**Interrupts**

Name: Haoze He
NYU ID: hh2537

1. Which of the following statements are true? Mark all that apply.
   (a) An interrupt is always an urgent, high-priority task.
   (b) Using interrupts is always faster than polling.
   (c) System latency is always larger than interrupt latency.
   (d) Global variables used within an ISR should be declared volatile.
   (e) The interrupt vector table has to be placed in a specific location in memory.
   (f) On the ARM Cortex-M4, if two interrupts with priority numbers 0 and 1 occur simultaneously, the interrupt controller (permanently) clears the one numbered 1 and passes the one numbered 0 to the CPU.
   (g) Level 0 is the highest (most urgent) interrupt priority on the ARM Cortex-M4.
   (h) The startup code for the STM32F4 Discovery includes assembly code to save registers r0-r3 to the stack before entering an ISR.

| | True | False |
|---|---|---|
| (a) | | F |
| (b) | | F |
| (c) | T | |
| (d) | T | |
| (e) | T | |
| (f) | | F |
| (g) | T | |
| (h) | | F |

2. Can an interrupt service routine ever return a value? Can an interrupt service routine take arguments? Why or why not?

   (You should be able to answer this question based on your understanding of interrupts, without having to explicitly look up the answer.)

Ans: Interrupt can not return a value. Interrupt Service routine can not take arguments.
Why not. Our program will not call ISR. Hence ISR take no arguments & return no value.

3. Under what conditions may an interrupt service routine safely use an SPI bus that has multiple slaves on it?

   (You should be able to answer this question based on your understanding of interrupts and the SPI protocol, without having to explicitly look up the answer.)

It's not safe for ISR to use an SPI bus with multi-slaves in a transaction.
When SPI bus begins, it can not use anything until the main transaction is finished.
In addition, we could disable the interrupts and enable it after the transaction is finished. After the transaction is complete, we could run the ISR has SPI bus with multi-slaves.

4. Refer to this file (`startup_stm32f4xx.S`) and the STM32F4 Discovery Reference Manual to answer these questions (for the Discovery board and this particular startup file):

   (a) Write C code to define an ISR that's triggered by the USART1 peripheral. (You don't have to set up and enable the interrupt, just write the ISR.) The ISR should be a noop (i.e., do nothing).

According to the file, the function should be like:

```
void USART1_IRQHandler(){
// printf("Do Nothing Here")
}
```

(b) If you enable an interrupt in your C code but don't define the ISR, what code will execute when the interrupt is triggered?

According to the file, following code will be executed (Default handler):

```
112      .section  .text.Default_Handler,"ax",%progbits
113  Default_Handler:
114  Infinite_Loop:
115    b  Infinite_Loop
116    .size  Default_Handler, .-Default_Handler
117  /******************************************************************************
```

(c) Write an `EXTI15_10_IRQHandler` (just the ISR, you don't have to enable the interrupt). Assume you do *not* have any peripheral library functions, but you do have the following `define` statements:

```
#define     __IO    volatile

typedef struct
{
  __IO uint32_t IMR;    /*!< EXTI Interrupt mask register,            */
  __IO uint32_t EMR;    /*!< EXTI Event mask register,                */
  __IO uint32_t RTSR;   /*!< EXTI Rising trigger selection register,  */
  __IO uint32_t FTSR;   /*!< EXTI Falling trigger selection register, */
  __IO uint32_t SWIER;  /*!< EXTI Software interrupt event register,  */
  __IO uint32_t PR;     /*!< EXTI Pending register,                   */
```

```
} EXTI_TypeDef;

#define PERIPH_BASE          ((uint32_t)0x40000000)
#define APB2PERIPH_BASE      (PERIPH_BASE + 0x00010000)
#define EXTI_BASE            (APB2PERIPH_BASE + 0x3C00)
#define EXTI                 ((EXTI_TypeDef *) EXTI_BASE)

#define EXTI_Line10      ((uint32_t)0x00400)     /*!< External interrupt line 10 */
#define EXTI_Line11      ((uint32_t)0x00800)     /*!< External interrupt line 11 */
#define EXTI_Line12      ((uint32_t)0x01000)     /*!< External interrupt line 12 */
#define EXTI_Line13      ((uint32_t)0x02000)     /*!< External interrupt line 13 */
#define EXTI_Line14      ((uint32_t)0x04000)     /*!< External interrupt line 14 */
#define EXTI_Line15      ((uint32_t)0x08000)     /*!< External interrupt line 15 */
```

- Remember, you do *not* have any peripheral library functions.
- Your ISR should increment a (previously declared) global variable named `firstCounter` if the interrupt was triggered by line 11, and increment a (previously declared) global variable named `secondCounter` if the interrupt was triggered by line 12.
- It should also clear the pending register for the line in both cases.
- To test if an interrupt was triggered by a particular line, you need to check whether the pending register flag is set for that line AND whether interrupts are enabled (i.e. not masked) for that line.
- Refer to the section beginning on page 378 of the Technical Reference Manual for more information.
- Your ISR should also handle the case where both lines trigger an interrupt.

```
void EXTI15_10_IRQHandler(){
    if ((EXTI->IMR & EXTI_Line11) && (EXTI->PR & EXTI_Line11)){
        firstCounter = firstCounter + 1;
        EXTI->PR = EXTI_Line11;
    }

    if ((EXTI->IMR & EXTI_Line12) && (EXTI->PR & EXTI_Line12)){
        secondCounter = secondCounter + 1;
        EXTI->PR = EXTI_Line12;
    }
```