

MANUAL TÉCNICO

201602782 – SERGIO FERNANDO OTZOY GONZALEZ
SEGUNDO PROYECTO DE LABORATORIO
ARQUITECTURA DE COMPUTADORES Y ENSAMBLADORES 1

1 TABLA DE CONTENIDO

2	Objetivos.....	2
2.1	Generales.....	2
2.2	Específicos.....	2
3	Alcance	2
4	Panorama general de la aplicación.....	3
5	Requisitos técnicos.....	3
6	Archivos incluidos	5
7	Uso de la memoria de video	6
8	Interrupción 1ah para manejar retrasos en hilos de ejecución.....	6
9	Funcionalidades de la aplicación	7
9.1	Sincronización con la memoria de video	7
9.2	Barrido hacia abajo de la pista de obstáculos	8
9.3	Función para obtener número de forma aleatoria	9
9.4	Ejemplo de uso de la interrupción 1ah para el manejo de hilos	9
9.5	Ordenamiento bubblesort.....	10

2 OBJETIVOS

2.1 GENERALES

1. Explicar partes selectas del código que compone el programa
2. Explicar el uso de la interrupción 1ah para manejar retrasos e hilos de ejecución
3. Explicar el uso de la memoria de video en el modo 13h

2.2 ESPECÍFICOS

1. Ofrecer una explicación sencilla sobre el código de ensamblador en el que está escrito el programa
2. Utilizar diagrama para explicar procedimientos y rutinas seleccionadas del programa
3. Explicar la administración de la memoria de video y el uso del doble buffer.

3 ALCANCE

Este manual técnico está dirigido a personas que posean un conocimiento avanzado sobre ensamblador, DOS y Microsoft® Macro Assembler (MASM).

El programa está escrito en MASM. Se utiliza DOSBox como entorno de pruebas. El bus de datos, el tamaño y uso de registros/segmentos y la sintaxis utilizada está basada y limitada por la arquitectura del procesador Intel® 80386 (32 bytes).

Se utiliza la versión 6.1 de MASM. No se utiliza ninguna librería. Todo el código está escrito con instrucciones de ensamblador.

4 PANORAMA GENERAL DE LA APLICACIÓN

La aplicación consta de dos módulos, un módulo de usuario y otro de administrador. Desde el módulo de usuario se puede acceder al juego y cargar un archivo para modificar los parámetros del juego. El objetivo del juego se explica en el manual de usuario.

Desde el módulo de administrador se es capaz de acceder a métricas tales como el top de punto y tiempo transcurrido en el juego. Podrá representar esos resultados en una gráfica de barras y visualizar una animación con sonido que ordene los resultados de forma ascendente o descendente.

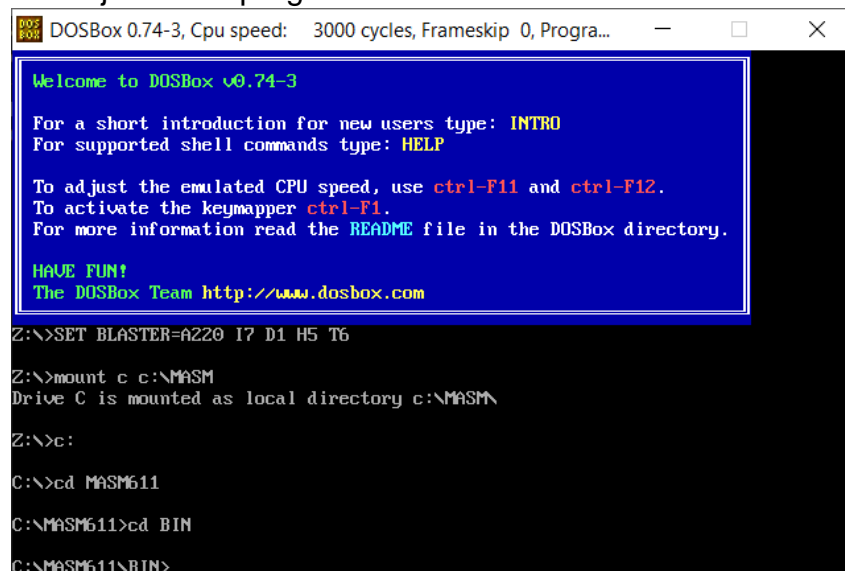
La aplicación comienza con el login, desde ahí se puede acceder al módulo de usuario o administrador o registrar un nuevo usuario.

5 REQUISITOS TÉCNICOS

Para el correcto funcionamiento y ejecución de la aplicación es necesario poseer el programa DOSBox instalado en el ordenador y tenerlos debidamente configurado para compilar código ensamblador o ejecutar el archivo .exe.

Para compilar el código ensamblador es necesario tener instalado MASM v6.1:

1. Deberá ejecutar el programa DOSBox



```
DOSBox 0.74-3, Cpu speed: 3000 cycles, Frameskip 0, Progra...
Welcome to DOSBox v0.74-3
For a short introduction for new users type: INTRO
For supported shell commands type: HELP
To adjust the emulated CPU speed, use ctrl-F11 and ctrl-F12.
To activate the keymapper ctrl-F1.
For more information read the README file in the DOSBox directory.
HAVE FUN!
The DOSBox Team http://www.dosbox.com

Z:\>SET BLASTER=A220 I7 D1 H5 T6

Z:\>mount c c:\MASM
Drive C is mounted as local directory c:\MASM\

Z:\>c:

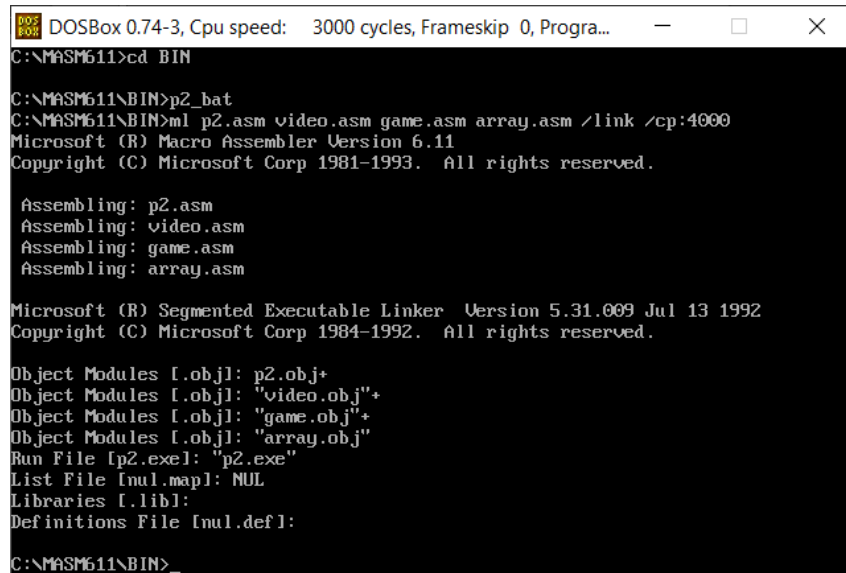
C:\>cd MASM611

C:\MASM611>cd BIN

C:\MASM611\BIN>
```

El archivo a compilar deberá estar en la carpeta montada

2. Compile el archivo escribiendo el nombre del archivo .bat que se incluye junto con los archivos .masm. El nombre del archivo .bat es **p2_bat.bat**



```
DOSBox 0.74-3, Cpu speed: 3000 cycles, Frameskip 0, Progra...
C:\MASM611>cd BIN
C:\MASM611\BIN>p2_bat
C:\MASM611\BIN>ml p2.asm video.asm game.asm array.asm /link /cp:4000
Microsoft (R) Macro Assembler Version 6.11
Copyright (C) Microsoft Corp 1981-1993. All rights reserved.

Assembling: p2.asm
Assembling: video.asm
Assembling: game.asm
Assembling: array.asm

Microsoft (R) Segmented Executable Linker Version 5.31.009 Jul 13 1992
Copyright (C) Microsoft Corp 1984-1992. All rights reserved.

Object Modules [.obj]: p2.obj+
Object Modules [.obj]: "video.obj"+
Object Modules [.obj]: "game.obj"+
Object Modules [.obj]: "array.obj"
Run File [p2.exe]: "p2.exe"
List File [nul.map]: NUL
Libraries [.lib]:
Definitions File [nul.def]:

C:\MASM611\BIN>_
```

De esta forma el compilador generará el archivo ejecutable y estará disponible para su posterior uso. El nombre del archivo ejecutable es **p2.exe**

El programa fue exitosamente ejecutado en un computador con las siguientes especificaciones:

- Windows ® 10 x64
- 8GB RAM
- Intel ® Core ™ i7-8550U
- DOSBox 0.74-3
- MASM v6.1

Sin embargo, se asegura su correcta ejecución con las siguientes especificaciones:

- Windows 8 o superior (x64 o x32)
- Cualquier versión de DOSBox compatible con MASM v6.1
- MASM v6.1

- 2GB RAM
- Intel ® Core ™ i3 (cualquier nomenclatura) o superior

6 ARCHIVOS INCLUIDOS

El programa consta de una serie de archivos que son necesarios para funcionar. Si se elimina cualquiera de esos archivos el programa podría quedar inutilizable. Se incluyen los siguientes archivos:

- Screen.asm
- String.asm
- FileH.asm
- Game.asm
- Array.asm
- Video.asm
- P2.asm
- P2linc.asm

Además, el programa trabaja sobre una serie de archivos que utiliza para almacenar información y luego cargarla para utilizarla durante su ejecución:

- Good.otz
- Bad.otz
- Car.otz
- Usr.tzy
- Score.tzy

7 USO DE LA MEMORIA DE VIDEO

La aplicación utiliza el modo video 13h. Se utiliza este modo por la facilidad que supone la manipulación de la memoria de video ya que esta se encuentra en un arreglo continuo y finito.

La aplicación manipula la memoria de video utilizando la técnica de doble buffer. Esta técnica permite reducir el parpadeo de animación, lo que hace que la aplicación se más amigable al usuario.

El doble buffer se crea de forma dinámica, es decir, esta no área de memoria no se reserva desde que se inicia la aplicación, sino durante su ejecución, esto permite ahorrar liberar la memoria cuando ya no se está utilizando y aprovecharla para otros usos.

8 INTERRUPCIÓN 1AH PARA MANEJAR RETRASOS EN HILOS DE EJECUCIÓN

La aplicación utiliza la interrupción por software 1ah. Esta interrupción se utiliza para acceder a servicios del reloj del procesador.

Con esta interrupción se manejan los retrasos que se utilizan en la aplicación y que permiten el uso de hilos de ejecución. A pesar de que pareciera que muchas cosas están ocurriendo al mismo tiempo, esto no es cierto para el procesador. A través de los hilos de ejecución es posible asignar recursos a varias tareas en diferentes instantes del tiempo de tal forma que parece que ocurren al mismo tiempo.

La macro utilizada es la 00h y 01h. Estas macros permiten conocer y establecer el número de *ticks* que han pasado desde que el computador está funcionando. 1 segundo equivale a aproximadamente 18 *ticks*. Dependiendo de la velocidad del procesador puede que ocurran más o menos *ticks* durante un segundo.

Al utilizar este método se asegura que el CPU no tenga tiempo de ociosidad y los recursos sean aprovechados eficientemente.

9 FUNCIONALIDADES DE LA APLICACIÓN

9.1 SINCRONIZACIÓN CON LA MEMORIA DE VIDEO

En el modo 13h la memoria de video comienza en la posición A000h. Lo que hace que este modo sea fácil de utilizar es que la memoria está ordenada continuamente.

La memoria de video se sincroniza con el doble buffer. El contenido del doble buffer se carga directamente a la memoria de video y así la pantalla mostrará el contenido del doble buffer y se evitará el parpadeo.

```
;-----
syncBuffer proc far c videoPos : word, startPos : word, base : word, height
: word, offPos : word
; STARTPOS : indica el offset de la memoria de video
; BASE      : largo de "línea" de información a copiar en una sola iteración
; HEIGHT    : número de iteraciones a realizar
; OFFPOS    : indica el offset desde donde se deberá copiar la información
; Copia la imagen almacenada en el doble buffer a la memoria de video
;-----
    local i : word
    pushad
    push es
    push ds
    mov i, 0                ;; inicializa i = 0
    mov ds, videoPos        ;; indica la pos de mem
    mov si, offPos
    mov dx, 0A000h
    mov es, dx              ;; indica la pos de mem de video
_syncBuff1:
    mov bx, i
    cmp bx, height
    jge _syncBuff2
    mov ax, 140h            ;; 320
    xor dx, dx
    mul bx                  ;; 320 * i
    add ax, startPos        ;; startPos + 320 * i
```



```

        mov di, ax        ;; indica el offset de la mem de video
        mov cx, base
        cld                ;; limpia el registro de flags
        rep movsb
        inc i
        jmp _syncBuff1
_syncBuff2:
        pop ds
        pop es
        popad
        ret
syncBuffer endp

```

9.2 BARRIDO HACIA ABAJO DE LA PISTA DE OBSTÁCULOS

```

;-----
scrollBackground proc near c
; Actualiza la pantalla principal.
; Reemplaza los pixeles inferiores con los superiores
;-----
        pushad
        push es
        push ds
        mov dx, vram
        mov ds, dx        ;; determina el origen
        mov es, dx        ;; determina el destino
        mov si, 32219      ;; indica el origen de la información  $178 * 180 + 17$ 
9
        mov di, 32399      ;; indica el destino de la información  $179 * 180 + 1$ 
79
        mov cx, 32220      ;;  $180 * 179$ 
        std                ;; los indices se decrementaran
        rep movsb
        mov al, 7
        mov cx, 180
        std                ;; el indice di se debe decrementar
        rep stosb          ;; la primera línea se pinta de color gris
        pop ds
        pop es
        popad
        ret
scrollBackground endp

```

9.3 FUNCIÓN PARA OBTENER NÚMERO DE FORMA ALEATORIA

Esta función se basa en el algoritmo de generador lineal congruencial para generar números aleatorios a partir de una semilla.

```
;-----  
rand proc near c uses eax ebx ecx edx  
; Genera un número aleatorio utilizando  
; el algoritmo de generador lineal congruencial (GLC)  
;  $X_{n+1} = (aX_n + c) \% m$   
; randomSeedn+1 = (randomSeed*48271 + 1) % (2^31)  
;-----  
    xor edx, edx  
    mov eax, randomSeed  
    mov ebx, 48271  
    mul ebx  
    add eax, 1  
    mov ebx, eax           ;; obtiene el modulo de la forma:  
    and ebx, 080000000h    ;; dividendo - (divisor * cociente) = residuo  
    sub eax, ebx  
    mov randomSeed, eax  
    ret  
rand endp
```

9.4 EJEMPLO DE USO DE LA INTERRUPCIÓN 1AH PARA EL MANEJO DE HILOS

```
;-----  
; Actualiza el contador de tiempo  
;-----  
    movzx dx, playState  
    cmp dx, 1           ;; está pausado  
    jz _playGame7       ;; se salta a leer el teclado  
    mov bx, dCtTime  
    add bx, 18           ;; se ejecuta cada 18 ticks  
    mov ah, 0h  
    int 1ah             ;; recupera el número de ticks  
    cmp dx, bx  
    jle _playGame71  
        mov dCtTime, dx       ;; actualiza el número de ticks  
        inc actualTime        ;; aumenta el numero de segundos  
        call timeComposing    ;; compone el contador a la forma hh:mm:ss  
        call printHeader      ;; actualiza el encabezado  
_playGame71:
```

9.5 ORDENAMIENTO BUBBLESORT

```
;-----
bubbleSort proc far c uses eax ebx ecx edx esi
; Ordena de forma ascendente un arreglo que inicia
; en statArr y de tamaño sizeArr
;-----
    local loc_sense : word, localDelay : word, i : word, j : word
    mov ax, sense
    mov loc_sense, ax                ;; especifica el sentido de ordenamiento
    mov ah, 00h
    int 1ah
    mov actualTicks, dx              ;; inicializa el número de ticks
    mov localDelay, dx
    xor cx, cx
    xor si, si
    mov cx, noUsers                  ;; especifica el tamaño del arreglo
    dec cx                           ;; disminuye el tamaño del arreglo
    mov i, cx
_bubbleSort0:
    cmp i, 0
    jz _bubbleSort8                  ;; termina el procedimiento
    xor si, si
    mov cx, i
    mov j, cx
    _bubbleSort1:
        cmp j, 0
        jz _bubbleSort7              ;; continúa con la siguiente iteración
    _bubbleSort2:
        call checkTimer
        mov bx, localDelay
        add bx, actualVel
        mov ah, 0h
        int 1ah
        cmp dx, bx
        jg _bubbleSort3
    jmp _bubbleSort2
    _bubbleSort3:
        mov localDelay, dx
        mov ax, sortArray[si]
        cmp loc_sense, 0
        jnz _bubbleSort4              ;; es ascendente
        cmp ax, sortArray[si + 2]
        jl _bubbleSort6               ;; si es menor no hace nada
        jmp _bubbleSort5
```

```

        _bubbleSort4:      ;; es descendente
            cmp ax, sortArray[si + 2]
            jg _bubbleSort6 ;; si es mayor no hace nada
        _bubbleSort5:
            xchg ax, sortArray[si + 2]
            invoke mkSound, ax
            mov sortArray[si], ax
            call graphSorted
            call printFooterA
        _bubbleSort6:
            add si, 2
            dec j
            jmp _bubbleSort1
        _bubbleSort7:
            dec i
            jmp _bubbleSort0
        _bubbleSort8:
            ret
bubbleSort endp

```