Write your answers to the special response sheet you received (with your name and photograph). If you are using more than single sheet of paper for your answers, then mark each sheet with is number / total number of sheets will hand over.

Task 1

Microprocessor MOC 6502 is an 8-bit processor having a 16-bit address bus (memory address space), and an accumulator architecture. The accumulator register is named A in the 6502 assembler. Processor also features a FLAGS register including a Carry (C) flag with a standard meaning.

Processor 6502 has the following instructions:

- LDA `$address/#constant` (Load Accumulator) – reads an 8-bit value of the given 16-bit address ($ marked argument) *or* immediate value (# marked argument) into the accumulator.
- STA `$address` (Store Accumulator) – saves contents of accumulator into memory location identified by the given 16-bit address
- ADC `$address/#constant` (Add with Carry) – addition of two 8-bit numbers and the C flag (second number is given as the only instruction argument)
- SBC `$address/#constant` (Subtract with Carry) – subtraction of two 8-bit numbers and negation of the C flag (second number is given as the only instruction argument)
- CLC (Clear Carry) – set C to 0 (no expl. args.)
- SEC (Set Carry) – set C to 1 (no expl. args.)

Rewrite the following statement from Pascal into an equivalent sequence of 6502 machine code instructions (assembler). The E and F are 16-bit unsigned little-endian integer global variables located on the following addresses: E = E002h, F = F002h). All operations should be carried out in 16-bit unsigned arithmetics:

```
E := E + F + 16
```

Task 2

The Unicode encoding defines the following assignment of codes to given characters: code 158h for char 'Ř', code 52h for char 'R', code 65h for char 'e', code 70h for char 'p', code 61h for char 'a', and code 30Ch for combining character for diacritic caron (or haček) (the same diacritic as in the 'Ř' character).

a) Is in the Unicode any difference in meaning (interpretation) of a single character 158h and a character sequence 52h 30Ch? If so, what is the difference?

b) We would like to store text "Řepa" in little endian UTF-16 encoding to memory starting at address 0. Using a hexadecimal format write values of all bytes of memory (starting with byte at address 0) representing the text given above in the given encoding.

Task 3

Having the following number:

-256.625

write its value as a binary 32-bit floating point number. Mantissa is 23 least significant bits, and is normalized with hidden one. Mantissa is followed by an 8-bit exponent (stored in +127 bias format). The most significant bit represents a sign. Write the result as values of the 32 bits (ordering them left to right MSB first).

Task 4

An 8-bit microcontroller (MCU) Intel 8051 with accumulator architecture has the following instructions:

- `MOV A, #constant` – load of an 8-bit immediate value into the accumulator
- `ADD A, address` – addition (without carry) of an 8-bit value stored on given 8-bit address

Suppose the in the internal memory of our 8051 MCU is starting from address 0 stored the following program in 8051 machine code (every second line represents the instruction rewritten in 8051 assembler):

```
0x74 0x05
    MOV A, #5
0x25 0x03
    ADD A, 3
0x25 0x04
    ADD A, 4
```

If the computer starts executing the program (e.g. as a result of an unconditional jump to address 0) , decide, whether it is possible (from the information given here) to definitely state the final content of register A after execution of all the instructions of the program above … ? If so, what is A's value? **Important note:** MCU 8051 has a strict Harvard architecture.

Task 5

Three MCUs A, B, a C are connected to a single I^2C bus. All connected devices are using 100 kb/s transfer rate and 7-bit addressing. The MCUs are assigned the following addresses: A = 02h, B = 07h, C = A1h. When no one is accessing the bus, the A MCU decides to send two bytes: AA 0F to MCU B. Draw and describe graphs of voltage levels (and their logical values) on lines Serial Data (SDA) and Serial Clock (SCL) during the whole transfer (assume no errors happen during the whole transfer).

Task 6

Describe all typical states of a thread in a typical multithreading system. Include description of all typical transitions between these states, and define when such transitions happen.

Task 7

We have an IBM PC compatible computer with an Intel 80386DX CPU (32-bit microprocessor with 32-bit address and data buses, and a separate 16-bit I/O address space). The system's motherboard has 16-bit ISA slots only (16 data + 24 address lines + a line to differentiate memory and I/O bus transactions [M/IO]), and does not contain any high capacity storage device (like hard drive, or floppy disk drive) controller.

There are few additional devices inserted into the motherboard: 2 SIMM memory modules with total capacity of 2 MB of DRAM, a 16-bit VGA Trident 8900C graphics card with 512 kB of video RAM in the 1st ISA slot, and in the 3rd ISA slot a 16-bit 3Com 3c509C network card connected to a 10 Mbit Ethernet network via an UTP cable (and the computer's neighbor network infrastructure is configured to our needs).

When we turn the computer on, it boots into the MS-DOS 5.0 operating system by downloading its kernel from a neighbor network server. Describe all actions since pressing the ON button, until the system starts loading the MS-DOS kernel. Especially describe what instructions (of what programs?) is the CPU executing (and where it gets them) during the whole computer start-up sequence.

Task 8

We would like to program a wide variety of different applications, all of them having a graphical user interface (GUI). We plan to distribute binary executable for every such application both for Linux and Mac OS X, however each of these operating systems has a different GUI API (incompatible with each other), and our source programming language does not have any GUI programming built-in support. We would like to implement every application only once, thus we need its source level portability between Linux and Mac OS X. Describe a best suitable approach to solve the problem. Describe the whole compilation process of all the code necessary to get the final binary executables.

Task 9
(For up to 2 points)

We have a computer with a 32-bit memory address bus. The system has 64 kB of ROM (mapped continuously to the highest addresses in the address space), and 0.25 GB RAM (mapped continuously from address `0` upwards) installed. The only exception are addresses `1F1h` to `1F8h` that are mapped to a HDC ports via MM I/O mechanism. We are implementing part of the computer's firmware stored in the ROM mentioned. Your task is to implement function `Write` (and all other funcs/procs that you need as well) having the prototype specified below – its goal is to write a single sector (512 bytes) of data to a hard disk drive, where: `sector` argument defines a target sector number, `data` argument contains a pointer to 512 bytes of date in memory to be written to the target sector (it points to the first of the 512 bytes of data). Function must return to caller as soon as possible, and must not wait for actual write operation completion (i.e. it is an asynchronous I/O operation). The function returns: (a) `True`, if the write operation was successfully started, (b) `False`, if there is another write operation pending or communication with HDC failed:

```
type
      PByte = ^Byte;
function Write(sector : Word;
               data : PByte) : Boolean;
procedure InitHarddisk;
procedure SetInterruptVector(
               intVec : Integer;
               handlerRoutine : Pointer);
```

You can assume the rest of the firmware code will call your procedure `InitHarddisk` (see above), that you can use to implement any initialization of your HDC communication infrastructure necessary (e.g. initialization of any global variables). The firmware also implements procedures `SetInterruptVector`, that modifies the target address for interrupt vector `intVec` to an address of a handler routine passed in the `handlerRoutine` argument (pointer to the first instruction). You can assume that all interrupts are masked during whole execution of any such handler procedure.

The communication with the HDC is done via PIO mode. To start a write operation to a connected hard disk drive, you have to execute the following sequence of writes into controller's ports (each line begins with an address, where the relevant port is mapped):

1) `1F6h`: most significant 8 bits of sector number
2) `1F4h`: least significant 8 bits of sector number
3) `1F8h`: 8-bit command ID: command `WRITE SECTOR` = value `80h`

Next step is to wait until the controller signals it is ready to receive data – it indicates such state by clearing (setting to `0`) the 7th bit (counting from 0) (called BSY [Busy]) mapped to address `178h`. Then all the sector bytes are sequentially written into controller's data port mapped to address `171h`. Writing a byte into the data port causes the controller to set the BSY bit to `1` again. Next byte can be written into the data port only after controller signals its ready state by resetting the BSY bit back to `0`. Each time the controller resets the BSY to `0` it generates an interrupt request `14`. Please note that it is necessary to verify in the interrupt `14` handler, that the source of the interrupt is the HDC, as other devices can assert the same IRQ or a spurious interrupt might have happened (such interrupts can be ignored).

Assume the system is not using segmentation nor paging, and that the `Word` type is an unsigned 16-bit number.