1) Exercise specification

Write an assembly language device driver, which emulates a printer device. The device driver should accept text streams from a DOS system sent by a user by invoking standard file operations, such as a DOS `copy` command, and display those text streams on the screen. Also the device driver should work in two modes – *a standard mode*, where all the text sent by a DOS system is displayed on the screen without any modifications and, let's say *a capital mode*, where all the letters included in the accepted text stream are displayed as capital letters. Switching between those two modes can be done by introducing some predetermined two-characters long switching sequence, such as for example @+ for capital and @- for standard mode. Of course those character sequences should be embedded in the original text stream (original text file) and should not be displayed on the screen by the driver printer emulation procedure.

2) Device drivers

Device drivers are used in a DOS system (and also other operating systems) as a part of the operating system, which is responsible for system's communication and interaction with external devices. Most of the devices have common functions, such as the ability of writing a peace of information to a device (i.e writing a text to a printer) or reading information from a device (i.e reading a string of characters from keyboard). Thus DOS system assumes, that all devices offer similar, in some way abstract, functions (for example reading from or writing to a device). The role of the driver is to provide specific procedures which handle and implement those abstract DOS functions for a specific kind of the device. This enables identical treatment of different kinds of devices in the operating system which, from the user's (or programmer's) point of view, are seen and handled by the same standard functions. For example so called *character devices* like a printer or console (logical device consisting of the keyboard and display screen) are represented and handled as streams (like they were ordinary files), to which one can use standard file functions such as *fread* or *fwrite* and standard DOS commands (i.e *copy, echo* or *type*). In case of so called *block devices* they on the other hand are treated and seen logically by a user or a programmer as they were disks – for example a *pendrive* device.

2) Device driver operation

An installable device driver is a specially-formatted program that is loaded into memory when DOS boots. Installation of the device driver is performed via `DEVICE=` command contained in the CONFIG.SYS file, which is a text file containing various system configuration directives. From an assembly language programmer's point of view a device driver must be compiled as a single segment application containing all code and data of the program. It also must start with an 18-byte long structure called a *device header*, which informs DOS about the attributes of the device represented by a driver. Device driver needs also to have a purely binary structure, so after compilation and linking, one has to use EXE2BIN tool program to convert a device driver program from *.EXE format to this purely binary image. It's also worth changing converted in such way output file's name to *.SYS extension, which is a standard DOS extension for device driver files. The mentioned before device header contains such information as the (maximum 8-byte long) name of the device (which identifies the device in DOS system), it's attributes and the addresses (pointers) to two main procedures of the device driver, that is to so called *strategy* and *interrupt* routines – the names of those procedures are somewhat misleading. When some operation is requested by a user to act upon a device, DOS system first calls the strategy routine (using the pointer contained in the device header), which only job is to store DOS

request header pointer in a device drivers' local variables for later use. This DOS request header pointer is passed to a strategy routine via ES:BX registers in a segment:offset manner. When strategy routine ends, passing control to DOS, the system immediately calls driver's interrupt routine, which retrieves the address of the request header pointer from device driver's local variable and analyses DOS request header, figuring out what operation is requested by a user (directly by DOS) on a device. The possible abstract operations might be i.e. device initialization request and also i.e. read or write requests, and are identified through a requested function number contained in a proper byte of the device request header. Depending on the function requested by DOS form a device driver, the request header might contain additional information, such as for example a DOS buffer address and it's length containing characters to be written to a device. After identification of the requested function device driver program performs requested operations on the device making use of the additional information contained in the request header. Upon completion of the requested operation the device driver program prior to returning control to DOS, fills *device status word* in the device request header, which role is to inform DOS about the status of the last requested operation, that is weather the operation ended successfully or some error occurred. After passing control to DOS, the system checks this status word and decides how to handle eventual remaining requests to a device.

In such way the protocol of communication of the operating system with a specified device via it's device driver is organised in DOS. To sum up, devices (through their driver programs) communicate with DOS via request header, which contains abstract requests (identified by the predetermined function numbers) and also is used to store returned information about the status of the last requested operations. This allows to treat all possible physical devices as ordinary files or disks, depending on the attributes of the device contained in the device header of the device driver program, which also contains device's name, by which it will be later identified in the system. For example standard devices such as a printer or keyboard are identified by names such as LPT1 or CON, and a user/programmer can access them by using standard file operations like those presented below (for a console user):

```
copy file.txt LPT1          ;physically print contents of file.txt

copy CON file.txt           ;enter characters form the keyboard
                            ;(ending with Ctrl+Z+Enter sequence)
                            ;and save them physically in a file.txt

copy file.txt CON           ;display contents of the file.txt on the screen

echo This is a test > LPT1  ;redirect echo command output to LPT1 file
                            ;physically printing 'This is a test' on the printer
```

3) Exercises device driver specification requirements and hints

The device driver that is to be implemented has to be a character device (set in the attributes of the device driver header) with it's unique name i.e. 'VPRINTER'. It should implement only two functions i.e. the *initialization function* (00h request number) and *output function* (08h request number). The initialization function code and data should be placed at the very end of the device driver file, so after initialization (which is done only once, when DOS boots) it can be safely removed by DOS form operational memory. The initialization function can only display some greeting message indicating that the device driver has been successfully installed. Output function should display all characters send by DOS on the screen in the standard and capital modes (see Exercise specification in pt 1), depending on the recently used, predetermined switching sequence. Please have in mind that for a character device only one character at a time is sent by DOS to a device driver, so for a given two-

characters' long switching sequence some additional logic must be contained within the output function routine to handle switching between the two modes. Also all the other requests, such as for example possible input function request, have to be ignored by the driver setting appropriate error-bits in the device status word upon request completion – which in this case might be an 'Unknown function' error bit and 'Done' bit. Additionally, no mater weather the device is capable of handling the requested function or not the 'Done' bit must be always set in the device status word upon request completion.

Lastly, all the information that is needed to complete the exercise can be found in additional explanatory materials. Especially see the section 'General Index->Device Drivers' in the DOS technical help program HELP.EXE contained in the explanatory materials, which are included as a part of the exercises' site.