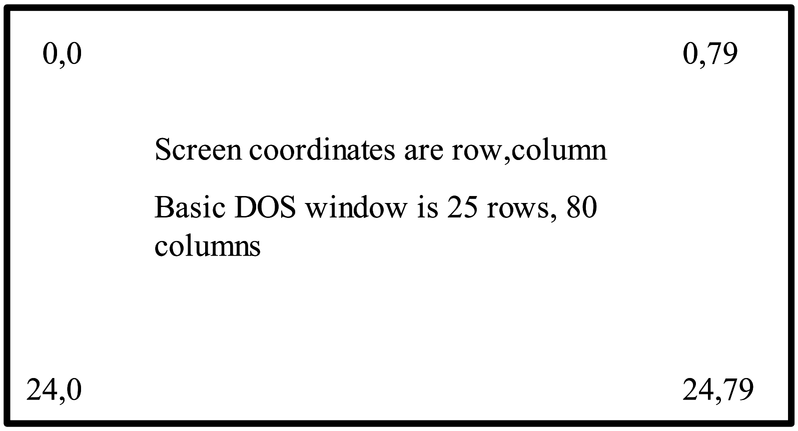
7. Text-based Graphics

Background

The basic character screen in a DOS-mode window can be thought of as an X,Y grid with 25 rows and 80 columns (see picture below).



Screen coordinates are row,column

Basic DOS window is 25 rows, 80 columns

The Basic Input Output System (BIOS) is a set of x86 subroutines stored in Read-Only Memory (ROM) that can be used by any operating system (DOS, Windows, Linux, etc) for low-level input/output to various devices. This lab will examine some BIOS routines for positioning the cursor and setting character attributes. The lab will also introduce you to the random number functions in the Irvine library.

Objectives:

Understand:

A. Cursor positioning and text mode attributes using the BIOS 10h software interrupt

B. Irvine procedures for pseudo random number generation

C. Use of a delay subroutine in a program

Pre-Lab

Read section 5.7 in the Irvine textbook about BIOS-level Video control for text modes.

Lab

A. Positioning the Cursor

The program below (*movcur.asm*) illustrates how to use the BIOS function for setting the position of the cursor.

```
       .model small
       .586
       .stack 100h
       .data
       row    db 12  ;;initially at row 12
       col    db 40  ;;initially at col 40
       .code
       extern Clrscr:proc

       main         proc
                    mov    ax,@data
                    mov    ds,ax
                    call   Clrscr  ;clear screen
                    call   setcur ; set cursor at row/col

       lp1:         mov    ah,7
                    int    21h    ; get character with no echo
                    cmp    al,'w'
                    jne    skip1
                    dec    row    ; move up
                    jmp    domove
       skip1:       cmp    al,'s'
                    jne    skip2
                    inc    row    ; move down
                    jmp    domove
       skip2:       cmp    al,'a'
                    jne    skip3
                    dec    col    ; move left
                    jmp    domove
       skip3:       cmp    al,'d'
                    jne    skip4
                    inc    col    ; move right
       skip4:       cmp    al,20h  ;space = exit
                    je     doexit
       domove:      call   setcur ; set cursor at new position
                    jmp    lp1
       doexit:      Mov    ax, 4c00h
                    Int    21h
       Main         endp

       setcur       proc
                    mov    ah,2   ;use BIOS 10h to set cursor
                    mov    dh,row ; row position
                    mov    dl,col ; column position
                    mov    bh,0
                    int    10h
                    ret
       setcur       endp
end main
```

How does this program work?

1. Two memory locations ‘row’ and ‘col’ are used to keep track of the cursor position. The *setcur* procedure calls the BIOS 10h, AH=2 cursor position function using the values stored in the locations ‘row’ and ‘col’. The program starts out by clearing the screen and then calls this procedure to position the cursor at the location initially specified by the row and column memory locations.
2. The program then enters a loop that reads a character from the keyboard using the DOS single character input function INT 21h, AH=7 which waits for a character and does not echo the character to the screen. If the character is ‘w’, the row value is decremented; for a ‘s’ the row value is incremented; for an ‘a’ the column value is decremented and for a ‘d’ the column value is incremented. The cursor is then positioned to the new row and column. A space character causes the loop to exit.

Lab Question 1: Assemble this program and execute it (you will need to link in the *irvine.lib* library). You should notice a problem when the cursor crosses a screen boundary (top, bottom, left or right) – the row and column values are not modified correctly when a boundary is crossed which causes erratic cursor movement. Modify the program so that the cursor wraps correctly around to the next boundary (e.g., if the cursor moves off the right edge it should appear at the left edge). Include the assembled listing of your program in your lab report.

B. Character Attributes

The BIOS display character function allows each displayed character to have an attribute (an 8-bit value) that specifies foreground and background color. The program below (*attr.asm*) has two nested loops which displays the letter ‘A’ for all possible attribute values.

```
      .model small
       .586
       .stack 100h
       .data
       row    db 0   ;;initially at row 0
       col    db 0   ;;initially at col 0
       attr    db 0    ;; initial attribute
       .code
       extern Clrscr:proc

       ;program will use BIOS 10h, function 9 to display the letter 'A' with all
possible attributes.
       main         proc
                    mov    ax,@data
                    mov    ds,ax
                    call   Clrscr  ;clear screen
       lp1:         xor    al,al
                    mov    col,al
       lp2:         call   setcur  ; set cursor at row/col
                    mov    al,'A'
                    mov    bl,attr
                    call   wchar
                    inc    attr   ; increment the attribute
                    inc    col
                    cmp    col,16 ; at column 16?
                    jne    lp2
                    inc    row
                    cmp    row, 16  ; at row 16?
                    jne    lp1
                    call   setcur
                    Mov    ax, 4c00h    ;exit
                    Int    21h
       Main         endp

       setcur       proc
                    mov    ah,2   ;use BIOS 10h to set cursor
                    mov    dh,row ; row position
                    mov    dl,col ; column position
                    mov    bh,0
                    int    10h
                    ret
       setcur       endp

;BIOS write character to page 0. Attribute in BL, char in AL
       wchar        proc
                    mov    ah,9
                    mov    bh,0
                    mov    cx,1   ;write only 1 time
                    int    10h
                    ret
       wchar        endp
end main
```

The *wchar* procedure uses the BIOS display character function to display the character/attribute pair passed in AL/BL. Be aware that the BIOS function displays the character at the current cursor position and does not change the cursor position (unlike the DOS display character function which advances the cursor). See section 5.7 in the Irvine textbook for more information on character attributes.

Lab Question 2: Assemble this program and execute it (you will need to link in the *irvine.lib* library). You may notice some discrepancies between what is documented in the Irvine book for attribute effects and what is displayed. For example, bit #7 is supposed to control blinking of the characters – do you get blinking characters? Be sure that you understand how character attributes work; you will need this for later in this lab. Include the assembled listing of this program in your lab report.

C. Pseudo Random Numbers and a Delay Subroutine

The program below (*rndchar.asm*) illustrates how to use the random number generation procedures in the Irvine library.

```
 .model small
.586
.stack 100h
.data
dtime  dw  500    ;; wait time in milliseconds

itimelow     dd 0      ;; used by delay routine
itimehigh    dd 0
.code
extern Crlf:proc
extern Randomize:proc, random_range:proc, Random32:proc

;; write a random digit with specified delay until any
;; character is entered

main         proc
             mov    ax,@data
             mov    ds,ax
             call   Crlf
             call   Randomize  ;; init random num gen
lp1:
             mov    eax,10
             call   Random_range  ;; gen random num 0 to 9
             add    al,30h      ;; convert to '0' to '9'
             mov    dl,al
             mov    ah,2
             int    21h    ;; display with DOS
             mov    cx,dtime   ;; get time to wait
             call   mywait
             ;; check if a key is pressed
             mov    ah,6
             mov    dl,0ffh
             int    21h
             jz     lp1    ; Zflag = 1 if no char, so loop

             Mov    ax, 4c00h    ;exit
             Int    21h

Main         endp

 CLKFREQ     EQU     800  ;; clock frequency in MHZ
TICS_MS      EQU CLKFREQ*1000

;; will delay # of milliseconds specified in CX. Register CX destroyed

mywait       proc

             push   ax
             push   dx
mywaitlp2:
             call   timget
             mov    itimelow,eax
             mov    itimehigh,edx
mywaitlp1:
             call   timget
             sub    eax,itimelow
             sbb    edx,itimehigh
       ;; edx:eax has delta time. Compare to TICS_MS
             sub    eax,tics_ms
             sbb    edx,0
             jc     mywaitlp1
             loop   mywaitlp2
             pop    dx
```

```
                    pop    ax
                    ret
      mywait        endp
;;; procedure that returns the Pentium+ 64 bit timer value
;;;  in EDX:EAX

      timget        proc
                    rdtsc    ;; read timestamp counter
                    ret
      timget        endp

end main
```

This program uses the Irvine pseudo random number procedures to choose a random digit between '0' and '9' and displays this character. The program continues doing this until a key is pressed on the keyboard.

How does this program work?

1. Two Irvine library procedures are used for pseudo-random number generation: *Randomize* and *Random_range*. The procedure *Randomize* needs to be called only once at the beginning of the program in order to initialize the 'seed' for the random number generator. The seed is a value that determines the sequence of numbers that will be generated – different seeds give different random sequences. The *Randomize* procedure uses the current time as the seed value. The term *pseudo-random* is used because if the seed value is known, then the random number sequence can be predicted. However, the random number sequence looks random to an external viewer who does not know the seed value. The *Random_range* procedure is used to return a random number between N-1 and 0 where N is passed in register EAX. *CAUTION* – the *Random_range* procedure will not generate very random sequences if the number range is too small. This is because many psuedo-random algorithms generate sequences that are not very random in the low order bits. To counter this, you can use Random32 (generate a 32-bit random number) and extract a group of bits from the middle of the 32-bit value and use this as your random number.
2. The *Random_range* procedure is called with EAX = 10 so that a random number between 0 and 9 is generated. The value 30h is added to the returned random number to generate the ASCII code for the digits '0' to '9' and this value is then displayed on the screen.

3. The *mywait* procedure is a procedure that will wait for the number of milliseconds passed in CX. This procedure uses a 64-bit hardware timer that is present in every Intel Pentium-compatible (586+) PC. The 64-bit counter is incremented on every clock cycle. The procedure *timget* accesses the 64-bit counter and returns it in EDX:EAX (EDX contains the high 32 bits, EAX contains the low 32 bits). The inner loop on the *mywait* procedure waits for the number of timer ticks that is equivalent to 1 ms (1 millisecond). It does this by first reading the timer and storing the 64-bit value in the locations *itimelow* (low 32 bits) and *itimehigh* (high 32 bits). The equate TICS_MS is the number of timer tics equivalent to 1 ms, and is computed as CLKFREQ*1000 where CLKFREQ is the clock frequency in Mhz . To understand why this works, consider that there are 1000 µs (microseconds) in 1 ms, and CLKFREQ number of clock cycles in 1 µs if the clock frequency is in Mhz (a 1 µs clock period is equal to a 1 Mhz clock frequency). The inner loop then continually reads the timer value, subtracting the original timer value from the new timer value. When this difference becomes greater than TICS_MS then 1 ms has passed. The outer loop executes the inner loop by the number of times specified in register CX.
4. The DOS 21h, AH=6 function is used to check for character input. This function does not wait for a character to be typed – if a character is available then it is returned in AL and the zero flag is cleared. If the zero flag is set (ZF = 1) upon return, then no character is available. The program loops until any key is pressed on the keyboard. This DOS function also does not echo the character to the screen..

Lab Question 3: Assemble this program and execute it (you will need to link in the *irvine.lib* library). You will need to modify the CLKFREQ parameter to match the clock frequency of the machine that you are on. To determine the clock frequency, use the 'Start→Search→Find Files/Folders" and find the program 'msinfo32.exe' (may also be available as "Programs→Accessories→SystemTools→SystemInfo). Under *System Summary* the *Processor* entry should give the clock frequency. Some older versions of Windows may not have 'msinfo32.exe' or it may not report the clock frequency. In this case, you can usually determine the clock frequency by rebooting your PC and watching the BIOS screen information - this will usually report the clock frequency and memory size of the PC during boot up.

A. The *main* program currently calls the *mywait* procedure with a value of 500 (wait for 500 ms or 1/2 second). Try changing this value to correspond to 10 seconds between characters. If you have a stopwatch available, check the accuracy of this delay. If the clock frequency is 1 GHz, how long would it take for this 64-bit timer to overflow? (reach 0xFFFFFFFFFFFFFFFF). Give your answer in the largest appropriate unit of years, days, hours, minutes, or seconds (if longer than a day, give the answer in days; if longer than a year, give your answer in years). SHOW YOUR CALCULATIONS!!!!

B. Modify the *main* program to call *mywait* with a delay value of '1' to see how fast characters can be sent to the screen. Be sure that you understand how this program works before proceeding to the rest of the lab. Include the assembled listing of this program in your lab report.

D. A Programming Task

Now that you understand the inner workings of the previous programming examples, use this knowledge to write a program that does the following:

A. The program should start by clearing the screen, and then place the cursor in the middle of the screen. Write a character at this location (you choose the character), and then randomly move the cursor one position either up, down, left, or right.

B. At the next position write a random character from the range 30h to FFh with a random attribute. Then move the cursor randomly one position again, except you cannot go backwards from the previous direction (if the previous move was up, then the next move can only be up again, left or right – it cannot go 'back' on itself). This means that after the first move, you can only move the cursor each time in one of three directions. You can overwrite previous characters. You must wrap correctly at screen boundaries.

C. Start out by having the cursor move with a 1 second delay between characters. Monitor the keyboard – if a 'w' is pressed; decrease the time between characters (speed it up). If a 's' is pressed, then increase the time between characters (slow it down). You can decide on how much to increase/decrease the delay time for each key press.

D. If the space bar is pressed, then exit the program.

Lab Question 4: Include the assembled listing of this program in your lab report and make sure that you have at least one comment for every two x86 instruction lines.

Lab Report

A. Describing What You Learned

Include the answers to all "**Lab Questions**" in your report.

B. Applying What You Learned

Demonstrate the programs you wrote for Lab Questions 1 and 4 to the TA.