

# Seminario 1. Programación de dispositivos a bajo nivel

*Duración: 1 sesión*

## Objetivos

- I. Instalar un simulador del sistema operativo MS-DOS para poder programar las llamadas a interrupciones de entrada/salida en este S.O.
- II. Programar a bajo nivel (ensamblador) las rutinas de interrupción de MS-DOS.

## 1. Introducción



El sistema operativo MS-DOS (*MicroSoft Disk Operating System*) es un sistema operativo en modo texto para ordenadores de la arquitectura 80x86. Fue el sistema más usado para PCs compatibles con IBM PC en la década de 1980 y mediados de 1990, hasta que fue sustituido gradualmente por sistemas operativos con GUI (interfaz gráfica), en particular por varias generaciones de Microsoft Windows.

MS-DOS es un sistema operativo monousuario y monotarea, con interfaz en modo texto en la cual la comunicación entre el usuario y el sistema operativo se realiza mediante instrucciones formadas por caracteres introducidos desde el teclado.

## 2. Sistema Básico de Entrada/Salida (BIOS) de un PC

La BIOS (Sistema Básico de Entrada/Salida, *Basic Input/Output System*) de un IBM-PC es un conjunto de programas alojados en una memoria RAM-CMOS dentro de la placa base. Estas rutinas se utilizan durante el arranque del computador para configurar los diferentes dispositivos y arrancar a su vez el sistema operativo.

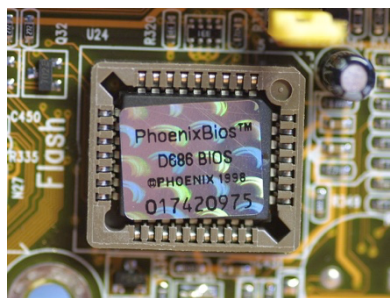


Figura 1. Chip de memoria para la BIOS en una placa base

En un menú típico de la BIOS, se pueden configurar las siguientes características:

- *Main o Standard CMOS Features*. Permite cambiar la hora y la fecha, y configurar varias opciones del disco duro u otras unidades de disco. Muestra informaciones sobre la BIOS, la CPU y la memoria.
- *Advanced o Advanced BIOS Features*. Permite activar o desactivar las funciones de red (LAN o inalámbrica), el USB, el teclado numérico. Definir el tipo de controlador del disco duro (SATA,

IDE). También opciones de la CPU, la memoria o la propia BIOS. Muchas de ellas orientadas a mejorar el rendimiento.

- **Security.** Definir, cambiar o quitar contraseñas para entrar en la configuración de la BIOS o en el sistema.
- **Power o Power Management Setup.** Gestionar las características de ahorro de energía del PC. Por ejemplo, si la pantalla o el disco duro deben o no entrar en suspensión. O cómo "despertar" la computadora cuando entra en ese estado.
- **Boot.** Se define la secuencia de arranque. Es decir, desde qué unidades y en qué orden el PC debe buscar un modo de iniciar.

Los PCs recientes sustituyen la BIOS por la llamada UEFI (*Unified Extensible Firmware Interface*), que extiende las capacidades de la BIOS permitiendo arranque en red, inicio selectivo de componentes y, en general, un arranque de los nuevos sistemas operativos más rápido.

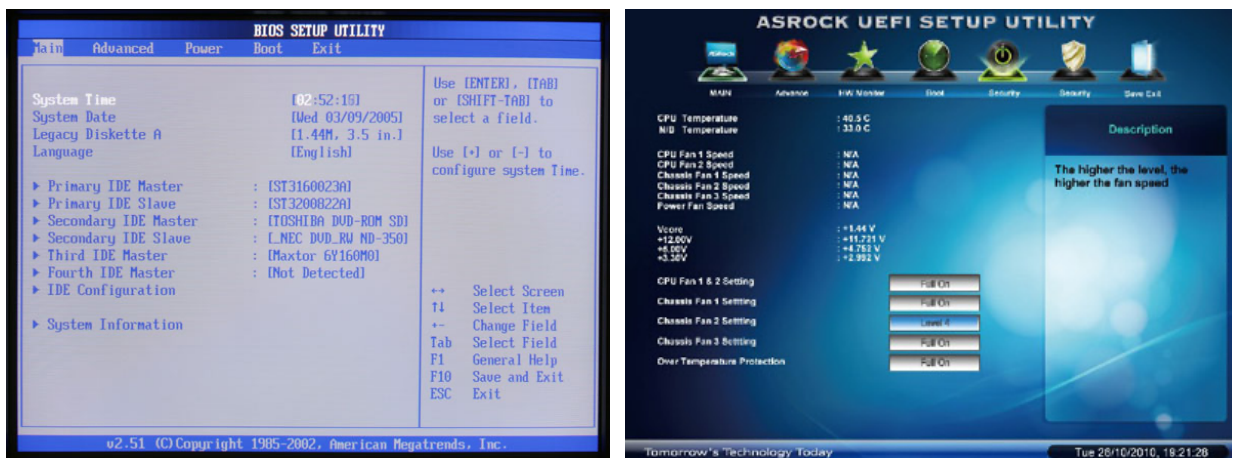


Figura 2. Menú BIOS (izquierda) y UEFI (derecha)

### 3. Arquitectura 80x86

La arquitectura 80x86 abarca la familia de procesadores Intel de 8, 16 y 32 bits. Le sucedieron las arquitecturas de 64 bits (IA-64). En los procesadores de 16 bits (8086, 80186 y 80286), éste dispone de 14 registros de 16 bits, 4 de ellos de propósito general:

1. **AX:** Registro de acumulador. Es el único que puede ser usado como multiplicando en la multiplicación y como dividendo en la división. Es fundamental en la programación de interrupciones, ya que selecciona los distintos servicios del Sistema Operativo.
2. **BX:** Registro de Base.
3. **CX:** Registro de Contador
4. **DX:** Registro de Datos

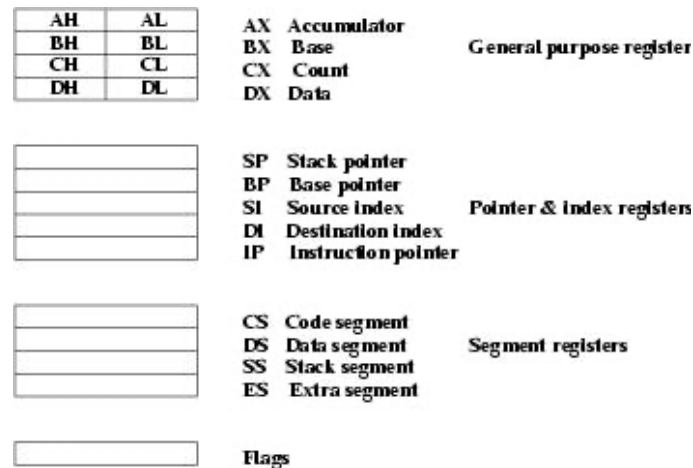


Figura 3. Registros de la arquitectura del procesador 8086

#### 4. Programación de interrupciones para acceder a los dispositivos de E/S

La memoria BIOS de cualquier PC contiene un conjunto de rutinas software para gestionar distintos dispositivos a nivel básico, tales como el teclado, el video, el ratón o las unidades de disco magnético. Estas rutinas son, de hecho, independientes del SO y se ejecutan a partir de interrupciones. Una interrupción es un evento que provoca que la CPU detenga su tarea actual y pase a ejecutar inmediatamente una determinada rutina. La comunicación entre la unidad básica y los periféricos se realiza frecuentemente por medio de estas interrupciones.

Las rutinas de interrupción de la BIOS pueden ser activadas desde un programa a través de la API de MS-DOS (Figura 4). Para ello, es preciso especificar qué rutina debe ejecutarse y los parámetros que ésta precise. Cada interrupción tiene asociado un número de rutina, bajo el cual se aglutinan a su vez a diversas subrutinas o subfunciones. Así pues, para ejecutar una rutina de interrupción específica hay que indicar, por un lado, un número de interrupción y, por otro, el número de subfunción. Además, una subfunción suele necesitar unos determinados parámetros de entrada y quizás devuelva una serie de valores al terminar de ejecutarse. Para pasar los parámetros de entrada se utilizan algunos de los registros internos de la CPU, a los que es necesario asignarle los distintos parámetros de entrada antes de invocar a la rutina de servicio de interrupción (ISR). Si la subfunción devuelve valores, éstos se encuentran en los registros internos justo al acabar de ejecutarse la subfunción.

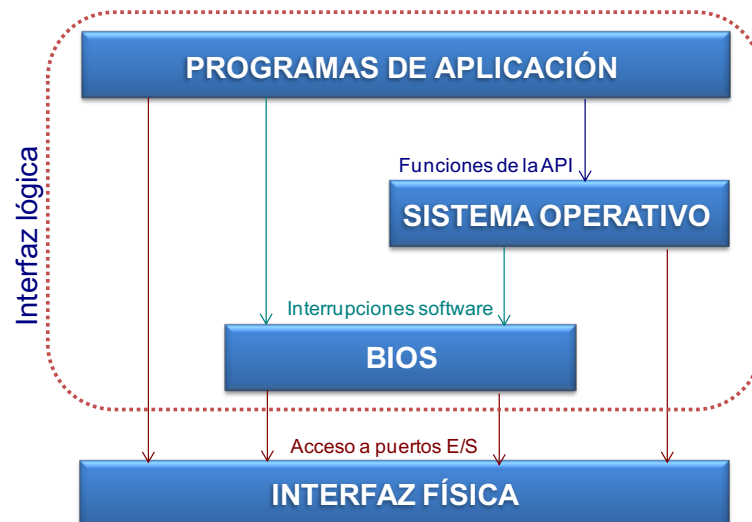


Figura 4. Niveles de acceso a los dispositivos de E/S. El acceso directo a la interfaz física a través de los programas no suele estar permitido por el sistema operativo, ni tampoco a las interrupciones de la BIOS. De esta forma, el sistema operativo se asegura que sólo se puede acceder a los dispositivos a través de su API (*Application Programming Interface*).

## 5. Emulando MS-DOS con DosBox

Los sistemas operativos de Microsoft posteriores a Windows XP no permiten la ejecución de programas realizados en MS-DOS. Cuando intentamos ejecutar un programa de ese tipo, nos aparece un mensaje parecido al de la Figura 5.

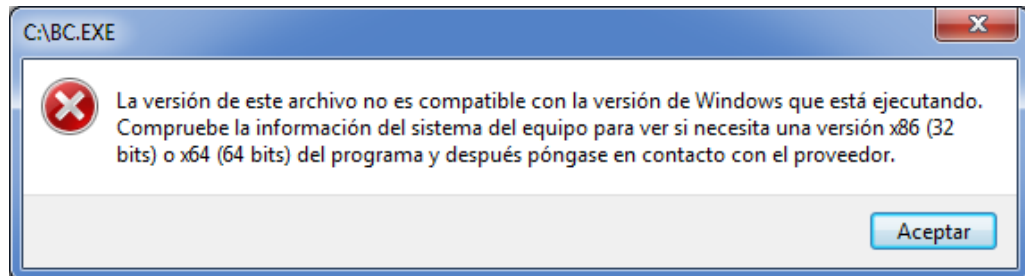


Figura 5. Ventana de error en Windows 7 al intentar ejecutar una aplicación no compatible

Aunque existen formas de ejecutar programas MS-DOS desde Linux o MacOS como Wine, WineSkin, etc; en estas prácticas se propone usar un software de emulación de MS-DOS como DOSBox. DOSBox es un emulador del sistema operativo de Microsoft MS-DOS (1983) en una ventana con el objetivo de poder ejecutar programas y videojuegos originalmente escritos para este sistema operativo ordenadores más modernos y con diferentes arquitecturas.

DOSBox está disponible para diferentes sistemas operativos (Linux, Windows, MacOS), es software libre y OpenSource. DOSBox emula la gestión de interrupciones del MS-DOS original de forma muy fiable, permitiendo acceder a la API de MS-DOS de manera idéntica. Se puede descargar desde <http://www.dosbox.com/> (Figura 6).

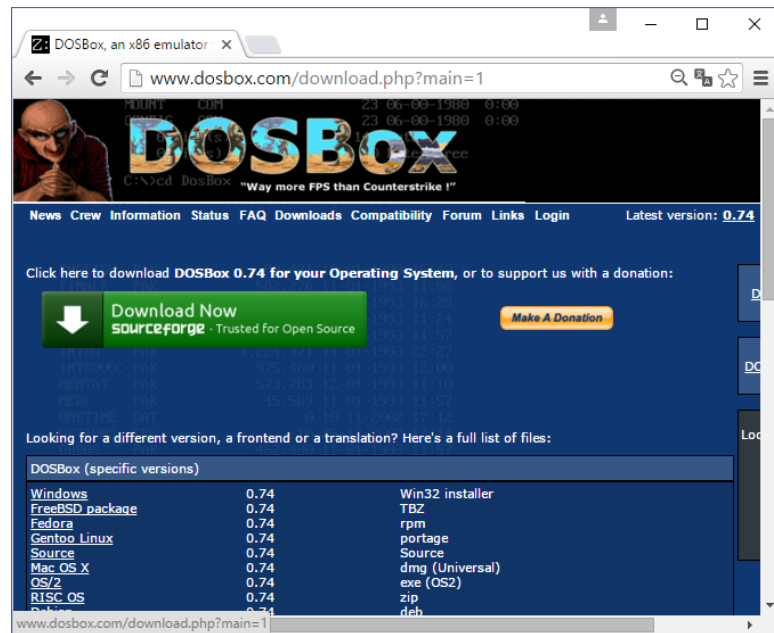


Figura 6. Sitio web para la descarga de DosBOX (<http://www.dosbox.com/>).

Existe una documentación bastante exhaustiva de cómo configurar y usar el emulador en [http://www.dosbox.com/wiki/Basic Setup and Installation of DosBox](http://www.dosbox.com/wiki/Basic_Setup_and_Installation_of_DosBox)

Para configurar las opciones de inicio y las unidades que queremos que DOSBox monte debemos modificar el fichero `dosbox-0.74.conf`

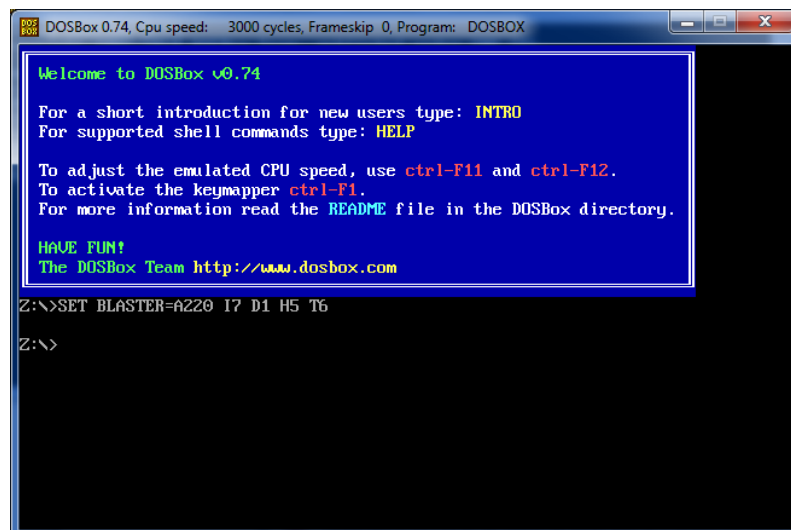


Figura 7. DOSBox en ejecución

Para montar en la unidad C: el contenido de una carpeta que tengamos en nuestra máquina anfitriona, debemos ejecutar en el DOSBOX:

```
mount C /home/pedro/DOSBOX
```

Y para configurar el teclado español hay que añadir lo siguiente a la sección `[autoexec]` del archivo de configuración:

```
keyb sp
```

En cada sistema operativo el archivo de configuración de DOSBOX está localizado en una carpeta diferente:

**Windows:** C:\Users\username\AppData\Local\DOSBox\dosbox-0.74.conf

**macOS:** /Users/username/Library/Preferences/DOSBox 0.74-3-3 Preferences

**Linux:** ~/.dosbox/dosbox-0.74.conf

Justo al final del todo de ese archivo podemos añadir las líneas para poner el teclado en español y para montar en la unidad C: la carpeta con todo el software necesario. A modo de ejemplo, el final del archivo de configuración del DOSBOX en mi máquina anfitriona es:

```
pedro ~ $ cat /Users/pedro/Library/Preferences/DOSBox\ 0.74-3\ Preferences

...
...
...

[autoexec]
# Lines in this section will be run at startup.
# You can put your MOUNT lines here.

mount C /Users/pedro/DOSBOX
keyb sp
path c:\bc\bin

c:

pedro ~ $ _
```

Como se puede ver, esas últimas cuatro líneas de la configuración las he añadido yo para:

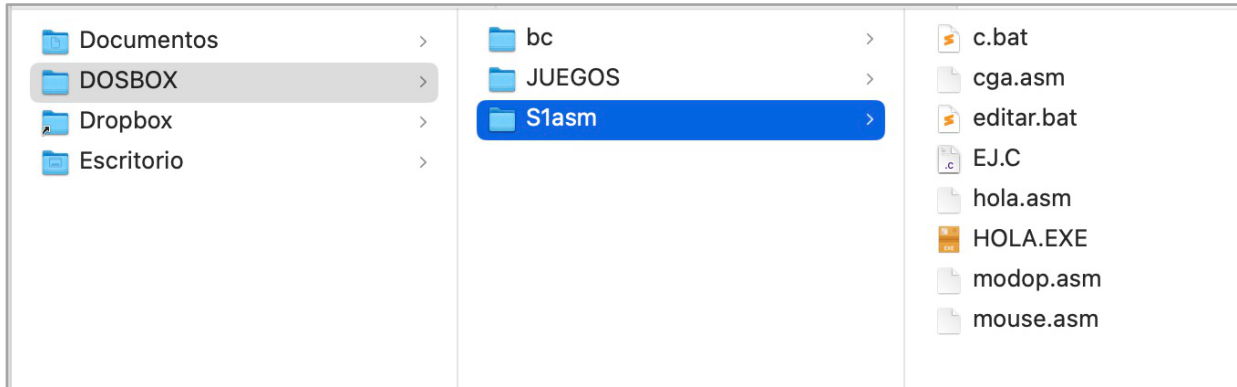
- (1) montar la carpeta para que DOSBOX la vea como su unidad C:
- (2) indico que quiero el teclado en español
- (3) indico que las utilidades de compilación están en ese path, y
- (4) hago que al arrancar DOSBOX ya esté localizado en C: para comenzar a trabajar con el contenido que he montado ahí.

Si además queremos que la ventana aparezca de un tamaño mayor porque la configuración por defecto hace que nos aparezca demasiado pequeña, podemos ir a la sección [sdl] del archivo de configuración y establecer el tamaño deseado en la variable “windowresolution” tal y como se muestra a continuación:

```
[sdl]
fullscreen=false
fulldouble=false
fullresolution=original
windowresolution=1024x768
output=opengl
autolock=true
sensitivity=100
```

Según la configuración indicada, sólo tenemos que copiar el software en el espacio de almacenamiento de nuestra máquina (que DOSBOX usará como unidad C). También debemos copiar las carpetas de ejemplos para trabajar los seminarios y la práctica 1.

Concretamente en mi caso he copiado la carpeta “bc”, la carpeta “S1asm” y la carpeta “JUEGOS” en /Users/pedro/DOSBOX/ tal y como se muestra a continuación en el explorador de archivos del sistema operativo anfitrión:



Así la aplicación DOSBOX tendrá acceso a esas carpetas y software una vez la ejecutemos:

```
DOSBox 0.74-3-3, Cpu speed: 3000 cycles, Frameskip 0, Program: DOSBOX
Z:\>mount C /Users/pedro/DOSBOX
Drive C is mounted as local directory /Users/pedro/DOSBOX/

Z:\>keyb sp
Keyboard layout sp loaded for codepage 858

Z:\>path c:\bc\bin

Z:\>c:

C:\>dir
Directory of C:\.
.                <DIR>          24-02-2023 10:12
..               <DIR>          17-02-2023 15:11
BC               <DIR>          23-09-2021 10:18
JUEGOS           <DIR>          17-10-2022 17:01
S1ASM            <DIR>          24-02-2023 10:11

C:\> _
```

## 6. Uso del ensamblador bajo MSDOS

Recordemos que un programa ensamblador para MSDOS sigue una estructura en tres segmentos, pila, datos y código, seguidos de una última línea con la directiva al compilador para declarar el punto de entrada al programa (función por la que se empieza a ejecutar el programa). El ejemplo más sencillo, “Hola mundo”, sigue la siguiente sintaxis (las líneas en negrita son fijas):

```
pila segment stack 'stack'
    dw 100h dup (?)
pila ends

datos segment 'data'
    msg db 'hola$'
datos ends

codigo segment 'code'
    assume cs:codigo, ds:datos, ss:pila
main PROC
    mov ax,datos
    mov ds,ax

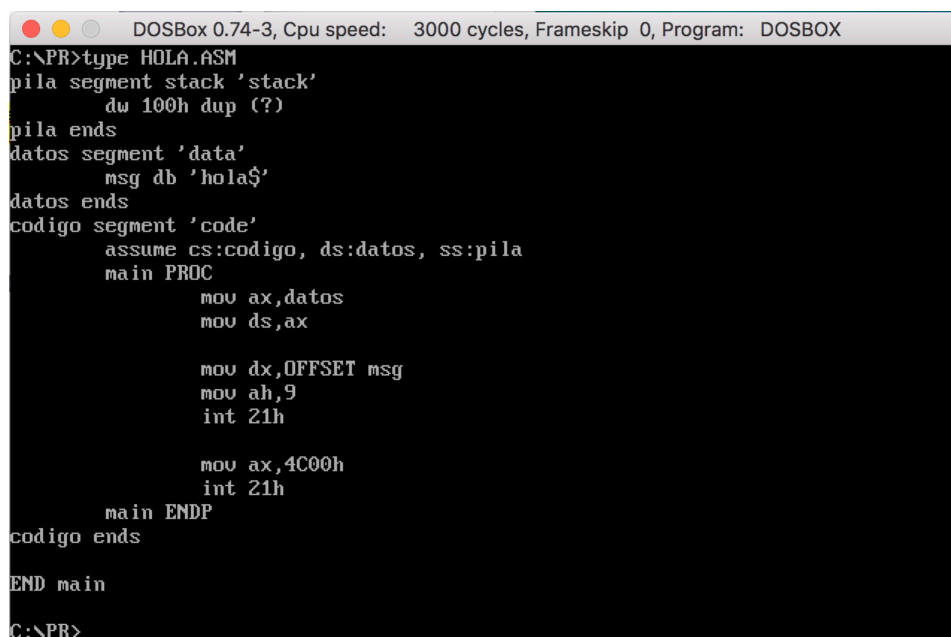
    mov dx,OFFSET msg ; mostrar por pantalla una cadena de texto
    mov ah,9
    int 21h

    mov ax,4C00h      ; terminar y salir al S.O.
    int 21h

    main ENDP
codigo ends

END main
```

Una vez hemos creado un archivo de texto plano, con extensión .asm, lo compilaremos en dos pasos (compilación y enlazado), llamando a los programas TASM.EXE y TLINK.EXE incluidos en el paquete Turbo Assembler de Borland:



```
DOSBox 0.74-3, Cpu speed: 3000 cycles, Frameskip 0, Program: DOSBOX
C:\PR>type HOLA.ASM
pila segment stack 'stack'
    dw 100h dup (?)
pila ends
datos segment 'data'
    msg db 'hola$'
datos ends
codigo segment 'code'
    assume cs:codigo, ds:datos, ss:pila
    main PROC
        mov ax,datos
        mov ds,ax

        mov dx,OFFSET msg
        mov ah,9
        int 21h

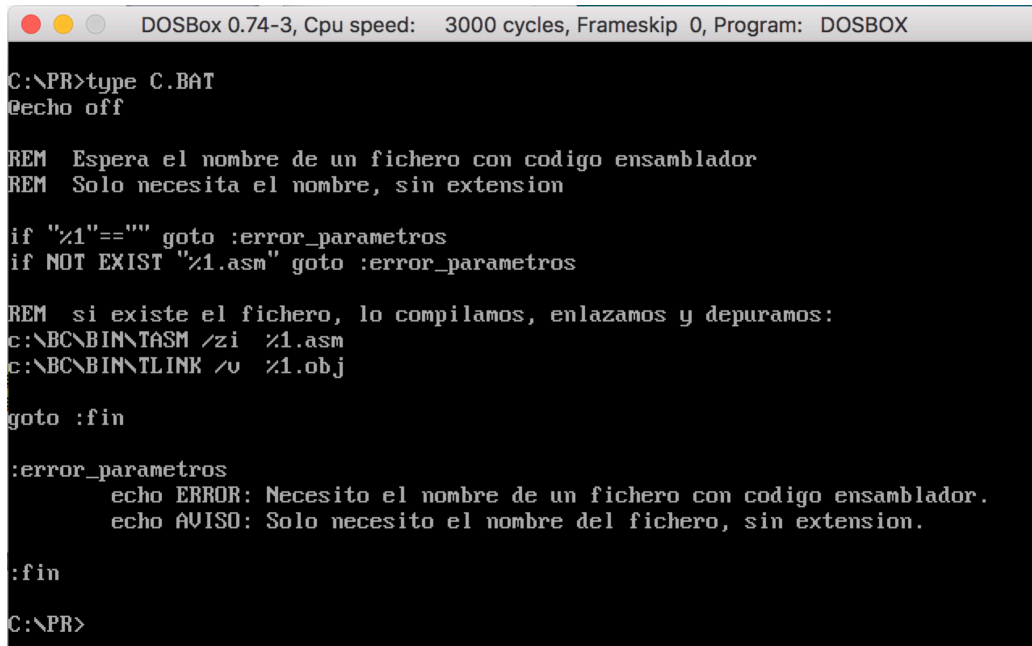
        mov ax,4C00h
        int 21h

    main ENDP
codigo ends

END main
C:\PR>_
```



Por simplicidad, merece la pena crear un script (en MSDOS son las macros con extensión .BAT) que recibe el nombre del programa a compilar:



```

DOSBox 0.74-3, Cpu speed: 3000 cycles, Frameskip 0, Program: DOSBOX

C:\PR>type C.BAT
@echo off

REM Espera el nombre de un fichero con codigo ensamblador
REM Solo necesita el nombre, sin extension

if "%1"==" " goto :error_parametros
if NOT EXIST "%1.asm" goto :error_parametros

REM si existe el fichero, lo compilamos, enlazamos y depuramos:
c:\BC\BIN\TASM /zi %1.asm
c:\BC\BIN\TLINK /v %1.obj

goto :fin

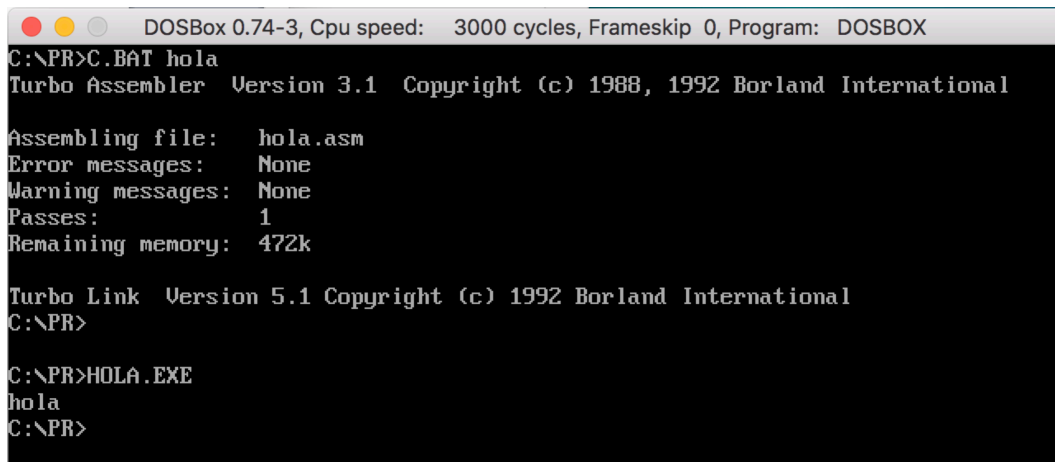
:error_parametros
    echo ERROR: Necesito el nombre de un fichero con codigo ensamblador.
    echo AVISO: Solo necesito el nombre del fichero, sin extension.

:fin

C:\PR>

```

La ejecución se hace llamando al nuevo archivo con extensión .EXE recién creado:



```

DOSBox 0.74-3, Cpu speed: 3000 cycles, Frameskip 0, Program: DOSBOX

C:\PR>C.BAT hola
Turbo Assembler Version 3.1 Copyright (c) 1988, 1992 Borland International

Assembling file: hola.asm
Error messages: None
Warning messages: None
Passes: 1
Remaining memory: 472k

Turbo Link Version 5.1 Copyright (c) 1992 Borland International
C:\PR>

C:\PR>HOLA.EXE
hola
C:\PR>

```

Si en nuestro programa queremos implementar un bucle dentro del cual se muestre repetidamente un mensaje de texto, debemos utilizar un registro como contador (CX), declarar una etiqueta, y comprobar la condición del bucle (salto condicional):

```

;imprimir N veces una cadena
mov cx,0
bucle:
    mov dx,OFFSET msg
    mov ah,9
    int 21h
    ;actualizar contador y comprobar condición
    inc cx
    cmp cx,5
    jne bucle

```

Otras **funciones de entrada/salida** en ensamblador, usando funciones de la **BIOS** o del **DOS**:

**Colocar el cursor en modo texto en (X,Y):**

```
mov dl,x ; dl=columna
mov dh,x ; dl=fila
mov bh,0
mov ah,2 ;función para posicionar el cursor
int 10h ;interrupción BIOS para pantalla
```

**Escribir un carácter en pantalla (int 10h):**

```
mov al,CHARACTER ;código ASCII del carácter a escribir
mov bx,0
mov ah,0Eh ;función para escribir un carácter
int 10h ;interrupción BIOS para pantalla
```

**Escribir un carácter en pantalla (int 21h):**

```
mov dl,CHARACTER ;código ASCII del carácter a escribir
mov ah,2 ;función para escribir un carácter
int 21h ;interrupción MSDOS para pantalla
```

**Esperar la pulsación de una tecla (int 16h):**

```
mov ah,0 ;función para leer una tecla
int 16h ;interrupción BIOS para teclado
;en AH devuelve el identificador de la tecla
;en AL devuelve el código ASCII de la tecla pulsada
```

**Esperar la pulsación de una tecla sin mostrarla por pantalla (controla Crtl-Break):**

```
mov ah,08h ;función para leer una tecla
int 21h ;interrupción DOS para teclado
;en AL devuelve el carácter tecleado
```

**Esperar la pulsación de una tecla mostrándola por pantalla (controla Crtl-Break):**

```
mov ah,01h ;función para leer una tecla
int 21h ;interrupción DOS para teclado
;en AL devuelve el carácter tecleado
```

**Poner el MODO gráfico y pintar un punto en X,Y de COLOR específico:**

```
0 - texto 40x25 b/n
1 - texto 40x25 color
2 - texto 80x25 b/n
3 - texto 80x25 color
4 - gráfico 320x200 con cuatro colores (CGA)
5 - gráfico 320x200 b/n
6 - gráfico 640x200 b/n
```

**Poner el modo texto o gráfico deseado (int 10h):**

```
mov al,MODO
mov ah,0 ;función para establecer el modo de pantalla
int 10h ;interrupción BIOS para pantalla
```

**Pintar un pixel en modo gráfico (previamente debemos haber puesto el modo gráfico):**

```
mov cx,X ;columna
mov dx,Y ;fila
mov al,COLOR ;color del pixel iluminado
mov ah,0Ch ;función para iluminar un pixel
int 10h ;interrupción BIOS para pantalla
```

La interrupción 33h controla el ratón en el modo gráfico. Usaremos las funciones para inicializarlo y para comprobar la pulsación de alguno de los botones:

Inicializar el ratón en modo gráfico:

```
mov ax,0001
int 33h
```

Comprobar si hay una pulsación de uno de los botones del ratón (0=izq ; 1=der):

```
mov ax,5
mov bx,0 ; 0=izq ; 1=der
int 33h
```

El siguiente ejemplo pone el modo gráfico en pantalla, a continuación inicializa el ratón para mostrar el puntero, y entra en un bucle que espera a que se pulse el botón izquierdo del ratón. Para terminar, restaura el modo texto de pantalla y sale al S.O:

```
pila segment stack 'stack'
    dw 100h dup (?)
pila ends

datos segment 'data'
datos ends

codigo segment 'code'
    assume cs:codigo, ds:datos, ss:pila
main PROC
    mov ax,datos
    mov ds,ax

    ; modo de vídeo GRAFICO
    mov al, 13h
    mov ah,0
    int 10h

    ; inicializar el ratón
    mov ax,0001
    int 33h

esperar_izq:
    mov ax,5
    mov bx,0 ; 0=botón_izq ; 1=botón_der
    int 33h

    cmp bx,0
    je esperar_izq

    ; restaurar modo de vídeo TEXTO
    mov al, 3h
    mov ah,0
    int 10h

    ; terminar y salir
    mov ax,4C00h
    int 21h

main ENDP
codigo ends
END main
```

## 7. Ejercicios

- Instalar el software DOSBOX y ejecutar aplicaciones de MS-DOS (p.ej. juegos clásicos).
- Configurar el inicio de DOSBOX para que monte en su unidad C: el directorio donde se encuentra el entorno de programación Borland C (BC) que incluye las herramientas para compilar no sólo lenguaje C, sino también ensamblador. Añadir a la variable "PATH" de inicio el directorio "bin" donde se encuentra el ejecutable BC.EXE
- Crear el ejemplo "Hola mundo" en ensamblador, compilarlo y comprobar su funcionamiento. A continuación, modificar ese ejemplo para incluir un bucle que muestre ese mensaje 7 veces.
- Sacar capturas de pantalla en las que se muestre la configuración realizada y el funcionamiento del programa realizado una vez ejecutado.

### Forma de Entrega:

La práctica o seminario podrá realizarse de manera individual o por grupos de hasta 2 personas.

Se entregará como un archivo de texto en el que se muestre la información requerida. También se puede utilizar la sintaxis de Markdown para conseguir una mejor presentación e incluso integrar imágenes o capturas de pantalla. La entrega se realizará subiendo los archivos necesarios al repositorio "**PDIH**" en la cuenta de GitHub del estudiante, a una carpeta llamada "**S1**".

Toda la documentación y material exigidos se entregarán en la fecha indicada por el profesor. No se recogerá ni admitirá la entrega posterior de las prácticas/seminarios ni de parte de los mismos.

La detección de copias implicará el suspenso inmediato de todos los implicados en la copia (tanto de quien realizó el trabajo como de quien lo copió).

Las faltas de ortografía se penalizarán con hasta 1 punto de la nota de la práctica o seminario.

## 8. Enlaces y recursos

<https://www.dosbox.com/DOSBoxManual.html>

<https://www.linuxadictos.com/dosbox-en-linux.html>

<https://es.wikihow.com/usar-DOSBox>

<http://ubuntudriver.blogspot.com/2011/09/instalacion-basica-de-dosbox-en-ubuntu.html>

[https://issuu.com/jorgils/docs/manual\\_b\\_sico\\_para\\_ensamblador\\_f](https://issuu.com/jorgils/docs/manual_b_sico_para_ensamblador_f)

<http://www.nachocabanes.com/tutors/asmperif.htm>

<http://irlenys.tripod.com/microii/interr.htm>

[http://ict.udlap.mx/people/oleg/docencia/ASSEMBLER/asm\\_interrup\\_21.html](http://ict.udlap.mx/people/oleg/docencia/ASSEMBLER/asm_interrup_21.html)

<https://www.dosgamers.com/es/dos/dosbox-dos-emulador/screen-resolution>

<https://www.enmimaquinafunciona.com/pregunta/168127/aumentar-el-tamano-de-la-ventana-de-dosbox>

## **ANEXO. Programación básica en ensamblador x86**

Un programa ensamblador para MSDOS sigue una estructura en tres segmentos, pila, datos y código, seguidos de una última línea con la directiva al compilador para declarar el punto de entrada al programa (función por la que se empieza a ejecutar el programa).

A modo de ejemplo muy sencillo, se muestra a continuación un programa que muestra una cadena de texto por pantalla. La sintaxis es muy estricta (las líneas en negrita son fijas):

```
pila segment stack 'stack'
    dw 100h dup (?)
pila ends

datos segment 'data'
    msg db 'hola$'
datos ends

codigo segment 'code'
    assume cs:codigo, ds:datos, ss:pila
    main PROC
        mov ax,datos
        mov ds,ax

        mov dx,OFFSET msg    ; mostrar por pantalla una cadena de texto
        mov ah,9
        int 21h

        mov ax,4C00h          ; terminar y salir al S.O.
        int 21h

    main ENDP
codigo ends

END main
```

Una vez hemos introducido estas instrucciones en un archivo de texto plano, con extensión .asm, lo compilaremos en dos pasos (compilación y enlazado). La ejecución se hace llamando al nuevo archivo con extensión .EXE recién creado.

Como vemos, cada tarea en el programa la hemos llevado a cabo haciendo una llamada a una función a interrupción. Las líneas en verde hacen una llamada a la función de la interrupción 21h (MSDOS) que muestra por pantalla (salida) una cadena de texto que le indiquemos. Por su parte, las líneas en azul hacen una llamada a la función de la interrupción 21h (MSDOS) que terminan el programa y salen al S.O.

De esta forma, si quisiéramos leer de teclado (entrada), sólo tenemos que incluir en el punto concreto del programa las instrucciones para hacer la llamada a interrupción para esperar una pulsación de tecla.

Para realizar E/S por los dispositivos convencionales (pantalla, teclado y ratón), podemos usar las siguientes funciones de interrupción. No son las únicas, ya que tanto la BIOS como el MSDOS ofrecen diferentes funciones para hacer las mismas tareas sencillas (lectura de una tecla, mostrar un carácter, etc).

#### **terminar el programa y salir al MSDOS**

```
mov ah, 4Ch ; función de terminar el programa y salir
mov al, 00h ; código de salida al S.O.
int 21h
```

#### **escribir una cadena (terminada en \$) por pantalla**

```
mov dx, OFFSET cadena
mov ah, 9
int 21h
```

#### **escribir el carácter CHARACTER en pantalla (int 21h)**

```
mov dl, CHARACTER
mov ah, 2
int 21h
```

#### **esperar la pulsación de una tecla sin mostrarla por pantalla**

```
mov ah, 08h ; función para leer una tecla
int 21h ; interrupción DOS para teclado
; en AL devuelve el carácter tecleado
```

#### **esperar la pulsación de una tecla mostrándola por pantalla**

```
mov ah, 01h ; función para leer una tecla
int 21h ; interrupción DOS para teclado
; en AL devuelve el carácter tecleado
```

#### **colocar el cursor en modo texto en la coordenada x,y (int 10h)**

```
mov dl, columna ; dl=columna
mov dh, fila ; dl=fila
mov bh, 0
mov ah, 2 ; función para posicionar el cursor
int 10h ; interrupción BIOS para pantalla
```

#### **poner el MODO gráfico y pintar un punto en X,Y de COLOR específico:**

MODO = 0 – texto 40x25 b/n

1 - texto 40x25 color

2 – texto 80x25 b/n

3 – texto 80x25 color

4 – gráfico 320x200 con cuatro colores (CGA)

5 – gráfico 320x200 b/n

6 – gráfico 640x200 b/n

```
mov al, MODO
mov ah, 0 ; función para poner modo texto o gráfico deseado
int 10h
```

**dibujar un pixel de cierto color en la coordenada x,y indicada en la pantalla gráfica (es necesario haber establecido un modo gráfico de pantalla):**

```
mov cx,X    ;columna
mov dx,Y    ;fila
mov al,COLOR
mov ah,0Ch
int 10h
```

Como ejemplo, veamos cómo hacer una pausa esperando una pulsación de tecla:

```
pila segment stack 'stack'
    dw 100h dup (?)
pila ends

datos segment 'data'
    cadena db 13,10,'pulsa una tecla...',13,10,'$'
datos ends

codigo segment 'code'
    assume cs:codigo, ds:datos, ss:pila
    main PROC
        mov ax,datos
        mov ds,ax

        mov dx,OFFSET cadena
        mov ah,9
        int 21h

        mov ah,8
        int 21h

        mov ax,4C00h
        int 21h
    main ENDP

codigo ends
END main
```