These problems are done in advance and discussed in practice session on the 5th course week 28.11-2.12.2016.
Exercises are based on lectures 6-7 (web lectures 6-7).

1. Desimal, binary, hexadesimal
   a. What is the 16-bit binary representation of decimal value 2050? Give the result in bits and in hexadecimal notation. Can you just write it down?
   b. What is the 32-bit binary representation of decimal value 2050? Give the result in bits and in hexadecimal notation. Can you just write it down?
   c. What is the hexadecimal presentation of binary 0111010111? What is the decimal value for it?
   d. What is 0x1AB in 32-bit in binary? What is the decimal value for it? Can you just write it down?
   e. What is 0x1234ABCD 32 in binary? What is the decimal value for it?

2. Integer representation. Assume that integer value -13 is stored in memory at byte address 0x1230.
   What is the contents of bytes 0x1230-0x1233, when that number is stored in the following representations:
   a. 32 bit Big-Endian sign and magnitude
   b. 32 bit Big-Endian two's complement
   c. 16 bit Big-Endian two's complement
   d. 8 bit Big-Endian two's complement
   e. 16 bit Big-Endian, biased by 32767 (=2¹⁵-1)
   f. 32 bit Little-Endian two's complement
   g. 8 bit Little-Endian two's complement

3. In memory location 0x4320 there is 32-bit value 0x02700064. What value does it denote, if it represents
   a. 32 bit Big-Endian sign & magnitude
   b. 32 bit Little-Endian sign & magnitude
   c. 32 bit Big-Endian floating point?
   d. 32 bit Big-Endian 4 characters with UTF-8 coding?
   e. 32 bit Big-Endian ttk-91 machine instruction?
   f. 32 bit Big-Endian ARM machine instruction (extra)?

4. Floating point values.
   a. Give an example on a real number that can be represented exactly as a floating point number?
   b. Give an example of a real number, that can not be represented exactly a floating point? How large is the error?
   c. What decimal number is in binary 101101.1010?
   d. What is the IEEE floating point representation for decimal number 5.1?
      Is it exact or not? How about 5.1000007 and 5.10000007?
   e. Compute the expressions "(1.0666668-1.0666666) * 1.23456" and "1.0666668*1.23456 - 1.0666666*1.23456" in some programming language that supports 32-bit IEEE floating points. Place some dummy output in the middle of the 1st expression before the multiplication (e.g., see test.c), so that the compiler will not optimize your code too much. Why are the results different? Which one is correct? Why would using 64-bit IEEE floating points (C or Java data type double) remove the difference in this case? Will using 64-bit floating points remove the problem completely?

5. Floating point representation. Notice that the IEEE floating point standard has a special representation for very small numbers.
   a. What is the representation for very small numbers? What is bad with this representation?
   b. What is the smallest positive 32-bit IEEE floating point value? What is its representation? What is its value in decimal?
   c. What is the largest positive 32-bit IEEE floating point value? What is its representation? What is its value in decimal?
6. Hamming code.
   a. Show, how error correcting Hamming-code locates and corrects the error, when in 7-bit data "011 0100" the 3rd bit from left becomes erroneous: "010 0100". These 7 bits include both data and ECC (error correction code).
   b. How many wires (bits) are needed to protect a 32-bit data bus using error correcting Hamming code?
      (We want to transfer 32 bits of data in addition to the ECC bits)
   c. What if it was a 64-bit data bus?
   d. Why is Hamming code not a good solution to protect data transmissions (local area) networks?

7. Ordinary SEC (Single-Error-Correction) Hamming code will correct 1 bit errors but does not necessarily even detect 2 bit errors.
   a. Think of a situation where in 7-bit data (4 bit real data and 3 parity bits, even parity) "011 0100" two data bits (bits 1 and 2 from the right) change: "011 0100". Show how normal (SEC) Hamming code does not detect this. What does it cause?
   b. The situation can be corrected with an extra parity bit (bit 8). How would compute the value for that parity bit? Show the resulting 8-bit data value for the case in part a.
   c. Show how the 2-bit error in part a is not detected with this SEC-DED Hamming code. (Single-Error-Correction, Double-Error-Detection).
   d. Why does adding one bit help to detect all 2-bit errors? Consider also that one of the errors could be in bit 8.