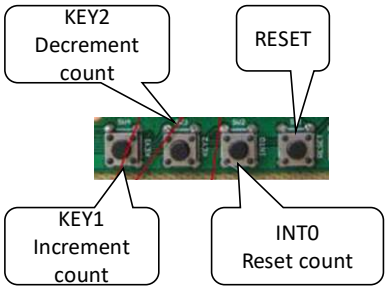
- Download the **template project** for Keil µVision "***03_sample_BUTTON_LED***" from the course material.

Implement an 8-bit "signed counter" by using LANDTIGER board; the software permits to use buttons to update a counting value which could be either positive or negative, and the LEDs to show the current value. By first using emulation capabilities (later, move your firmware on the board), please implement the following functionalities:

- increment a variable every time the button KEY1 is pressed
- decrement when KEY2 if pressed (in case, go to negative number)
- reset the count when INT0 is pressed.

LEDs are showing the current count in a binary, 2's complement representation.




**HINT**: It could be useful to use a global variable to keep the information about turned ON LEDs. For example, using a variable called "`char led_value`", already available in the project.

**Q1:** Do you observe any unexpected behaviour on the board with respect to SW emulation? Please describe.

Si verifica l'effetto "bouncing", ovvero il pulsante nonostante sia tenuto fermo crea delle leggere fluttuazioni del voltaggio che possono essere interpretate dal processore come nuovi interrupt producendo, quindi, risultati inaspettati. Nonostante il simulatore TigerBoard può simulare tale fenomeno, non tiene in considerato il caso in cui il pulsante, mentre sia mantenuto abbassato, si muova leggermente lungo l'asse ortogonale alla pressione, creando di conseguenza conteggi diversi in base al modo d'uso (tale

fenomeno dipende molto dal tipo di pulsante: precisione nella realizzazione, materiali usati, usura ed altro).

**Exercise 2)** Experiment the SVC instruction.

- Download the **template project** for Keil µVision "**01_SVC**" from the course material.
- You must execute the debug of the project on the LandTiger Board.

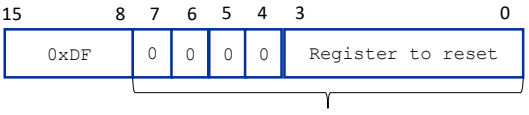
2.1) Write, compile, and execute a code that invokes an `SVC` instruction in the reset handler.

You must set the control to `user mode(unprivileged)`.

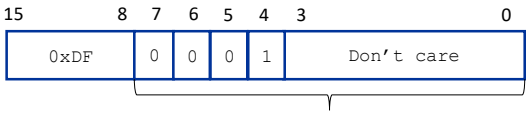
By means of invoking a SuperVisor Call, we want to implement a RESET, a NOP and a MEMCPY functions. The MEMCPY function is used to copy a block of data from a source address to a destination address and return information about the data transfer execution.

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

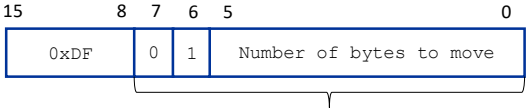
1. 0 to 7: RESET the content of register R?, where ? can assume values from 0 to 7
2. 8 to 15: NOP (no operation)
3. 64 to 127: the `SVC` call must implement a MEMCPY operation, with the following input parameters and return values:
    - the 6 least significant bits of the SVC number indicates the number 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 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.
La struttura dello stack è di tipo LIFO (last in, first out) ovvero l'ultimo elemento inserito nello stack tramite push sarà anche il primo ad essere recuperato tramite pop.
In aggiunta vi è una differenza di stack pointer usati tra il reset_handler e l'svc: nel primo caso impostando il controllo non privilegiato e modalità thread, lo stack pointer diventa il program_sp, nel secondo invece essendo sempre privilegiato diventa main_sp. Ciò permette di usare lo stack in modo indipendente e creare una barriera per l'utente ed impedirgli di modificare valori possibilmente importanti dentro lo stack del sistema.

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 livello d'accesso dell'utente diventa privilegiato possono esserci due possibilità:
1) utente privilegiato usa il program stack pointer (psp), allora l'unica modifica del programma risulta modificare i bit scritti nel control register:

```
MOV          R0, #2_10
MSR          CONTROL, R0
```

Ma nessuna modifica sarà necessaria nel svc

2) l'utente privilegiato usa il main stack pointer (msp), in questo caso allora il control register non ha bisogno di modifiche nei suoi bit (di base all'avvio del programma si consideri l'utente privilegiato in handler mode, quindi viene usato il msp).
Considerando il secondo caso l'svc diventa:

```
SVC_Handler     PROC
        EXPORT  SVC_Handler             [WEAK]

                STMFD SP!, {R0-R12, LR}
                MRS    R1, msp
                LDR R0, [R1, #80]     ;0x000000DC
                LDR R0, [R0,#-4]      ;0x000000D8
                BIC R0, #0xFF000000
                LSR R0, #16
                ; your code here

                […]

                STR     R5, [R1, #88]                   ;overwrite user last push

                […]
```

Dove in rosso ho evidenziato le modifiche essenziali per permettere di ottenere il parametro passato al svc considerando che 14 nuovi registri saranno presenti nello stack al momento del retrieve e il nuovo indirizzo relativo per permettere di sovrascrivere il valore a cui punterà lo stack quando ritornerà all'utente.

Q3: Is the encoding of the SVC numbers complete? Please comment.

Dal manuale Arm:
0 to $2^{24}$-1 (a 24-bit value) in an ARM instruction.
0-255 (an 8-bit value) in a Thumb instruction.

Nel nostro caso il template considera un range di 0-255 dal quale non vengono elaborate due partizioni: 16-63 e 128-255. Mentre nel primo caso si può considerare come un'estensione del range 8-15 in quanto anche lì non vengono azioni, potrebbe accadere in futuro di voler aggiungere delle funzionalità per quel range di parametri e bisognerebbe modificare leggermente il codice per permettere tale implementazione, la stessa cosa vale per il range finale da 128 a 255. In conclusione, non vi è un completo encoding del parametro passato a SVC però allo stesso tempo le possibili implementazioni future risultano abbastanza semplici da aggiungere considerando l'astrazione permessa eseguendo il programma come subroutine.