VRIJE UNIVERSITEIT BRUSSEL - ETRO

Computersystems: practicum

David Blinder, Jan Cornelis, Stijn Bettens, Tim Bruylants and Peter Schelkens

October 18, 2020

1 INTRODUCTION

The following exercise requires knowledge of the Video BIOS interrupt (`int 10h`) and the Keyboard BIOS interrupt (`int 16h`). It also allows experimenting with video memory access and color palette programming.

2 VIDEO BIOS INTERRUPT (INT 10H)

The video BIOS interrupt facilitates access to the video card settings. Among various functionalities, it allows switching to graphical video modes and programming the indexed colors (i.e. specifying a color index value that represents specific Red, Green and Blue (RGB) values). Most of our programs will use video mode 13h of `int 10h`, which is defined as a graphical video mode where the screen is set to a resolution of 320 pixels wide by 200 pixels high. This amounts to a total of 64000 pixels to specify one screen. Because mode 13h uses one 8-bit value (= one byte) per pixel the total amount of bytes for one screen is 64000. Using one-byte values, the video adapter is able to show 256 colors simultaneously.

The following code-snippet sets the graphics card to graphical mode of 320x200:

```
mov  ah, 0
mov  al, 13h
int  10h
```

Please note that setting `AH` and `AL` can also be done in one instruction by setting `AX`.

In order to draw pixels, a program writes bytes directly to the video memory. For mode 13h, the video memory is located in segment 0a000h (= effective address 0a0000h). So, in practice, accessing video memory is done by accessing memory, starting at 32-bit address 0a0000h and subsequently reading/writing bytes at the correct offsets to memory. The byte value specifies the color index, and the offset specifies the position on the screen of a pixel.

Video memory of mode 13h is organized in a raster-scan order; this means line-based, left to right. Thus, the upper-left pixel is at offset 0, and the lower-right pixel at offset 63999. The pixel at the right of the first pixel is at offset 1, and so on. The last pixel of the first line has offset 319. The first pixel on the second line has offset 320. At power-on, a default color palette is installed. It is a table in the video card memory that defines for each indexed color, the associated 6-bit red (R), green (G) and blue (B) values. From physics, we know that by adding R, G and B frequency light, it is possible to create a wide range of colors, including white. Int 10h provides the necessary functions to re-program the palette table with programmer specified RGB values. Using 6-bit RGB values the total number of representable colors is $64^3 = 2^{18} = 262144$.

```
DATASEG
palette db 0, 0, 0, 63, 63, 63 ; defines black and white

CODESEG
cld
mov   esi, offset palette   ; set the palette (DAC) address
mov   ecx, 2 * 3            ; set color 0 and 1 (2 indexes in total, 2 * 3
   bytes)
mov   dx, 03c8h             ; VGA DAC set port
mov   al, 0                 ; set start color index
out   dx, al
inc   dx
rep   outsb
```

3 Keyboard BIOS Interrupt (INT 16h)

This interrupt facilitates writing code that allows a program to respond to user keyboard interactions. Using this interrupt is illustrated with code-snippets:

```
mov  ah, 01h                ; function 01h (test key pressed)
int  16h                    ; call keyboard BIOS
jz   @@no_key_pressed       ; jump to some label if no key was pressed
mov  ah, 00h                ; function 00h (get key from buffer)
int  16h                    ; call keyboard BIOS
; process key code here (scancode in AH, ascii code in AL)
```

The difference between function 00h and 01h is that 01h does not block until a key is pressed, while function 00h stops the program and waits (indefinately) until the user presses a key. For our example code, it is OK to only use function 00h. However, **please do not use this interrupt for interactive gameplay in your video game**; it's not made for fast and reliable key press detection, but for typing text. Otherwise, gameplay will be stiff and delayed, and you can't register multiple simultaneous key presses. A better alternative for games can be found in the "EXAMPLES/KEYB" code example.

4 Exercises

(Please use the provided template "les4_template.zip")

Write a program that:

1. Changes the video output to 320x200 resolution with 8-bit indexed colors (mode 13h) by implementing `setVideoMode`. Exit when ESC is pressed by making `waitForSpecificKeystroke`. Try to turn a single pixel to white.
2. Create procedure `fillBackground` to fill the screen in a single color, provided as an argument. Use the `rep stosb/stosw/stosd` instruction(s) to fill the video memory.
3. Reprogram the palette to have 64 grayscale values, going from black (0,0,0) to white (64,64,64). Implement `gradientPalette` to create this palette in the array `palette` and transfer an arbitrary palette to the video card with `updateColourPalette` by means of `rep outsb`. Use this to draw a horizontal gradient filling the screen, going from black (left) to white (right).
4. Draw white rectangles via a procedure (not only squares) in the middle of the black screen background. The dimensions of the rectangle (width and height) are 32-bit values, passed as arguments to a procedure using the IDEAL syntax. That is, create a procedure `drawRectangle` that takes width and height. Draw only the rectangle outline, don't draw filled rectangles.

Please note:

1. Restore the original video mode after exiting the program (use function 0fh of int 10h). Default mode is typically mode 03h (text mode).
2. Upon setting mode 13h of int 10h, the video memory might be full of random bytes. If so, this will show garbage on the screen. So, in order to clean up this mess, your program should first set all pixels to black (or something less garbled) with `fillBackground`. This will also be useful to clear the screen between successive frames of a dynamic program, such as a game, simulation or video.