

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.

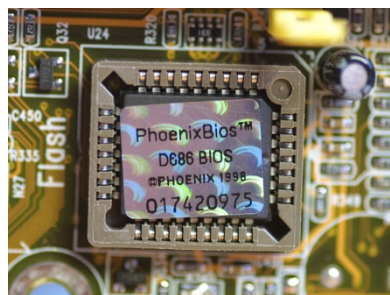


Figura 1. Chip de memoria para la BIOS en una placa base

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.

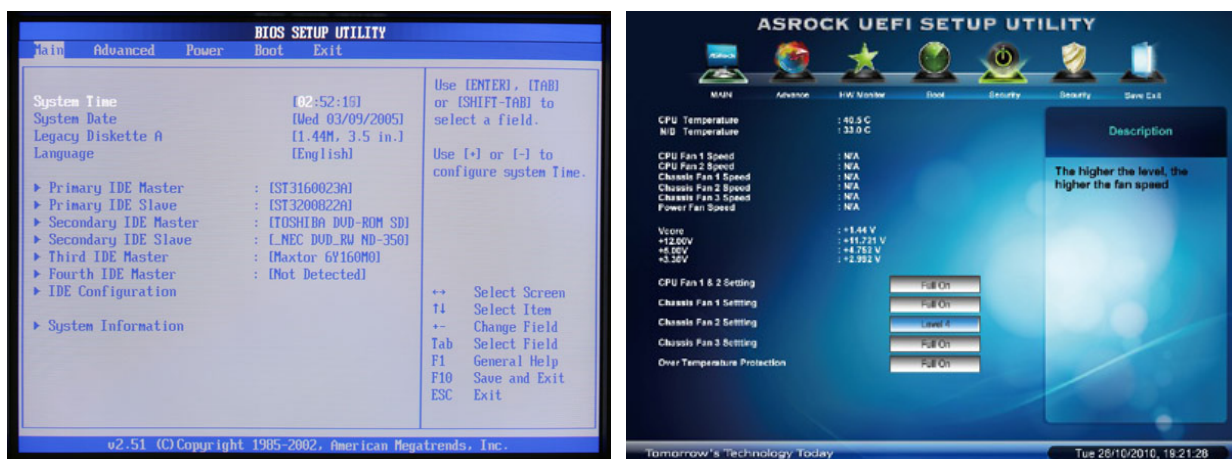


Figura 2. Menú BIOS (izquierda) y UEFI (derecha)

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

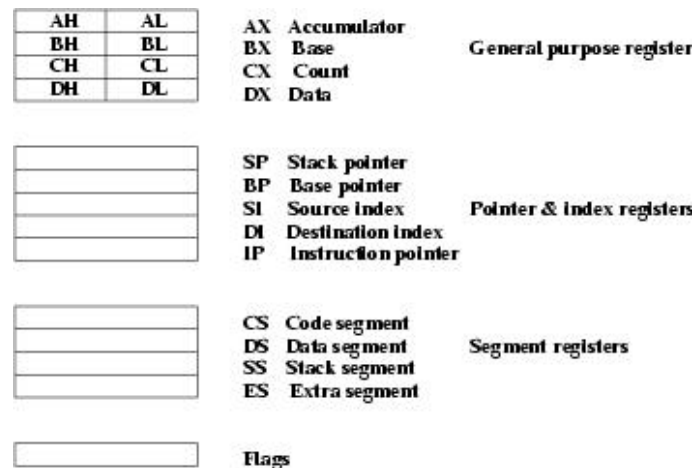


Figura 3. Registros de la arquitectura del procesador 8086

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.

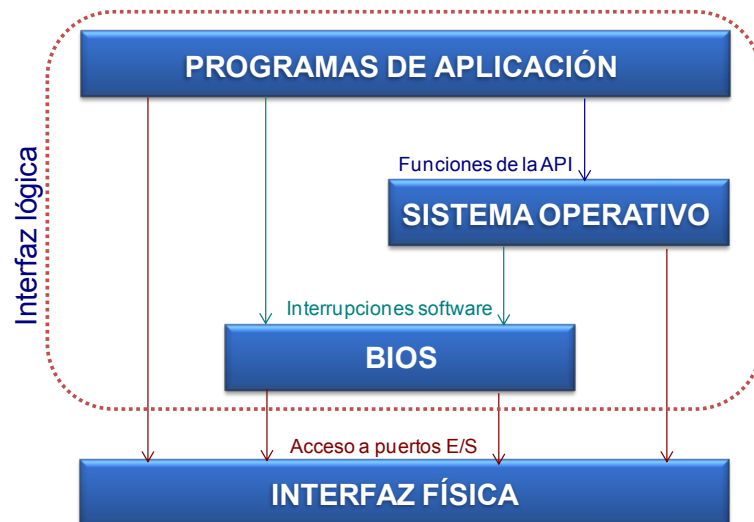


Figura 4. Niveles de acceso a los dispositivos de E/S. El acceso directo a la interfaz física a través de los programas no suele estar permitido por el sistema operativo, ni tampoco a las interrupciones de la BIOS. De esta forma, el sistema operativo se asegura que sólo se puede acceder a los dispositivos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

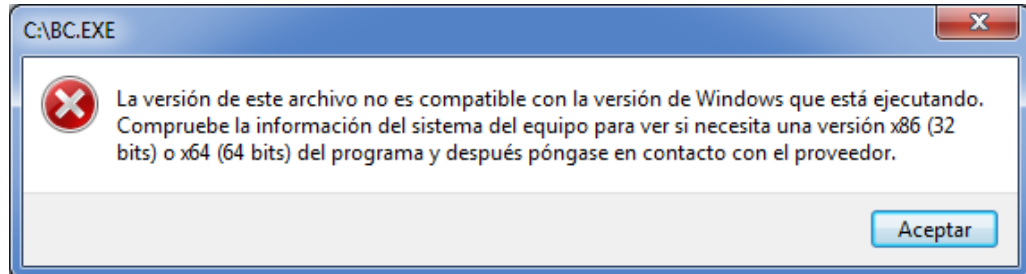


Figura 5. Ventana de error en Windows 7 al intentar ejecutar una aplicación no compatible

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).

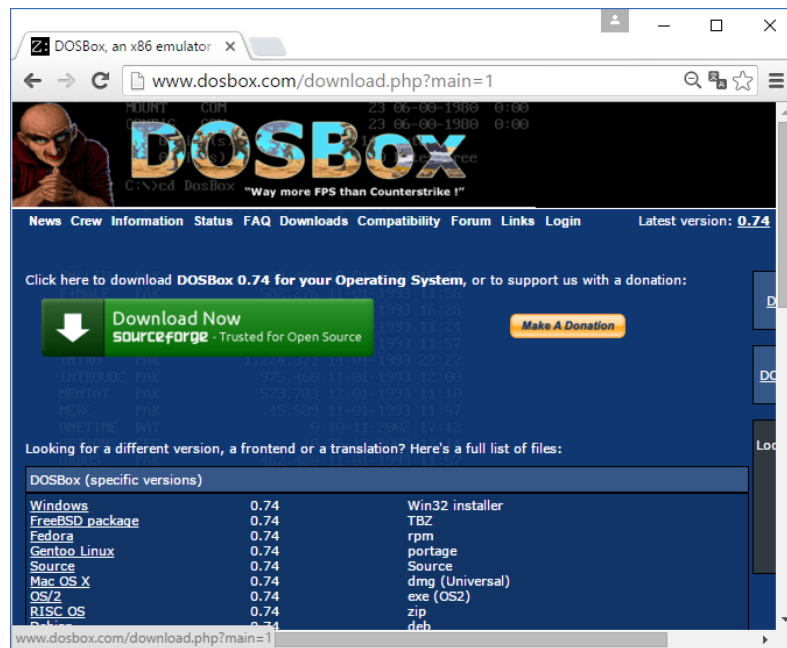


Figura 6. Sitio web para la descarga de DosBOX (<http://www.dosbox.com/>).

Existe una documentación bastante exhaustiva de cómo configurar y usar el emulador en http://www.dosbox.com/wiki/Basic_Setup_and_Installation_of_DosBox

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

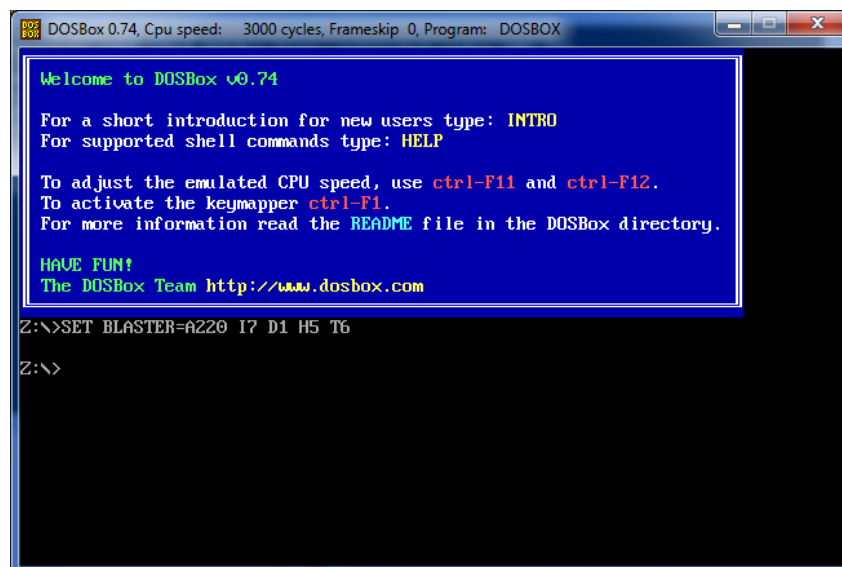


Figura 7. DOSBox en ejecución

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 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.

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

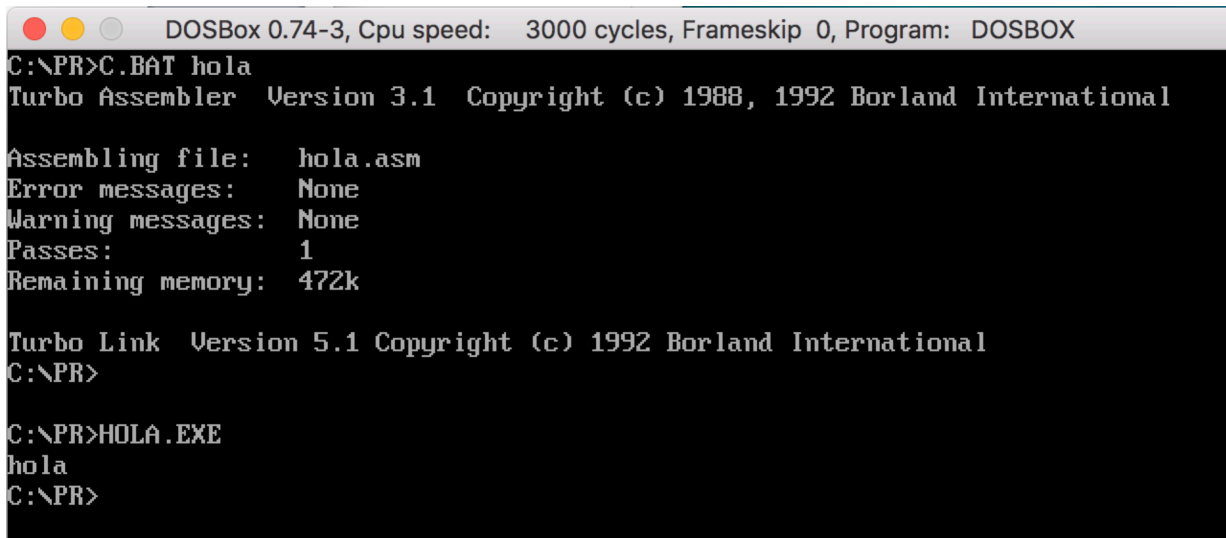
REM si existe el fichero, lo compilamos, enlazamos y depuramos:
c:\BC\BIN\TASM /zi %1.asm
c:\BC\BIN\TLINK /v %1.obj

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 Ctrl-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 Ctrl-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 la unidad C: el directorio donde se encuentra el entorno de programación Borland C (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 incluir un bucle para mostrar el mensaje 7 veces.

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/using-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

<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

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, column ; 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
```

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    ;funcion para posicionar el cursor
int 10h     ;interrupcion BIOS para pantalla
```

escribir el caracter CHARACTER en pantalla (int 10h)

```
mov al,CHARACTER
mov bx,0
mov ah,0Eh  ;funcion para escribir un caracter
int 10h     ;interrupcion BIOS para pantalla
```

escribir el caracter CHARACTER en pantalla (int 21h)

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

esperar la pulsacion de una tecla

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

esperar la pulsacion de una tecla sin mostrarla por pantalla (controla Crtl-Break)

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

esperar la pulsacion de una tecla mostrándola por pantalla (controla Crtl-Break)

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

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 - grafico 320x200 color
        5 - grafico 320x200 b/n
        6 - grafico 640x200 b/n
```

```
mov al,MODO
mov ah,0    ;funcion para poner el modo texto o grafico deseado
int 10h
```

```
;esta parte (pintar pixels en modo grafico
;sólo si hemos puesto modo gráfico)
mov cx,X    ;columna
mov dx,Y    ;fila
mov al,COLOR
mov ah,0Ch
int 10h
```

escribir una cadena (terminada en \$) por pantalla

función 09h de int 21h

la cadena a mostrar se pasa en el reg DX

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

esperar la pulsación de una tecla

función 01h de int 21h

el valor leído es devuelto en el registro AL

```
mov ah, 01h
int 21h
```

escribir una letra por pantalla

función 02h de int 21h

la letra a mostrar se pasa en el reg DL

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

terminar el programa y salir al MSDOS

función 4Ch de int 21h

en el reg AL se indica el código de salida

```
mov ah, 4Ch
mov al, 00h
int 21h
```

colocar el cursor (modo texto) en la posición (X,Y)

función 02h de int 10h

el reg BH se inicializa con valor 00h

en el reg DL se le pasa la columna, y en DH la fila

```
mov dl, COLUMNA
mov dh, FILA
mov bh, 0
mov ah, 2
int 10h
```

cambiar el modo de pantalla a 40 columnas por 25 filas

función 00h de int 10h

en el reg AL se indica el nuevo modo de pantalla (1 = 40x25)

```
mov al, 1
mov ah, 0
int 10h
```

cambiar el modo de pantalla a 80 columnas por 25 filas

función 00h de int 10h

en el reg AL se indica el nuevo modo de pantalla (3 = 80x25)

```
mov al, 3
mov ah, 0
int 10h
```


<p><i>;Ejemplo de procedimiento</i></p> <pre> pila segment stack 'stack' dw 100h dup (?) pila ends datos segment 'data' msg db 'hola mundo (con procedim)\$' datos ends codigo segment 'code' assume cs:codigo, ds:datos, ss:pila main PROC mov ax,datos mov ds,ax mov dx,OFFSET msg call escribir mov ax,4C00h int 21h main ENDP escribir PROC mov ah,9 int 21h ret escribir ENDP codigo ends END main </pre>	<p><i>;Ejemplo de macro</i></p> <pre> escribir macro msg mov dx,OFFSET msg mov ah,9 int 21h endm pila segment stack 'stack' dw 100h dup (?) pila ends datos segment 'data' msg db 'hola mundo (con macro)\$' datos ends codigo segment 'code' assume cs:codigo, ds:datos, ss:pila main PROC mov ax,datos mov ds,ax escribir msg mov ax,4C00h int 21h main ENDP codigo ends END main </pre>
<p><i>;mostrar un número por pantalla</i></p> <pre> pila segment stack 'stack' db 128h dup ('pila') pila ends datos segment 'data' numero dw 89 datos ends codigo segment 'code' assume cs:codigo, ds:datos, ss:pila ;----- main PROC mov ax,datos mov ds,ax mov ax,numero call escribir_numero mov ax,4C00h int 21h main ENDP ;----- escribir_numero PROC push ax push bx push dx mov bx,10 mov dl,al cmp ax,bx jb escribir_resto sub dx,dx div bx call escribir_numero escribir_resto: add dl,'0' mov ah,2 int 21h pop dx pop bx pop ax ret escribir_numero ENDP codigo ends END main </pre>	<p><i>;pasar una cadena a número</i></p> <pre> pila segment stack 'stack' dw 100h dup (?) pila ends datos segment 'data' cadena db '0123' long_cad dw 4 numero dw ? datos ends codigo segment 'code' assume cs:codigo, ds:datos, ss:pila main PROC mov ax,datos mov ds,ax mov cx,long_cad mov ax,0 mov di,10 mov bh,0 mov si,0 bucle: mul di mov bl,byte ptr cadena[si] sub bl,'0' add ax,bx inc si loop bucle mov numero,ax mov ax,4C00h int 21h main ENDP codigo ends END main </pre>

<pre> ;ejemplo de manejo de cadenas de caracteres pila segment stack 'stack' dw 100h dup (?) pila ends datos segment 'data' cadena db 'cadena de texto',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 bx, offset cadena mov ah,[bx+0] mov al,[bx+1] mov byte ptr [bx+0],al mov byte ptr [bx+1],ah mov si,6 mov byte ptr [bx+si], '-' mov si,9 mov byte ptr [bx+si], '-' mov dx, OFFSET cadena mov ah, 9 int 21h mov ah, 4Ch mov al, 00h int 21h main ENDP codigo ends END main </pre>	<pre> ;transformar un número entero en cadena de caracteres pila segment stack 'stack' db 128h dup ('pila') pila ends datos segment 'data' numero dw 23456 cadena db ' ' datos ends codigo segment 'code' assume cs:codigo, ds:datos, ss:pila main PROC FAR mov ax,datos mov ds,ax mov ax,numero mov bx,5 ;desplz sobre la cadena mov si,10 ;base bucle: sub dx,dx div si add dl,'0' dec bx mov byte ptr cadena[bx],dl or ax,ax jnz bucle mov byte ptr cadena[5], '\$' ;mostrar la cadena creada mov dx,OFFSET cadena mov ah,9 int 21h mov ax,4C00h int 21h main ENDP codigo ENDS END main </pre>
<pre> pila segment stack 'stack' dw 100h dup (?) pila ends datos segment 'data' matriz db 1,2,3,4,5 db 6,7,8,9,0 db 23,45,12,78,96 db 34,6,78,2,9 db 0,10,20,30,40 datos ends codigo segment 'code' assume cs:codigo, ds:datos, ss:pila main PROC mov ax,datos mov ds,ax mov bx,offset matriz ; acceder a la [fila=0,col=0]=="1" mov si,0 ; si=5*fila add si,0 ; si=si+col mov al,[bx+si] ; acceder a la [fila=3,col=2]=="78" mov ax,3 ; ax=fila mov si,5 ; si=num_elems_por_fila mul si add ax,2 ; ax=(num_els*fila)+col mov si,ax mov al,[bx+si] mov ah, 4Ch mov al, 00h int 21h main ENDP codigo ends END main </pre>	<pre> pila segment stack 'stack' dw 100h dup (?) pila ends datos segment 'data' x dd 1000 y dd 100000 z dd 34000 datos ends codigo segment 'code' assume cs:codigo, ds:datos, ss:pila main PROC mov ax,datos mov ds,ax mov ax, word ptr x ; parte baja de x mov dx, word ptr x+2 ; parte alta de x add ax, word ptr z ; sumar parte baja adc dx, word ptr z+2 ; sumar parte alta sub ax, 5 ; restamos una constante sbb dx, 0 ; ...por si "nos llevamos" mov word ptr y, ax ; parte baja en Y mov word ptr y+2, dx ; parte alta en Y mov ah, 4Ch mov al, 00h int 21h main ENDP codigo ends END main </pre>

<u>CGA (320x200 ; 4 colores)</u>	<u>VGA (320x200 ; 256 colores)</u>
<pre> ;cambia el modo de video ;texto=3h o grafico_cga=4h modo_video MACRO modo push ax mov al,modo mov ah,0 int 10h pop ax ENDM ;pone un pixel en X,Y de color C pixel MACRO X,Y,C push ax push cx push dx mov ax,Y mov cx,X mov dx,ax mov al,C mov ah,0Ch int 10h pop dx pop cx pop ax ENDM 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_video 4h pixel 10,10, 1 pixel 20,20, 2 pixel 30,30, 3 mov ah,0 int 16h modo_video 3h mov ax,4C00h int 21h main ENDP codigo ends END main </pre>	<pre> modo_video MACRO modo push ax mov al, modo mov ah,0 int 10h pop ax ENDM ;pone el color de un pixel pixel MACRO X,Y,C push ax push bx push cx push dx mov ax, 0a000h mov es,ax mov ax,Y mov cx,X mov dx,0 mov bx,320 imul bx add ax,cx mov bx,ax mov es:[bx],byte ptr C pop dx pop cx pop bx pop ax ENDM 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_video 13h pixel 10,100, 5 pixel 20,80, 30 pixel 30,40, 145 mov ah,0 int 16h modo_video 3h mov ax,4C00h int 21h main ENDP codigo ends END main </pre>

MOVIMIENTO DE UN PUNTO EN PANTALLA EN MODOS CGA

```
TEXTO EQU 3h ; darle nombre a esos valores
GRAFICO EQU 4h ; como constantes

modo_video MACRO modo
    push ax
    mov al,modo
    mov ah,0
    int 10h
    pop ax
ENDM

pixel MACRO X,Y,C
    push ax
    push cx
    push dx
    mov ax,Y
    mov cx,X
    mov dx,ax
    mov al,C
    mov ah,0Ch
    int 10h
    pop dx
    pop cx
    pop ax
ENDM

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

datos segment 'data'
    msg_inicio db 13,10,'mueve con las teclas j
y k (q para terminar)$'
datos ends

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

        mov dx,offset msg_inicio
        mov ah,9
        int 21h
        mov ah,0
        int 16h
        modo_video GRAFICO

        mov cx,100 ; coord. X del punto
        mov dx,100 ; coord. Y del punto
```

```
bucle:
    ;ver si hay teclas disponibles
    mov ah,0bh
    int 21h
    ;al=00 -> buffer vacío
    ;al=FF -> leer pulsaciones
    ;borramos el pixel anterior (negro)
    pixel cx,dx,0

    cmp al,00
    jz seguir
    ; si hay pulsaciones, entonces...
    mov ah,08h
    int 21h

    comp_izq:
        ;si pulsa j -> decrem.coord.X
        cmp al,'j'
        jnz comp_dcha
        dec cx
        jmp seguir
    comp_dcha:
        ;si pulsa k -> increm.coord.X
        cmp al,'k'
        jnz comp_otras
        inc cx
        jmp seguir
    comp_otras:
        ;si pulsa q -> ir a fin
        cmp al,'q'
        jnz seguir
        jmp fin
    seguir:
        ;pintamos en la posic. actualiz.
        pixel cx,dx,3
        jmp bucle

fin:

    mov ah,0
    int 16h
    modo_video TEXTO



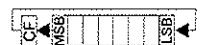
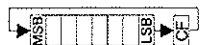

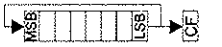
    mov ax,4C00h
    int 21h

    main ENDP
codigo ends



END main
```

TRANSFERENCIA		Código	Operación	Flags									
Nombre	Comentario			O	D	I	T	S	Z	A	P	C	
MOV	Mover (copiar)	MOV Dest,Fuente	Dest:=Fuente										
XCHG	Intercambiar	XCHG Op1,Op2	Op1:=Op2 , Op2:=Op1										
STC	Set the carry (Carry = 1)	STC	CF:=1									1	
CLC	Clear Carry (Carry = 0)	CLC	CF:=0									0	
CMC	Complementar Carry	CMC	CF:= ¬CF									±	
STD	Setear dirección	STD	DF:=1 (interpreta strings de arriba hacia abajo)		1								
CLD	Limpiar dirección	CLD	DF:=0 (interpreta strings de abajo hacia arriba)		0								
STI	Flag de Interrupción en 1	STI	IF:=1			1							
CLI	Flag de Interrupción en 0	CLI	IF:=0			0							
PUSH	Apilar en la pila	PUSH Fuente	DEC SP, [SP]:=Fuente										
PUSHF	Apila los flags	PUSHF	O, D, I, T, S, Z, A, P, C 286+: También NT, IOPL										
PUSHA	Apila los registros generales	PUSHA	AX, CX, DX, BX, SP, BP, SI, DI										
POP	Desapila de la pila	POP Dest	Destino:=[SP], INC SP										
POPF	Desapila a los flags	POPF	O, D, I, T, S, Z, A, P, C 286+: También NT, IOPL	±	±	±	±	±	±	±	±	±	
POPA	Desapila a los reg. general.	POPA	DI, SI, BP, SP, BX, DX, CX, AX										
CBW	Convertir Byte a Word	CBW	AX:=AL (con signo)										
CWD	Convertir Word a Doble	CWD	DX:AX:=AX (con signo)	±				±	±	±	±	±	
CWDE	Conv. Word a Doble Exten.	CWDE 386	EAX:=AX (con signo)										
IN <i>i</i>	Entrada	IN Dest,Puerto	AL/AX/EAX := byte/word/double del puerto especifi.										
OUT <i>i</i>	Salida	OUT Puerto,Fuente	Byte/word/double del puerto especifi. := AL/AX/EAX										

i para más información ver especificaciones de la instrucción Flags: ±=Afectado por esta instrucción ?=Indefinido luego de esta instrucción

ARITMÉTICOS		Código	Operación	Flags								
Nombre	Comentario			O	D	I	T	S	Z	A	P	C
ADD	Suma	ADD Dest,Fuente	Dest:=Dest+ Fuente	±				±	±	±	±	±
ADC	Suma con acarreo	ADC Dest,Fuente	Dest:=Dest+ Fuente +CF	±				±	±	±	±	±
SUB	Resta	SUB Dest,Fuente	Dest:=Dest- Fuente	±				±	±	±	±	±
SBB	Resta con acarreo	SBB Dest,Fuente	Dest:=Dest-(Fuente +CF)	±				±	±	±	±	±
DIV	División (sin signo)	DIV Op	Op=byte: AL:=AX / Op AH:=Resto	?				?	?	?	?	?
DIV	División (sin signo)	DIV Op	Op=word: AX:=DX:AX / Op DX:=Resto	?				?	?	?	?	?
DIV 386	División (sin signo)	DIV Op	Op=doublew.: EAX:=EDX:EAX / Op EDX:=Resto	?				?	?	?	?	?
IDIV	División entera con signo	IDIV Op	Op=byte: AL:=AX / Op AH:=Resto	?				?	?	?	?	?
IDIV	División entera con signo	IDIV Op	Op=word: AX:=DX:AX / Op DX:=Resto	?				?	?	?	?	?
IDIV 386	División entera con signo	IDIV Op	Op=doublew.: EAX:=EDX:EAX / Op EDX:=Resto	?				?	?	?	?	?
MUL	Multiplicación (sin signo)	MUL Op	Op=byte: AX:=AL*Op si AH=0 ♦	±				?	?	?	?	±
MUL	Multiplicación (sin signo)	MUL Op	Op=word: DX:AX:=AX*Op si DX=0 ♦	±				?	?	?	?	±
MUL 386	Multiplicación (sin signo)	MUL Op	Op=double: EDX:EAX:=EAX*Op si EDX=0 ♦	±				?	?	?	?	±
IMUL i	Multiplic. entera con signo	IMUL Op	Op=byte: AX:=AL*Op si AL es suficiente ♦	±				?	?	?	?	±
IMUL	Multiplic. entera con signo	IMUL Op	Op=word: DX:AX:=AX*Op si AX es suficiente ♦	±				?	?	?	?	±
IMUL 386	Multiplic. entera con signo	IMUL Op	Op=double: EDX:EAX:=EAX*Op si EAX es sufi. ♦	±				?	?	?	?	±
INC	Incrementar	INC Op	Op:=Op+1 (El Carry no resulta afectado !)	±				±	±	±	±	
DEC	Decrementar	DEC Op	Op:=Op-1 (El Carry no resulta afectado !)	±				±	±	±	±	
CMP	Comparar	CMP Op1,Op2	Op1-Op2	±				±	±	±	±	±
SAL	Desplazam. aritm. a la izq.	SAL Op,Cantidad		i				±	±	?	±	±
SAR	Desplazam. aritm. a la der.	SAR Op,Cantidad		i				±	±	?	±	±
RCL	Rotar a la izq. c/acarreo	RCL Op,Cantidad		i								±
RCR	Rotar a la derecha c/acarreo	RCR Op,Cantidad		i								±
ROL	Rotar a la izquierda	ROL Op,Cantidad		i								±
ROR	Rotar a la derecha	ROR Op,Cantidad		i								±

i para más información ver especificaciones de la instrucción ♦ entonces CF:=0, OF:=0 sino CF:=1, OF:=1

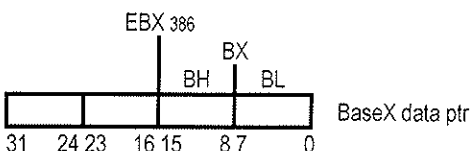
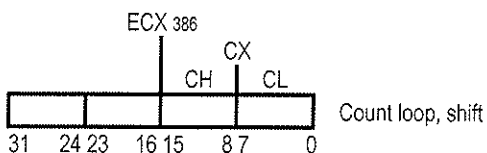
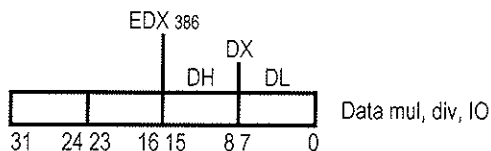
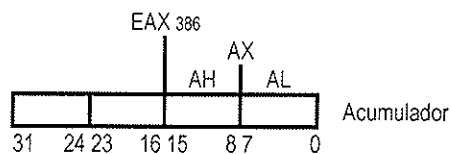
LÓGICOS		Código	Operación	Flags								
Nombre	Comentario			O	D	I	T	S	Z	A	P	C
NEG	Negación (complemento a 2)	NEG Op	Op:=0-Op si Op=0 entonces CF:=0 sino CF:=1	±				±	±	±	±	±
NOT	Invertir cada bit	NOT Op	Op:=-Op (invierte cada bit)									
AND	'Y' (And) lógico	AND Dest,Fuente	Dest:=Dest∧Fuente	0				±	±	?	±	0
OR	'O' (Or) lógico	OR Dest,Fuente	Dest:=Dest∨Fuente	0				±	±	?	±	0
XOR	'O' (Or) exclusivo	XOR Dest,Fuente	Dest:=Dest (xor) Fuente	0				±	±	?	±	0
SHL	Desplazam. lógico a la izq.	SHL Op,Cantidad		i				±	±	?	±	±
SHR	Desplazam. lógico a la der.	SHR Op,Cantidad		i				±	±	?	±	±

MISCELÁNEOS				Flags								
Nombre	Comentario	Código	Operación	O	D	I	T	S	Z	A	P	C
NOP	Hacer nada	NOP	No hace operación alguna									
LEA	Cargar dirección Efectiva	LEA Dest,Fuente	Dest := dirección fuente									
INT	Interrupción	INT Num	Interrumpe el progr. actual, corre la subrutina de int.			0	0					

SALTOS (generales)							
Nombre	Comentario	Código	Operación	Name	Comentario	Código	Operación
CALL	Llamado a subrutina	CALL Proc		RET	Retorno de subrutina	RET	
JMP	Saltar	JMP Dest					
JE	Saltar si es igual	JE Dest	(= JZ)	JNE	Saltar si no es igual	JNE Dest	(= JNZ)
JZ	Saltar si es cero	JZ Dest	(= JE)	JNZ	Saltar si no es cero	JNZ Dest	(= JNE)
JCXZ	Saltar si CX es cero	JCXZ Dest		JECXZ	Saltar si ECX es cero	JECXZ Dest	386
JP	Saltar si hay paridad	JP Dest	(= JPE)	JNP	Saltar si no hay paridad	JNP Dest	(= JPO)
JPE	Saltar si hay paridad par	JPE Dest	(= JP)	JPO	Saltar si hay paridad impar	JPO Dest	(= JNP)

SALTOS Sin Signo (Cardinal)				SALTOS Con Signo (Integer)			
JA	Saltar si es superior	JA Dest	(= JNBE)	JG	Saltar si es mayor	JG Dest	(= JNLE)
JAE	Saltar si es superior o igual	JAE Dest	(= JNB = JNC)	JGE	Saltar si es mayor o igual	JGE Dest	(= JNL)
JB	Saltar si es inferior	JB Dest	(= JNAE = JC)	JL	Saltar si es menor	JL Dest	(= JNGE)
JBE	Saltar si es inferior o igual	JBE Dest	(= JNA)	JLE	Saltar si es menor o igual	JLE Dest	(= JNG)
JNA	Saltar si no es superior	JNA Dest	(= JBE)	JNG	Saltar si no es mayor	JNG Dest	(= JLE)
JNAE	Saltar si no es super. o igual	JNAE Dest	(= JB = JC)	JNGE	Saltar si no es mayor o igual	JNGE Dest	(= JL)
JNB	Saltar si no es inferior	JNB Dest	(= JAE = JNC)	JNL	Saltar si no es inferior	JNL Dest	(= JGE)
JNBE	Saltar si no es infer. o igual	JNBE Dest	(= JA)	JNLE	Saltar si no es menor o igual	JNLE Dest	(= JG)
JC	Saltar si hay carry	JC Dest		JO	Saltar si hay Overflow	JO Dest	
JNC	Saltar si no hay carry	JNC Dest		JNO	Saltar si no hay Overflow	JNO Dest	
				JS	Saltar si hay signo (=negativo)	JS Dest	
				JNS	Saltar si no hay signo (=posit.)	JNS Dest	

Registros Generales:



Ejemplo: (programa 16 bits bajo MSDOS)

```

.DOSSEG ; Programa de demostración
.MODEL SMALL
.STACK 1024
Two EQU 2 ; Constante
.DATA
VarB DB ? ; define un Byte, cualquier valor
VarW DW 1010b ; define un Word, en binario
VarW2 DW 257 ; define un Word, en decimal
VarD DD 0AFFFFh ; define un DoubleWord, en hexa
S DB "Hello!",0 ; define un String
.CODE
main: MOV AX,DGROUP ; resuelto por el linker
MOV DS,AX ; inicializa el reg. de segmento de datos
MOV [VarB],42 ; inicializa VarB
MOV [VarD],-7 ; setea VarD
MOV BX,Offset[S] ; dirección de "H" de "Hello !"
MOV AX,[VarW] ; poner el valor en el acumulador
ADD AX,[VarW2] ; suma VarW2 a AX
MOV [VarW2],AX ; almacena AX en VarW2
MOV AX,4C00h ; regresa al sistema
INT 21h
END main

```

Flags: ☐ ☐ ☐ ☐ ☐ ☐ ☐ ☐ ☐ ☐ ☐ ☐ ☐ ☐

Flags de Control (cómo se manejan las instrucciones):

D: Dirección 1=Los op's String se procesan de arriba hacia abajo
I: Interrupción Indica si pueden ocurrir interrupciones o no.
T: Trampa Paso por paso para debugging

Flags de Estado (resultado de las operaciones):

C: Carry resultado de operac. sin signo es muy grande o inferior a cero
O: Overflow resultado de operac. sin signo es muy grande o pequeño.
S: Signo Signo del resultado. Razonable sólo para enteros. 1=neg. 0=pos.
Z: Cero Resultado de la operación es cero. 1=Cero
A: Carru. Aux. Similar al Carry, pero restringido para el nibble bajo únicamente
P: Paridad 1=el resultado tiene cantidad par de bits en uno

Números sin signo

Instrucción	JE
Condición	$A=B$
Operación	$A-B=0$
Flags	$Z=1$
Explicación	Si al hacer la resta los dos valores son iguales el resultado será 0 y por tanto se activará el biestable de cero (Z)

Instrucción	JNE
Condición	$A \neq B$
Operación	$A-B \neq 0$
Flags	$Z=0$
Explicación	Si al hacer la resta los dos valores no son iguales el resultado nunca será 0 y por tanto no se activará el biestable de cero (Z)

Instrucción	JB / JNAE
Condición	$A < B$
Operación	$A-B < 0$
Flags	$C=1$
Explicación	Si al hacer la resta A es más pequeño que B el resultado será negativo. Como estamos trabajando con números sin signo el resultado está fuera del rango y se activará el biestable de acarreo (C).

Instrucción	JNB / JAE
Condición	$A \geq B$
Operación	$A-B \geq 0$
Flags	$C=0$
Explicación	Si al hacer la resta A es mayor o igual que B el resultado será un número dentro del rango y por tanto no se producirá acarreo.

Instrucción	JNBE / JA
Condición	$A > B$
Operación	$A-B > 0$
Flags	$C=0$ y $Z=0$
Explicación	La condición $A > B$ se puede expresar en términos de $A \geq B$ y $A \neq B$. Por tanto se deben de cumplir ambas condiciones para que se produzca el salto. Estas condiciones son: $C=0$ y $Z=0$

Instrucción	JBE / JNA
Condición	$A \leq B$
Operación	$A-B \leq 0$
Flags	$C=1$ ó $Z=1$
Explicación	La condición $A \leq B$ se puede expresar en términos de $A < B$ ó $A=B$. Por tanto basta que se cumpla una condición para que se produzca el salto. Estas condiciones son $C=1$ ó $Z=1$

Números sin signo

Instrucción	JE
Condición	$A=B$
Operación	$A-B=0$
Flags	$Z=1$
Explicación	Si al hacer la resta los dos valores son iguales el resultado será 0 y por tanto se activará el biestable de cero (Z)

Instrucción	JNE
Condición	$A \neq B$
Operación	$A-B \neq 0$
Flags	$Z=0$
Explicación	Si al hacer la resta los dos valores no son iguales el resultado nunca será 0 y por tanto no se activará el biestable de cero (Z)

Instrucción	JB / JNAE
Condición	$A < B$
Operación	$A-B < 0$
Flags	$C=1$
Explicación	Si al hacer la resta A es más pequeño que B el resultado será negativo. Como estamos trabajando con números sin signo el resultado está fuera del rango y se activará el biestable de acarreo (C).

Instrucción	JNB / JAE
Condición	$A \geq B$
Operación	$A-B \geq 0$
Flags	$C=0$
Explicación	Si al hacer la resta A es mayor o igual que B el resultado será un número dentro del rango y por tanto no se producirá acarreo.

Instrucción	JNBE / JA
Condición	$A > B$
Operación	$A-B > 0$
Flags	$C=0$ y $Z=0$
Explicación	La condición $A > B$ se puede expresar en términos de $A \geq B$ y $A \neq B$. Por tanto se deben de cumplir ambas condiciones para que se produzca el salto. Estas condiciones son: $C=0$ y $Z=0$

Instrucción	JBE / JNA
Condición	$A \leq B$
Operación	$A-B \leq 0$
Flags	$C=1$ ó $Z=1$
Explicación	La condición $A \leq B$ se puede expresar en términos de $A < B$ ó $A=B$. Por tanto basta que se cumpla una condición para que se produzca el salto. Estas condiciones son $C=1$ ó $Z=1$

Instrucción	JE
Condición	A=B
Operación	A-B = 0
Flags	Z=1
Explicación	Si al hacer la resta los dos valores son iguales el resultado será 0 y por tanto se activará el biestable de cero (Z)

Instrucción	JNE
Condición	A≠B
Operación	A-B ≠ 0
Flags	Z=0
Explicación	Si al hacer la resta los dos valores no son iguales el resultado nunca será 0 y por tanto no se activará el biestable de cero (Z)

Instrucción	JL / JNGE
Condición	A < B
Operación	A-B < 0
Flags	S≠V
Explicación	Si al hacer la resta A es más pequeño que B el resultado será negativo. La diferencia entre A y B puede estar dentro del rango. En ese caso se activará el biestable de signo (S), quedando el resto de flags a 0 (y en especial el flag de desbordamiento (V)). Sin embargo si la diferencia entre A y B se sale del rango el resultado no será negativo por que pasará del mínimo valor del rango al máximo, quedando S=0 y activándose el biestable de desbordamiento (V). Un ejemplo de este caso podría ser A = -EEEEh y B=CCCCh ya que el resultado se sale del mínimo rango.

Instrucción	JNL / JGE
Condición	A ≥ B
Operación	A-B ≥ 0
Flags	S=V
Explicación	Si al hacer la resta A es mayor o igual que B el resultado será un número positivo. La diferencia entre A y B puede estar dentro del rango, en cuyo caso el resultado será positivo (S=0) y no se producirá acarreo (V=0). Ahora bien, si la diferencia se sale del rango el resultado será negativo al pasarse del valor máximo permitido al mínimo permitido (S=1) y por este motivo se activará el flag de overflow (V=1). Un ejemplo de este caso podría ser A=FFFFh y B=-CCCCCh ya que el resultado de la resta supera al máximo permitido.

Instrucción	JNLE / JG
Condición	A > B
Operación	A-B > 0
Flags	S=V y Z=0
Explicación	La condición A>B se puede expresar en términos de A≥B y A≠B. Por tanto se deben de cumplir ambas condiciones para que se produzca el salto. Estas condiciones son: S=V y Z=0

Instrucción	JLE / JNG
Condición	A ≤ B
Operación	A-B ≤ 0
Flags	S≠V ó Z=1
Explicación	La condición A≤B se puede expresar en términos de A<B ó A=B. Por tanto basta que se cumpla una condición para que se produzca el salto. Estas condiciones son S≠V y Z=1

Instrucción	JE
Condición	A=B
Operación	A-B = 0
Flags	Z=1
Explicación	Si al hacer la resta los dos valores son iguales el resultado será 0 y por tanto se activará el biestable de cero (Z)

Instrucción	JNE
Condición	A≠B
Operación	A-B ≠ 0
Flags	Z=0
Explicación	Si al hacer la resta los dos valores no son iguales el resultado nunca será 0 y por tanto no se activará el biestable de cero (Z)

Instrucción	JL / JNGE
Condición	A < B
Operación	A-B < 0
Flags	S≠V
Explicación	Si al hacer la resta A es más pequeño que B el resultado será negativo. La diferencia entre A y B puede estar dentro del rango. En ese caso se activará el biestable de signo (S), quedando el resto de flags a 0 (y en especial el flag de desbordamiento (V)). Sin embargo si la diferencia entre A y B se sale del rango el resultado no será negativo por que pasará del mínimo valor del rango al máximo, quedando S=0 y activándose el biestable de desbordamiento (V). Un ejemplo de este caso podría ser A = -EEEEh y B=CCCCCh ya que el resultado se sale del mínimo rango.

Instrucción	JNL / JGE
Condición	A ≥ B
Operación	A-B ≥ 0
Flags	S=V
Explicación	Si al hacer la resta A es mayor o igual que B el resultado será un número positivo. La diferencia entre A y B puede estar dentro del rango, en cuyo caso el resultado será positivo (S=0) y no se producirá acarreo (V=0). Ahora bien, si la diferencia se sale del rango el resultado será negativo al pasarse del valor máximo permitido al mínimo permitido (S=1) y por este motivo se activará el flag de overflow (V=1). Un ejemplo de este caso podría ser A=FFFFh y B=-CCCCCh ya que el resultado de la resta supera al máximo permitido.

Instrucción	JNLE / JG
Condición	A > B
Operación	A-B > 0
Flags	S=V y Z=0
Explicación	La condición A>B se puede expresar en términos de A≥B y A≠B. Por tanto se deben de cumplir ambas condiciones para que se produzca el salto. Estas condiciones son: S=V y Z=0

Instrucción	JLE / JNG
Condición	A ≤ B
Operación	A-B ≤ 0
Flags	S≠V ó Z=1
Explicación	La condición A≤B se puede expresar en términos de A<B ó A=B. Por tanto basta que se cumpla una condición para que se produzca el salto. Estas condiciones son S≠V y Z=1