1. (10%) Interrupts can be completely asynchronous with the execution flow of the main program. They can be triggered at any point in time and in an unpredictable order. Is this statement true? Explain your answer if you don’t think this statement is true.

Yes, the statement is true. (P. 82 of Di Jasio)

2. (10%) The goal of interrupt management is to minimize interrupt latency. What is interrupt latency?

It is the time between the triggering event and the execution of the first instruction of the ISR. (P. 82)

3. (15%) An ISR is required to have a return type of `void`, takes no parameters, and cannot be called directly by main(). Explain the reason in sufficient details.

Since an ISR is triggered by an asynchronous event, there cannot be parameters or a return value because there is not any explicit function call from main() in the first place. (P. 83)

4. (25%) Given the following ISR code, write down the main() that corresponds to this ISR. Here `count` is defined as a global integer variable. Configure Timer2 to have a period of 15.

```c
#pragma  interrupt  InterruptHandler  ipl1  vector 0
void InterruptHandler(void)
{
    count++;
    mT2ClearIntFlag();
}

main()
{
    // 1. init timers
    PR2 = 15;
    T2CON = 0x8030;

    // 2. init interrupts
    mT2SetIntPriority(1);
    INTEnableSystemSingleVectoredInt();
    mT2IntEnable(1);

    // 3. main loop
    while(1);
}  (P. 91 of Di Jasio.)
```
5. (15%) In the lectures, we examined the nested ISR code, where one ISR is used to cover two interrupt devices coming in at different priority levels. Explain the C code constructs that are used to implement this nesting design. Show how the periods and priority levels for each device are configured to create the scenario of a running ISR being interrupted by another higher priority level interrupt.

The nesting ISR design was implemented with the if-else-if C code constructs. Two devices, i.e. Timer 2 and Timer 3, are configured such that an ISR is set up for Timer 2 at the lower priority level, while Timer 3 is set up at a higher priority level. Also, Timer 2 is configured to have a smaller period than Timer 3. This implementation ensures that the ISR for Timer 2 will be executed before a Timer 3 interrupt hits. Thus, Timer 2 ISR will “nest” the Timer 3 interrupt without having a separate ISR for Timer 3. (P. 95)

6. (5%) In the nested single-vector ISR application, is it possible to “embed” yet another even higher priority interrupt within the “Timer3” block, i.e. after the Timer3 interrupt occurs? Why or why not?

Yes, we can keep “embedding” another higher priority device as long as long as there is enough memory to grow the stack (each new interrupt will need additional stack space, except for IPL7).

7. (20%) Each interrupt source has 7 associated control bits, including Interrupt Enable, Interrupt Flag, Group-Priority Level, and Sub-Priority Level. Explain the purpose of each, and specify how many bits are allocated for each.

Interrupt Enable: 1 bit, when set, it allows the interrupt to be processed. Otherwise it is blocked.

Interrupt Flag: 1 bit, when set, it means an interrupt has happened/triggered. Must be cleared by the ISR after being set by the triggering event.

Group-Priority Level: 3 bits, used to encode priority levels for an interrupt device, e.g. Timer 2.

Sub-Priority Level: 2 bits, used to define 4 more sub-priority levels within the same Group-Priority Level.