OBJECTIVE

Since graphics plays a very important role in modern computer application, it is important to know more information about its hardware and software operations. Despite the simplicity of the concepts required for this assignment, upon completion of this lab, you will be able to:

- understand what is and how to access a video buffer
- understand how colors are assigned and how the information stored in the video buffer is translated
- obtain system time (real time clock)
- detect keyboard clicks and decode the pressed key
- write user defined interrupt service routines (ISR)

BACKGROUND

1. Software Interrupts

Software interrupts are generated by programs, not by hardware. Since the first PCs with DOS and BIOS functions, all modern operating systems, including MS Vista, Linux, etc., still use software interrupts to access libraries, invoke systems functions, and connect with the operating system kernel.

Software interrupts are caused by a program issuing the software interrupt instruction to make the processor act as though it receives a hardware interrupt. This method is convenient for accessing OS services independently of any program location in memory. Although software interrupts may not be as critical as hardware interrupts, they are called frequently and must be efficiently implemented.

A software interrupt can be seen as an indirect call to a procedure. In this case, the data structure were the address of such procedure is stored is called interrupt vector table, or just interrupt vector. The interrupt vector is stored in the lowest 1024 bytes of system memory with 4 bytes (CS and IP) reserved per interrupt, for a total of 256 distinct interrupt vector entries. Upon examining this table, one will notice that there are some interrupts which are not used. Those vacancies are reserved for the users. In other words, INT 60H to INT 67H are reserved for the user defined interrupt services.

Because modifying an interrupt vector directly is not compliant with the specifications of any OS, the MSDOS also provides a safe way to consult and change the interrupt vector entries using interrupts numbers 25H and 35H. Here is what they look like:
(1) Set interrupt vector:

Calling registers:

<table>
<thead>
<tr>
<th>Register</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AH</td>
<td>25H</td>
</tr>
<tr>
<td>AL</td>
<td>interrupt number</td>
</tr>
<tr>
<td>DS</td>
<td>Code segment of the interrupt service routine</td>
</tr>
<tr>
<td>DX</td>
<td>Offset of the interrupt service routine</td>
</tr>
</tbody>
</table>

Return registers: None

(2) Get interrupt vector:

Calling registers:

<table>
<thead>
<tr>
<th>Register</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AH</td>
<td>25H</td>
</tr>
<tr>
<td>AL</td>
<td>interrupt number</td>
</tr>
</tbody>
</table>

Return registers:

<table>
<thead>
<tr>
<th>Register</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ES</td>
<td>Code segment of the interrupt service routine</td>
</tr>
<tr>
<td>BX</td>
<td>Offset of the interrupt service routine</td>
</tr>
</tbody>
</table>

For more, go to http://heim.ifi.uio.no/~stanisls/helppc/idx_interrupt.html.

II. Real Time Clock

Many applications require some sort of Idle Loop in order to “kill time” while the user is preparing to hit a key, move the mouse, etc… Most of the times, the OS itself will need to use such an idle loop for its internal operation (e.g. mouse movements).

For this assignment we will use a BIOS system call for obtaining the system time in the form of a counter that is incremented every 55ms. This system function is:

INT 1Ah, with calling register AH = 00h and return registers:
INT 1Ah, with calling register AH = 02h and return registers:

- AL: Midnight flag, 1 if 24 hours passed since reset
- CX: High order word of tick count
- DX: Low order word of tick count

CH: Hour (BCD)
CL: Minute (BCD)
DH: Second (BCD)
CF: 0, Clock is working
     1, No clock.

There is a DOS function call – service 2Ch – which can also yield the system time. However, we don’t use it in TSRs because of DOS reentrance considerations.

### III. The Keyboard Hit Detection

From Previous labs, we are familiar with keyboard character/string input. However, for some situations, we may need to detect if there is any input/keyboard hit, rather than wait until an input has been actually made. For that, we will use:

DOS INT 21, with calling register AH = 0B and on return:

- AL = 00 if no character available
- = FF if character available

### VI. Video Buffer

Video RAM is an element common to all video cards, whether this is programmed in text mode or graphics mode. While video RAM is organized differently for different video cards when this is set to graphics mode, the structure is virtually identical for all video cards when in text mode.

Video cards only need memory segments A and B, starting at segment addresses A000H and B000H, which requires only 64K. Monochrome video cards (i.e., MDA and Hercules) use video RAM in the range B000:0000 to B000:7FFF. Color Cards (i.e., CGA, EGA and VGA) use video RAM beginning at physical address B8000H or A0000H.
In DOS and Win98, whenever the card is in text mode (i.e. inside the command prompt for Win98), the screen is divided into 80 columns. For Win XP and up, one is able to change the number of columns and the number of rows. The top-left most character on the screen starts at RAM address 0B800:0. Each character uses 2 bytes, 16 bits, whose meaning is shown below:

<table>
<thead>
<tr>
<th>Background Color</th>
<th>Foreground Color</th>
<th>ASCII value of character</th>
</tr>
</thead>
<tbody>
<tr>
<td>First 4 bits</td>
<td>Second 4 bits</td>
<td>Last 8 bits</td>
</tr>
<tr>
<td>Brightness</td>
<td>Red</td>
<td>Green</td>
</tr>
</tbody>
</table>

Where, by “Last 8 bits” we mean the LSB (least significant bits).

**PRELAB**

1. Read this Lab carefully.
2. Design a Pseudo Code for the Lab

**REQUIREMENTS**

This experiment consists of two parts. First, write a set of 80x86 assembly language routines and a main program that can be installed as a TSR (Terminate and Stay Resident) program and that defines an user-defined ISR so it can be called by other programs. This user-defined interrupt service should contain the following functions:

- **INT 63H**
  Function 01H ==> clear screen

  Call register AH 01H

- **INT 64H**
  Function 01H ==> draw a rectangle

  Call register AH 03H
  AL color code (refer to table 2)
  CX X coordinate of the starting point
  DX Y coordinate of the starting point
When a user-defined interrupt service (or any ISR) is called, the DS (data segment) is unchanged at the entry point of the ISR. Therefore, it will point to the original data segment of the calling function. As a result, the DS must be manually initialized before access to any data inside the ISR can be made. Also, remember that an ISR is supposed to save (“PUSH”) all registers that it will modify during the course of its execution. These registers must be restored at exit.

For the second part of the lab, extend your original ISR to perform the following services:


b. Select a color on the egg ‘◊’ and the stone ‘×’.

c. Display the game field and the score on the top right corner of the screen.

d. Start the game by dropping multiple eggs and/or the stones by a random amount of rows, starting at a random column – you may use the time counter divided by the last keyboard input as a form of generating this “random” number.

e. The nest, represented by a yellow rectangle in the figure below, can be moved only horizontally with the keyboard.

f. If the player is able to catch the egg, he scores one point (+1), but if he accidentally catches the stone he loses one point (-1). The game’s speed would increase or decrease for every 3 points.

g. Game would stop when you get 10 or -10 points.

h. After the game ends you should still be able to see the final score on the screen, which the program asks if the player whether he wants to exit the game or play again.

i. Before exiting, the game should:
   i. clear the screen,
   ii. restore the original interrupt vector table and return to DOS.
This program should contain AT LEAST (remember that you must be able to “erase” these entities besides being able to draw them) the following subroutines:

a. MAIN: entry procedure.
b. AI: control egg and stone dropping. (A random drop is ok)
c. DRAWEGG: draw an egg.
d. DRAWSTONE: draw a stone.
e. DRAWNEST: draw the nest.

All BIOS function calls (macros) are included in a macro, BIOS.MAC, while the DOS function calls (macros) can be found in DOS.MAC.

LABORATORY

The interrupt service routine you write should become part of the Disk Operating System (DOS). In other words, it has to reside in the memory and become a Terminate and Stay Resident (TSR) program. INT 27H is recommended for this purpose:

- INT 27
  Calling registers:

  DX Offset of last byte plus 1 of the program to remain resident
  CS  Segment of PSP
The following program structure is suggested:

```assembly
code SEGMENT
ASSUME cs:code, ds:code, ss:code
ORG 100h
begin:
jmp initialize
; needed memory data here

; begin resident program here
int63 PROC FAR
.
. iret
int63 ENDP
.
int64 PROC FAR
.
. iret
int64 ENDP
.
int65 PROC FAR
.
. iret
int65 ENDP
; end resident program here
initialize:
ASSUME cs:code, ds:code, ss:code
.
.
lea dx, initialize
int 27h
code ENDS
code ENDS
END begin
```