You should attempt as many questions as you can before the examples class.
You can discuss the questions with other students, but remember that the exam may contain similar questions, so make sure you can answer them by yourself.

Make sure that you can answer all the exercises given in the lectures
If there is anything that you don’t completely understand in a previous example-sheet, try to sort it out now.

1. These questions are based on Lecture 13: Input/Output (1).
Here is part of a specification for a piece of code, used to poll a keyboard:

   The code for polling the system keyboard will access the peripheral via two memory mapped registers: the “data register” at address 0x4000000C and the “status register” at address 0x40000008.

   As there are several peripherals attached to the system, each peripheral has been allocated a different status register bit pattern. The status register test bit for this polled keyboard peripheral is specified as mapped to the third bit (i.e. bit 2, where bit 0 is the least significant bit) in the status register. This bit is set 1 to show that a character can be be read from the “data register”.

   a. Make sure you understand the requirements:
What is the hexadecimal test pattern used to test the peripheral’s status register?
Which single ARM code instruction should you use to test for this specific bit pattern in the peripheral’s status register?
What are the memory mapped addresses of the peripheral’s status register and data register?

   b. You are to implement a polling loop that will read the next character typed by a user on the keyboard peripheral, based on the following design:
repeatedly:
   read the status register
   and test the “ready” bit
   until the peripheral is ready,
   then read the data register

   Add the appropriate ARM instruction(s) to each line of the design above and so write a complete ARM coding solution, based on your design above.

Questions 2 and 3 are on the next page
2. These questions are based on Lecture 14: Input/Output (2).

In reality an interrupt handler might have to deal with several peripherals, and this code has to cope with devices being plugged in or removed.

A loop that can check a table of peripherals to see which caused the interrupt would be something like:

```assembly
ADRL R0, table ; load address of table of peripherals
loop LDR R1, [R0],#4 ; status register ADDRESS
    LDRB R1, [R1] ; load value from status register ADDRESS
    LDR R2, [R0],#4 ; pattern to test for
    TST R1, R2 ; test ready bit
    BEQ loop
```

Here is an example table of peripheral data, for 4 peripherals (0-3), each with a memory-mapped status register at a different address, and with a “ready” bit in various places within the status register:

```assembly
table DEFW 0x40000008 ; peripheral 0 - address of status register
    DEFW 0x80 ; peripheral 0 - ‘‘ready’’ bit-pattern
    DEFW 0x40000018 ; peripheral 1
    DEFW 0x80
    DEFW 0x4000006C ; peripheral 2
    DEFW 0x40
    DEFW 0x40000040 ; peripheral 3
    DEFW 0x10
```

a. Create a similar table, but this time for 4 peripherals (0-3) at addresses: 0x40000080, 0x40000088, 0x4000008F, and 0x400000FF, that use different “ready” bits within the status registers, so peripheral 0 uses bit 0 (the least significant bit), peripheral 1 uses bit 1, and so on.

b. Assuming that R2 contains the “ready” bit-pattern for peripheral 2 from the table in your answer to part (a) above, what is the result in R0 of obeying:

```assembly
MOV R1, #0xAC
AND R0, R1, R2
```

3. These questions are based on Lecture 16: Assemblers and Compilers (and lectures 1 to 12).

a) Which of the 5 phases of compilation is not found in an Assembler? Explain why.