Group A

Q1
What are differences between `iret` and `ret` instructions in terms of (a) use case (b) operation.

Q2
How do scancodes of key press and release differ?

Q3
Suppose we previously hooked into int 6h. Write code to unhook the new ISR. Address of old ISR is stored in 4 bytes at location `[origISR]`.

[4 + 1 + 5 marks]

Q1
(a) iret is to return from interrupt
ret is to return from subroutine
(b) iret will pop IP, CS and flags from stack, ret will pop only IP.

Q2
Scancodes are 1 byte long. **Press scancode has 0 in MSB, release has it 1**. Other seven bits are same.

Q3
```
mov ax, 0
mov es, ax
mov ax, [origISR]
mov bx, [orgISR+2]
cli
mov [es:6*4], ax
mov [es:6*4+2], bx
sti
```

Group B

Q1
Where will the vector for int 3 be found in memory? Give starting and ending physical addresses.

[2 + 2 + 3 + 1 + 2 marks]

Q2
Why flag register is pushed to stack upon an interrupt, but not on subroutine call?

Q3
Write code to disable IRQ 6 via interrupt mask register. PIC command port is 20h.

Q4
Find error in the following code if any

```
out 21h, bx
```

Q5
Why is it helpful to chain interrupt handlers?

Q1
Starting address 0000C (or 12 decimal)
Ending address 0000F (or 15)

Q2
Subroutines calls are programmer initiated, so they are already aware that flags will not be preserved after subroutine call
Interrupts are asynchronous, can be signalled at any time. When CPU has to suddenly jump to an interrupt handler, it must ensure that logic original program remains unaffected upon return.

Q3
Note, mask register is actually at port 21h. There was error in question, so code below is using port 20.

```
in al, 20h
or al, 01000000b  ; or 64 dec or 40 hex
out 20h, al
```

Q4
Only AL or AX can be source operand for out instruction.

Q5
So that we do not lose the default vendor-provided functionality of an interrupt. With chaining, we get a chance to write our own interrupt response and then forward control to the default handler.