

El byte y los formatos de datos

Toda información que se procesa en una PC es almacenada en celdas llamadas *registros*. Un *registro* es una agrupación física (algo que se puede ver y tocar) de 8, 16, 32 o 64 *flip-flops* (también conocidos como biestables) cuyo contenido es operado en forma simultánea.

Desde el punto de vista lógico (o lo que es lo mismo, la forma en que el procesador ve, mira u opera los datos) podemos describir los siguientes formatos de datos:

- una agrupación de 64 bits se conoce como “cuádruple palabra”,
- una agrupación de 32 bits se conoce como “doble palabra”,
- una agrupación de 16 bits se conoce como “palabra” y
- una palabra puede dividirse en grupos de 8 bits conocidos como “bytes”;
- un grupo de 4 bits recibe el nombre de “nibble”.

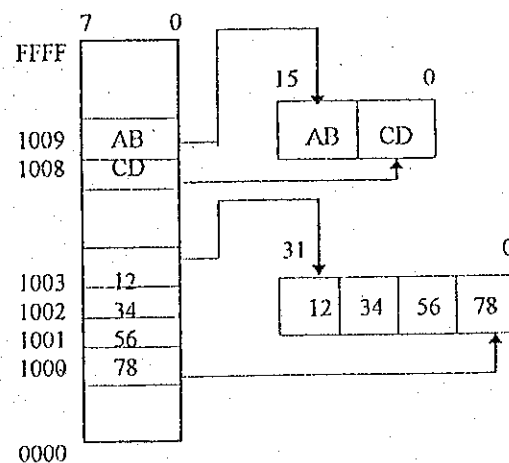
Intel es una empresa que se encarga de fabricar la UCP de muchas PC. Estos son los microprocesadores 8086 y sus derivados (el 8088, 80286, 80386, 80486 y 80586 o pentium). Los ejemplos y prácticos de este laboratorio apuntan a la utilización de los diseños de Intel 80286, 80386 u 80486 por ser éstas las más utilizadas en el mercado actual.

Dependiendo del tipo de procesador, la UCP cuenta con algunos registros, que pueden ser utilizados como si fueran de diferente tamaño. Por ejemplo los registros del 80286 son de 16 bits, pero éstos también pueden ser usados como si fueran de 8 bits. Así, el registro AX de 16 bits o *flip-flops* puede almacenar un número binario de 16 dígitos, o bien uno de 8 dígitos.

Almacenamiento de datos en memoria

En la memoria de la arquitectura en que basamos este laboratorio, usualmente, se “direccionan” octetos, es decir que las locaciones de memoria tienen 8 bits. Como los formatos de datos que puede manejar la UCP puede ser de 8, 16, o 32 bits, cuando se accede a memoria para formatos mayores de 8 bits, se referencia en realidad a un grupo de locaciones almacenadas en 2 o 4 bytes consecutivos.

El almacenamiento en la memoria, ya sea de datos o código de programa, sigue un orden específico. Como se podrá visualizar más adelante, el byte menos significativo se almacena en la dirección más baja y el más significativo en la dirección más alta. Sin embargo, hay que tener en cuenta que, si el dato es numérico, cada octeto se almacena en memoria en forma invertida, ej.: si la representación hexadecimal de una palabra de 16 bits, es ABCD₍₁₆₎, en memoria se almacena primero CD y luego AB. Es decir que la palabra queda almacenada como CDAB, técnica denominada *almacenamiento inverso*. Esto ocurre para cualquier entidad numérica, incluso para datos en representación de punto flotante o para direcciones de memoria.



REPRESENTACIÓN DE BYTES Y PALABRAS EN MEMORIA

Cuando la UCP (por ejemplo una 80286) "direcciona" un byte cuya dirección (*address*) es par, se referencia al byte (n) y al impar que le sigue (n+1). Es decir que por cada orden de lectura se obtienen 16 bits en el data bus. Si por el contrario, se "direcciona" un byte cuya dirección es impar, se referencia al byte impar (n+1) y al par que le sigue (n+2). Este mecanismo se extiende para los procesadores 80386 y 80486 de 32 bits, considerando esta vez 4 octetos.

Existen diferentes formas de almacenar datos en la memoria de una PC.

Si deseamos almacenar texto, por ejemplo la cadena de caracteres *1º de Septiembre* en la memoria de la PC, lo que queda guardado es la representación ASCII de cada uno de ellos. Tenga en cuenta que los caracteres incluyen también a los números y a los caracteres especiales:

31 A7 20 64 65 20 53 65 70 74 69 65 6D 62 72 65
1 º d e S e p t i e m b r e

Las formas de almacenar números en una PC pueden ser:

- en un formato que permita representar números BCD (Decimales Codificados en Binario),
- en un formato que permita representar números binarios enteros (con o sin signo), o
- en un formato que permita representar números binarios reales (enteros y fraccionarios).

La notación BCD se usa para representar números enteros y evitar errores de redondeo en las operaciones. Utiliza grupos de 4 bits para representar cada dígito decimal del 0 al 9, siendo estos valores equivalentes a la representación hexadecimal de cada número de acuerdo con la tabla ASCII:

0 = 0000,

1 = 0001,

2 = 0010,

9 = 1001.

Si deseamos almacenar en la memoria el número 592 en la modalidad BCD desempaquetado (o su equivalente ASCII), éste ocuparía un byte por cada dígito, pero si se almacena en BCD en la modalidad empaquetado (o lo que es lo mismo, equivalente hexadecimal) se almacenaría como:

0011 0101 0011 1001 0011 0010
 5 9 2
 Desempaquetado

0101 1001 0010
 5 9 2
 Empaquetado

La representación de números binarios enteros mejora el uso de la memoria, ya que permite con un octeto un rango de representación de 0 a 255 para números sin signo y de -128 a +127 para enteros signados, en contraposición con la notación BCD que permite representar como máximo hasta el número 99.

La representación de números en el formato de punto flotante es la que resulta óptima, ya que permite manejar números muy grandes o muy chicos, basándose en lo que conocemos como notación científica. El número se representa en dos partes: la mantisa y el exponente. Por ejemplo, para representar el número 1 578 000 lo escribiríamos $0.1578 * 10^7$. Para representar el número decimal 0.0000125 lo escribiríamos como $0.125 * 10^{-4}$. La mantisa es un número entre 0.1 y 0.99.....9 y constituye los dígitos significativos del número, mientras que el exponente nos dice dónde posicionar el punto decimal. En el primer ejemplo la mantisa es 0.1578 y el exponente +7 nos indica que debemos mover el punto decimal siete lugares hacia la derecha para obtener el valor real del número. En el segundo ejemplo la mantisa es 0.125, y el exponente -4 nos indica que debemos mover el punto decimal 4 lugares hacia la izquierda. Lo mismo podemos realizar con los número binarios.

Cualquier número binario puede ser representado en el formato mantisa-exponente.

Una norma del IEEE (Institute of Electrical and Electronics Engineers) pretende normalizar el uso del formato interno para este tipo de representaciones, dado que, generalmente, muchos lenguajes de programación utilizan su propio formato.

Direcciones de memoria en modo real

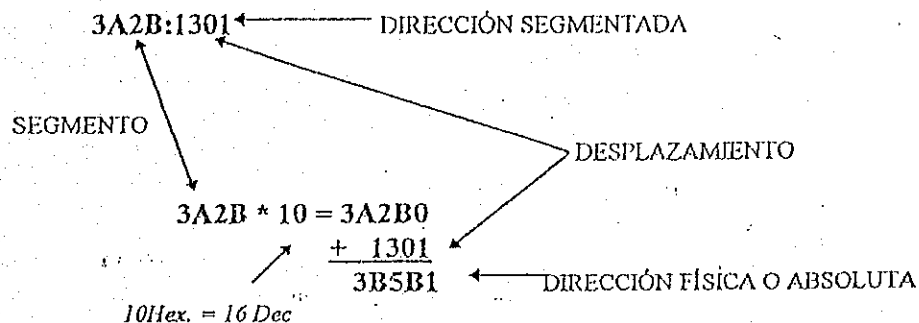
Cuando su PC utiliza como UCP la línea i286/i386/i486 usted puede elegir entre dos modos de operación:

- modo real y
- modo protegido.

Si opera en *modo real* su PC es compatible con el PC original, de modo que los programas originales de los primeros PC XT son perfectamente ejecutables en los actuales, con la ventaja de procesarlos a mayor velocidad.

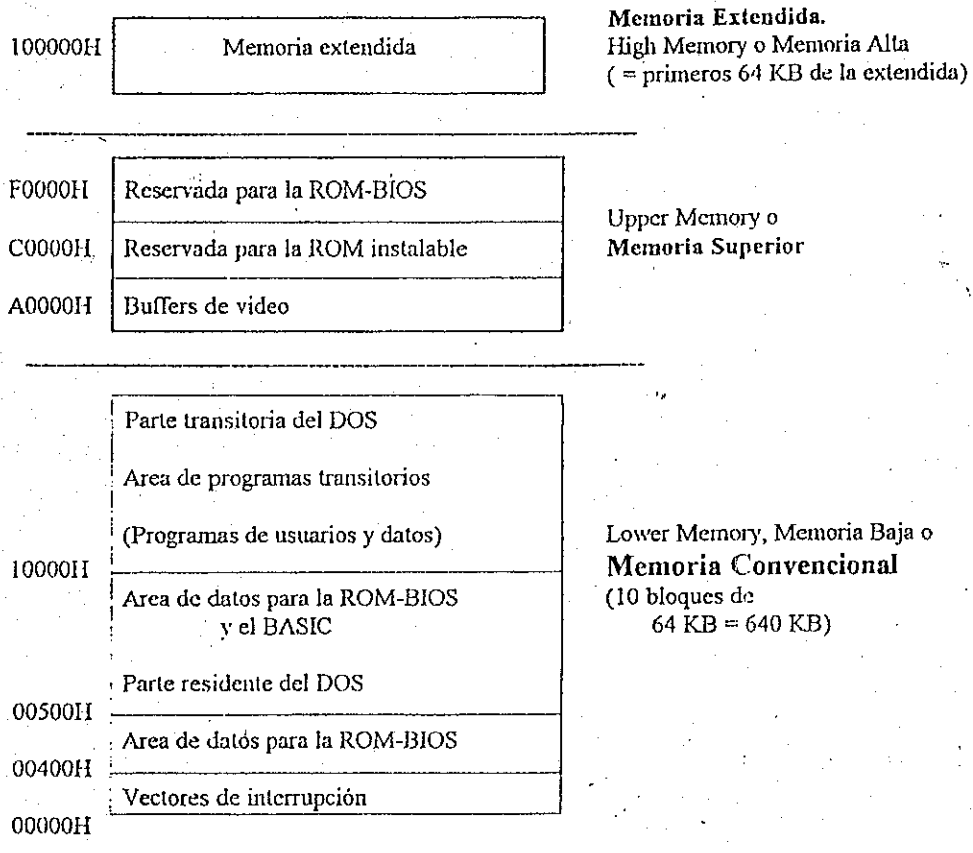
La palabra *direccionamiento* se utiliza para describir técnicas que permiten acceder a un dato, generalmente, en memoria principal. Una dirección de memoria se representa con un número binario que permite identificar cada octeto dentro de la misma. Cuando la cantidad de bits de la dirección es 16, el rango de direccionamiento queda limitado a $(2^{16} - 1) = 65535$, que es la dirección del último octeto direccionable, es decir, que de 0 a 65535 hay 65536 octetos direccionables. Se deduce que la capacidad de memoria total es, entonces, de 64 KBytes (65536 octetos). Sin embargo, esto no es del todo cierto, un PC con UCP de 16 bits (ej., i286) permite acceder a 1 MB ($1.048.576 = 2^{20}$). Esto se logra utilizando 2 palabras de 16 bits y un algoritmo que las relacione. La primera palabra indica "una parte" de la memoria de 1 MB, la segunda, indica el *desplazamiento* del octeto dentro de "esa parte"; luego, la capacidad de cada "parte" de memoria es de 64 KB. Si el total de la memoria es 1 MB y cada parte es de 64 KB, cuántas partes habrá?. Hay 16 partes ($1024 \text{ KB} / 64 \text{ KB}$). A cada una de estas "partes" se las denomina *segmento*. La primera palabra indica donde comienza el segmento y la segunda la posición del octeto dentro del segmento. Así un octeto de memoria puede ser indicado como 3A2B:1301, esta expresión se denomina *dirección segmentada*.

La pregunta ahora es ¿cómo se calcula la dirección física real?. Se sabe que 1 Mega es $1 \text{ Kb} * 1 \text{ Kb} = 2^{10} * 2^{10} = 2^{20}$, esto implica que con 20 bits se calcula la dirección de la palabra mayor ($2^{20} - 1$). El algoritmo debe permitir generar con dos palabras de 16 bits una entidad de 20 bits. Esto se logra así: a 3A2B₍₁₆₎ se le agrega un 0₍₁₆₎ o lo que es lo mismo se lo multiplica por 10₍₁₆₎. A este valor se le suma 1301₍₁₆₎ que es el desplazamiento dentro del segmento. Entonces 3B5B1 es la *dirección física* del octeto en memoria y se conoce como *dirección absoluta*.



CALCULO DE LA DIRECCIÓN FÍSICA O ABSOLUTA EN MODO REAL

El mapa de memoria representa su espacio de direcciones y es, en realidad una vista imaginaria de la memoria, un mapa de cómo los programas la ven. Los límites entre los distintos segmentos no son físicos sino lógicos y pueden empezar y terminar en cualquier octeto cuya dirección sea múltiplo de 16. El siguiente mapa presenta una estructura de 16 bloques (0-F) y es sólo una de las tantas vistas que podemos tener de la memoria.



MAPA DE MEMORIA

Los bloques 0 a 9 se consideran área de memoria para el usuario (RAM = 640 KB); el resto está al servicio del propio ordenador. Sin embargo, el área de memoria de usuario deja las direcciones 00000 hasta la 00600 aproximadamente, para que sean utilizadas como área de datos para el propio software de base del computador; el resto de los 640 KB quedan disponibles para los programas y datos de usuario y algunos programas del SO que quedarán residiendo mientras se ejecute un proceso. Los bloques A y B se destinan para direccionar las plaquetas adaptadoras de video, que permiten entre otras cosas, almacenar temporalmente los caracteres de visualización en pantalla (buffers de video). Los bloques C D y E quedan disponibles para instalar otros módulos, por ejemplo: si se añade un disco rígido, la plaqueta adaptadora puede quedar "direccionada" en una parte del bloque C. El bloque F está asignado para "direccionar" una memoria permanente que se denomina ROM-BIOS (servicios básicos de E/S); sus funciones se pueden agrupar en tres tipos de prestaciones: la primera permite poner en

funcionamiento el computador cuando se lo enciende; la segunda parte son programas de apoyo para controlar la E/S de información desde y hacia los periféricos; y la tercera son las rutinas de interpretación del lenguaje BASIC (ROM- BASIC).

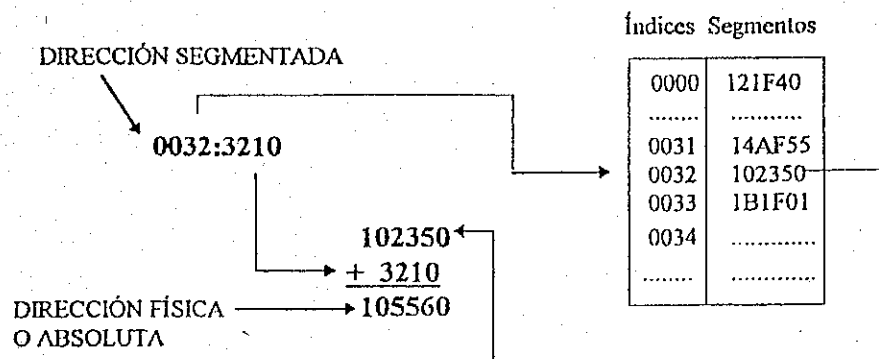
Direcciones de memoria en modo protegido

Estos microprocesadores tienen potencia extra y características que les permiten superar ampliamente a sus antecesores. Para aprovechar esta potencia se define un modo de operación con mayores ventajas que se conoce como *modo protegido* y que permite entre otras cosas:

- protección, no permite a un programa que invada zonas reservadas a otro programa;
- memoria expandida o ampliada;
- memoria virtual y;
- gestión multitask o multitarea, con la cual puede ejecutar varios programas a la vez.

Si bien, el laboratorio se realizará en modo real, y los temas a tratar no están relacionados con estas características de la gestión de memoria, vamos a indicar solo alguna de estas ventajas.

Definamos que se denomina *memoria extendida* a aquellos bloques que se agregan a partir del bloque F con lo que se supera el límite físico real de 1 Mb de memoria original. Desde el punto de vista físico la memoria es una agrupación de chips cuyo número y capacidad de almacenamiento determina la cantidad de memoria física del computador; agregando más chips se incrementa la capacidad física de almacenamiento. Los programas, no ven la memoria como un conjunto de chips sino como un conjunto de octetos identificables a partir de una única dirección. Por lo tanto, existen para los programas una cantidad determinada de bytes que se pueden "direccionar", limitada por el espacio de direcciones descrito con un máximo de 1 Mb. El esquema de direcciones en modo protegido permite superar este límite. Las direcciones segmentadas en modo protegido se descodifican de manera distinta. La primera palabra no referencia el comienzo de un segmento, sino que representa el índice de una tabla que contiene una dirección. Por ejemplo En el i286 cada ítem de la tabla tenía 24 bits e indicaba el comienzo de un segmento. El algoritmo es el mismo, solo que ahora puede hacer referencia a una palabra ubicada en el la dirección $2^{24}-1$. Es decir, que permite instalar en el computador una capacidad de memoria real de $16 \text{ Mb} = 2^{24}$; así la dirección 0032;3210 será calculada en base a una de las direcciones de comienzo de segmentos contenidas en la tabla.



CALCULO DE LA DIRECCIÓN FÍSICA O ABSOLUTA EN MODO PROTEGIDO

Sin embargo, como el DOS no fue desarrollado pensando en esta característica un PC/AT tiene un límite 4 Mb impuesto por el sistema operativo.

¿Como ver el mapa de memoria?

En un PC i286 con memoria de 1 MB se denomina *memoria convencional* a los 640 KB de uso del usuario. Las primeras direcciones de esta memoria de usuario se utilizan para almacenar ciertos datos del sistema operativo y de la ROM-BIOS. Este área es llamada *memoria baja (Lower Memory)*. Bajo el sistema operativo DOS y en modo real, los programas de aplicación y sus datos, pueden alojarse solo en los bytes restantes, cuyo límite son los 640 KB. Los bytes por encima de esta barrera de 640 KB, se denominan *memoria superior o Upper Memory*, reservada, como ya dijimos, para la instalación de otros módulos y las plaquetas adaptadoras de Video, CGA, EGA, VGA, SUPER VGA, con tamaños de RAM para los caracteres de visualización, en pantalla de 512 Bytes o 1 KB. *High memory o memoria alta* es el primer bloque de 64 KB a partir del primer MB (forma parte de la memoria extendida) y puede usarse para alojar el núcleo del DOS (Kernel) que por omisión se carga en memoria convencional, obteniéndose así, un mejor aprovechamiento de los primeros 640 KB de memoria.

El S.O. DOS provee un comando que brinda información de la memoria de su propio PC. El comando es MEM, pruébelo.

Por ejemplo, C:>MEM/P le permite visualizar el estado de la memoria principal asignada y libre.

C:\>MEM/P

Tipo de memoria	Total	Usada	Libre
Convencional	640K	140K	500K
Superior	91K	91K	0K
Reservada	384K	384K	0K
Extendida (XMS)	15.269K	1.433K	13.836K
Memoria total	16.384K	2.048K	14.336K
Total por debajo	731K	231K	500K

Expandida total (EMS) 14M (14.876.672 bytes)

Expandida libre (EMS) 14M (14.876.672 bytes)

Mayor tamaño de programa ejecutable: 500K (512.272 bytes)

Mayor bloque de memoria superior libre: 0K (0 bytes)

MS-DOS es un programa residente en el área de memoria alta.

¿A que se denomina Memoria Expandida ?

El término *memoria expandida* se refiere a otra técnica para "direccionar" más allá del MB reglamentario, gestionada por un programa del SO llamado EMM (Expanded Memory Manager) o EMS (Expanded Memory System), que divide la memoria extendida en páginas fijas de 16 KB (paginación lógica). Estas páginas se agrupan en bloques de 4 páginas, formando una entidad de 64 KB, en una parte de la memoria alta. La gestión se ve apoyada por una tarjeta especial de hardware denominada above board. El software de gestión EMM fue desarrollado después de un acuerdo entre Lotus, Intel y Microsoft (LIM), para un entorno de trabajo DOS. Actualmente Windows no utiliza *memoria expandida* para poder acceder a la *memoria extendida*.

¿A que se denomina Memoria Virtual?

Por último, otra manera que tiene el modo protegido de asignar más memoria a los programas en el momento de la ejecución es el conocido como **memoria virtual** y se refiere a la cantidad de memoria secundaria (en general: disco magnético) que el

ordenador le proporciona a cada programa para trabajar, simulando así, disponer de una memoria mucho más grande que la que tiene realmente.

El ensamblador

Para que usted entienda como funciona el hardware de su PC, o sea, conocer la operatoria de la misma, es conveniente la utilización del lenguaje *ensamblador*. El *ensamblador* es un lenguaje *simbólico* del lenguaje de máquina, que permite que usted posea el control de todo lo que su PC puede realizar. Por ejemplo, es útil para diseñar rutinas de procedimientos especiales, tales como las que permiten el óptimo el uso del hardware o controlan la entrada/salida de datos y, en general, para desarrollar tareas que a otros lenguajes de programación les resultaría imposible realizar.

Los programas desarrollados en este lenguaje se traducen en menos instrucciones que los escritos en otros lenguajes, como ser Basic, Pascal, etc., lo que significa generar un programa ejecutable (extensión = .EXE) más reducido y, por lo tanto, más rápido (ocupa menor tiempo para cargarlo y ejecutarlo). Es un lenguaje que se encuentra un paso antes del código de máquina (código binario), donde las instrucciones se deben proporcionar con mayor exactitud y a diferencia de los lenguajes de alto nivel, requiere más instrucciones para ejecutar el mismo comando. Recordemos que una instrucción en lenguaje de alto nivel puede ser equivalente a n instrucciones en lenguaje assembler.

El ensamblador es un programa que permite que cada instrucción en código de máquina sea representada por una abreviatura que la represente denominada *mnemónico*. A cada instrucción en lenguaje assembler (lenguaje fuente) le corresponde sólo una en lenguaje de máquina (lenguaje nativo o programa objeto o programa ejecutable).

La operación interna de la PC sigue pasos lógicos, ya que ejecuta las instrucciones en forma secuencial, utilizando para ello la memoria, la UCP, los flags, los registros generales, por ejemplo el acumulador (AX de 16 bits, AL de 8 bits o AH de 8 bits), etc.

Ejemplo 1 : Secuencia de instrucciones de un programa sencillo

SECUENCIA	INSTRUCCIÓN
1	mov ax, [0200]
2	add ax, [0203]
3	mov [0205], ax

La secuencia de instrucciones utilizadas en el ejemplo se encuentran escritas en lenguaje ensamblador y determinan los siguientes pasos: mover al registro **AX** el contenido de la dirección 0200 (paso 1; los corchetes indican una dirección que puede apuntar a un registro o a una dirección de memoria, por lo tanto, se opera sobre el contenido del valor dado y no sobre el valor mismo); sumar al registro **AX** el contenido de la dirección 0203 (paso 2) y finalmente mover el contenido de **AX** a la dirección 0205 (paso 3).

El formato de las instrucciones, en general, se define como *acción, destino, fuente* (en el ejemplo, `mov ax, [0200]`).

		Dirección Memoria	Contenido Memoria
A10002	MOV AX, [0200]	0102 0101 0100	02 00 A1

MAPA DE MEMORIA PARA LA INSTRUCCIÓN MOV AX, [0200]

La forma en que se almacena en la memoria cada instrucción es una cadena de bytes (ver Mapa de memoria para la instrucción ...), siendo los caracteres de la izquierda la representación en código de máquina de la instrucción para el lenguaje assembler. Luego, *A1* es el código de operación correspondiente a la instrucción: `MOV AX`, y *0002* es la *dirección de memoria* (en almacenamiento inverso, por ser un valor numérico) donde se encuentra el operando que se almacenará en AX.

Describamos como se ejecutan las instrucciones del programa, teniendo en cuenta que cada una genera cuatro fases o ciclos que se detallan a continuación.

En la primera fase la UCP lee el *IP* (*Instruction Pointer*), o también conocido como *CP* (*Contador de Programa*), que contiene la dirección de memoria de la siguiente instrucción a ser ejecutada; también suele decirse que apunta a la instrucción que será ejecutada.

Seguidamente, se aloja en la Unidad de Descodificación de la UCP, actualizando el valor del IP con la cantidad de bytes que ocupa la instrucción. En el ejemplo dado representará sumarle 3 al IP, ya que la instrucción ocupa 3 bytes como se desprende del código de operación.

A este primer procedimiento se le conoce como **Ciclo de Búsqueda de la Instrucción** (*Instruction Fetch Cycle*) o Fase Fetch.

Una vez leída la instrucción que va a ser ejecutada, la Unidad de Descodificación descodifica e interpreta la misma, para saber si el o los operandos asociados a la instrucción se hallan en sí misma o tiene que proceder a su obtención. A esta segunda etapa se la conoce como **Ciclo o Fase de Descodificación**.

El siguiente ciclo corresponde a la **Fase de Búsqueda del Operando**. El o los operandos necesarios para la ejecución de la instrucción pueden estar indicados en la misma instrucción; o se puede acceder a la memoria para obtenerlos (ej.: `ADD AX, [0203]`), en este caso la dirección de la memoria se indicará entre corchetes; o bien se accederá a un registro, donde puede estar el operando o una dirección que indique donde se encuentra el mismo.

El último ciclo corresponde a la **Fase de Ejecución**, propiamente dicha, y por lo tanto, ejecuta dicha instrucción.

La suma de las cuatro fases determinan el **Ciclo de Instrucción**, que es equivalente a decir que es el tiempo que tarda en ser ejecutada una instrucción en forma completa.

Luego, la ejecución de un programa será una sucesión de las siguientes actividades:

- lectura de las localidades de memoria "direccionadas" por el registro IP,
- incremento del mismo de acuerdo con lo que se va leyendo,
- descodificación de instrucciones,
- búsqueda de los operandos si fuere necesario y
- ejecución de cada una de ellas.

Cuando se enciende la PC se pone en funcionamiento un proceso conocido como *proceso de carga inicial o bootstrapping*, en el cual el IP se inicia con el valor de una localidad fija de la memoria, que normalmente corresponde a la memoria ROM. Dicha memoria contiene un programa llamado *BIOS* (Basic Input/Output System) que realiza un número de chequeos inherentes a la integridad de la PC y , además, carga desde un disquete o desde el disco rígido el *Sistema Operativo* (generalmente, conocido por DOS: Disk Operating System).

Todos los programas necesitan, para que puedan finalizar con normalidad, una instrucción que les permita regresar el control a quien los "invocó". En el caso del lenguaje assembler esta instrucción es *RET* , que produce el mismo efecto que *INT 20* (Interrupción 20), salvo que genera un código de operación diferente (el código para *RET* es *C3*, mientras que el código para *INT20* es *CD20*).

¿Cómo se puede crear un programa en lenguaje assembler?

Existen dos formas de crear un programa assembler. Una es mediante la utilización de un *ensamblador*, como por ejemplo el *MASM* (Macro Assembler, de Microsoft) y la segunda es utilizar un programa utilitario llamado *Debugger*.

Las etapas para la creación de un programa en assembler, a partir de un ensamblador son:

- la primera generar el archivo o programa fuente, lo cual puede lograrse, por ejemplo, con un editor,
- la segunda ensamblarlo (traducirlo a código de máquina) y enlazarlo (generar las referencias cruzadas).

El Debbuger

Este es el nombre de un programa que posibilita la ejecución de otro programa y el rastreo de los movimientos de información de los registros que afecten las instrucciones del proceso. Permite:

- ejecutar un programa, total o parcialmente,
- ensamblar y ejecutar programas sobre la marcha, y
- ver el contenido de las memorias RAM y ROM.

Debemos enunciar ciertas características para utilizar el Debug como herramienta de programación assembler:

- la memoria se divide en segmentos. Cada segmento no debe ser mayor de 64 kb (resultado de $2^{16} = 65\,536$, dividido entre 1024),
- para acceder a locaciones de memoria se efectúan desplazamientos dentro de los segmentos,
- algunos registros asociados a la UCP pueden definirse como registros de 32, de 16 y/o de 8 bits.

Para poder invocar al debugger ingrese:

C:\>DEBUG <ENTER>

Verá que aparece un guión en la siguiente línea. Este guión le indica que Debug se encuentra esperando algún comando. Para más referencias consulte el manual del MSDOS 5.0 sección DEBUG o el manual de la versión del sistema operativo que tenga instalado. Si quiere visualizar los comando que acepta el debug, tipee:

-??

y aparecerá la siguiente información:

ensamblar	A [dirección]
comparar	C intervalo de direcciones
volcar	D [intervalo]
introducir	E dirección [lista]
llenar	F lista de rango
ir	G [=dirección] [direcciones]
hexadecimal	H valor1 valor2
info/entrada	I puerto
cargar	L [dirección] [unidad] [primersector] [número]
mover	M intervalo de direcciones
nombre	N [ruta\nombre] [arglist]
info/salida	O byte de puerto
continuar	P [=dirección] [número]
-salir	Q (cierra la sesión sin guardar archivo de prueba)
registro	R [registro]
buscar	S lista de rango
seguir	T [=dirección] [valor]
desensamblar	U [intervalo]
escribir	W [dirección] [unidad] [primersector] [número]
asignar-memoria expandida	XA [#páginas]
desasignar-memoria expandida	XD [identificador]
trazar páginas-memoria expandida	XM [Lpágina] [Ppágina] [identificador]
mostrar estado-memoria expandida	XS

Ejemplo 2 : Visualización de una cadena

```
nombre db 'INGRESE SU NOMBRE Y APELLIDO : ', $
mov dx, offset nombre
mov ah, 09h
int 21h
int 20
```

Las instrucciones del ejemplo anterior, en assembler, definen la visualización de la cadena entre apóstrofes, mientras que en Basic, se lograría el mismo resultado con la instrucción PRINT "INGRESE SU NOMBRE Y APELLIDO : ".

DB significa define byte. OFFSET le pide al ensamblador MASM que calcule la *dirección absoluta* (desplazamiento) del símbolo dentro del segmento, en este caso, moverá al registro DX la dirección del símbolo NOMBRE. La instrucción *int 21h* produce la interrupción del procesamiento del programa en estado de ejecución para requerirle a la UCP un servicio del DOS. Como previamente se movió al registro AH el valor 09h, que identifica a la rutina "*PRINT SCREEN STRING*", la UCP ejecutará la rutina correspondiente ubicada, generalmente, en hardware. Para lograrlo, tomará la dirección de inicio de la cadena definida en DX y su terminación que estará dada por el carácter "\$". Para observar que tipo de rutina realiza cada función de la interrupción INT 21, observe el Manual de Referencias de DOS que corresponda a su procesador.

La instrucción "*int 20*" genera una interrupción que finaliza el programa, libera los *buffers* (áreas temporales de memoria) y devuelve el control al Debugger.

Tanto INT 20, como INT 21 se denominan *interrupciones de software*.

Debug sólo puede crear archivos con extensión .COM que no pueden ser mayores de 64 KB y deben comenzar, necesariamente, en el desplazamiento 0100H.

Registros asociados a la UCP

Los registros asociados a la UCP de los microprocesadores 808X y 80X86 se identifican con dos caracteres alfabéticos. El contenido de estos registros puede ser modificado. También podemos alterar el valor del *Registro de estado de las banderas* (F) o *Status Register*. Este registro permite ser consultado para realizar bifurcaciones basadas en el resultado de alguna operación. Por ejemplo, si el resultado de comparar dos registros es cero (resta de ambos), entonces, el resultado debería ser igual a cero y el biestable Z debería tener el valor 1. Esta condición se verifica en el estado de las banderas cuando aparece ZR (lo cual significa que el resultado de la operación es cero).

A continuación se presenta la Tabla de Registros asociados a la UCP.

REF.	NOMBRE	FUNCION
AX	Acumulador	Almacenar resultado de operaciones; lectura/escritura desde/hacia los puertos; área de memoria temporal.
BX	Registro base	Registro apuntador de base o índice.
CX	Registro contador	En iteraciones, como contador que se autoincrementa o se autodecrementa, según el tipo de instrucción.
DX	Registro de datos	Puente para el acceso de datos (ej.: identifica un port -puerto- de datos).
DS	Registro de segmento de datos	El contenido de este registro (es una dirección) apunta a un área de memoria separada para datos.
ES	Registro de segmento extra	El contenido de este registro (es una dirección) apunta a un área de memoria 'extra' separada para datos.
SS	Registro de segmento de pila	El contenido de este registro (es una dirección) apunta a un área de memoria separada para la pila (stack).
S	Registro de segmento de código	El contenido de este registro (dirección) apunta a un área de memoria separada para almacenar el código ejecutable de cada programa.
BP	Registro puntero de base	Para manipular la pila sin afectar al registro de segmento SS.
SI	Registro de índice fuente	El contenido de este registro (es una dirección) apunta a un área de memoria donde se encuentra un vector de datos.
DI	Registro de índice destino	El contenido de este registro (es una dirección) apunta al área de memoria donde se copiará un vector de datos.
SP	Registro puntero de la pila	El 'stack pointer' es un registro cuyo contenido apunta al último registro (o plato) utilizado de la pila en uso.
IP	Registro apuntador de la siguiente instrucción	El contenido de este registro apunta a la dirección de memoria de la siguiente instrucción que será ejecutada.

A continuación se describe el significado de cada flag del registro de banderas.

BANDERA	BANDERAS APAGADAS	BANDERAS PRENDIDAS
Overflow	NV=no hay desbordamiento;	OV=hay desbordamiento;
Direction	UP=hacia adelante;	DN=hacia atrás;
Interrupts	DI=desactivadas;	EI=activadas;
Sign	PL=positivo;	NG=negativo;
Zero	NZ=no es cero;	ZR= es cero;
Auxiliary Carry	NA=no hay acarreo aux.;	AC=hay acarreo aux;
Parity	PO=paridad non;	PE=paridad par;
Carry	NC=no hay acarreo;	CY= hay acarreo.

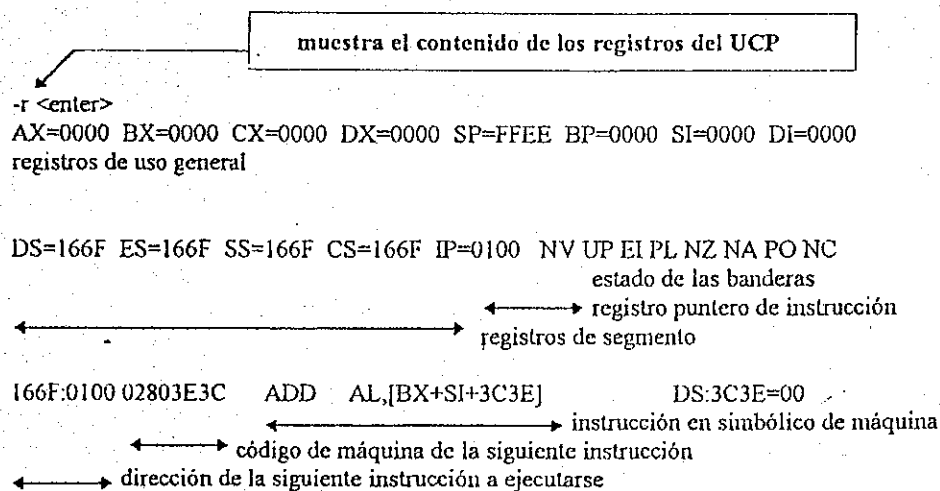
El registro de banderas es un registro de 16 bits, pero no se usan todos los bits.

Para examinar o modificar uno o más registros de la UCP, ingrese:

- r <Enter>

La sintaxis de este comando es r [registro], donde [registro] son los caracteres que identifican al registro de la UCP. Si el nombre del registro no se indica, Debug mostrará el contenido de todos ellos. Si, por el contrario, el nombre del registro es indicado, su contenido se visualizará en hexadecimal y en la línea siguiente aparecerán dos puntos [:] que le permitirán cambiar el valor del registro ingresando el nuevo valor, o presionar <Enter> para dejar el valor que ya poseía. En el caso de querer examinar el registro F, se debe ingresar -rf. Debug mostrará el estado de todas las banderas y éstas podrán ser modificadas por su nuevo valor o saltadas, si es que no se desea cambiarlo, oprimiendo la barra espaciadora.

A continuación se muestran algunos ejemplos:



-r muestra el total de los registros de uso interno de la UCP y es probable que el contenido de los mismos sea diferente.

AX, BX, CX y DX son registros de uso general y son los únicos que pueden usarse de modo dual, o sea, tomándolos como de 8 bits o como de 16 bits; por lo tanto, cada uno de ellos puede ser dividido en dos partes. Por ejemplo, AX se puede dividir en AH (byte alto -High-) y en AL (byte bajo -Low-).

Veamos qué pasa cuando se modifica el contenido en alguno de los registros antes mencionados. En este caso se operará sobre el registro AX.

-rax ;(muestra el contenido del registro AX)
AX 0000
:1A ;(nuevo valor asignado al registro, digitado después del signo :)
-r ;(muestra el estado de todos los registros para verificar el cambio)

AX=001A BX=0000 CX=0000 DX=0000 SP=FFEE BP=0000 SI=0000 DI=0000
DS=166F ES=166F SS=166F CS=166F IP=0100 NV UP EI PL NZ NA PO NC
166F:0100 07 POP ES

Como podemos observar el cambio fue realizado.

Programando en assembler a partir del Debugger

El comando (A)ssembler permite ingresar instrucciones assembler, que son traducidas a medida que son ingresadas. Es decir que bajo debug, usted podrá emular una tarea de programación assembler. Su sintaxis es A [dirección], siendo [dirección] es la ubicación inicial en memoria donde se empezará a almacenar el código de instrucción del programa. De no ser especificada la dirección inicial, el ensamblador generará el código a partir de la localidad estipulada por CS:IP.

A continuación se presentan algunos ejemplos para que usted pueda empezar a familiarizarse sobre cual es el resultado que producen algunas instrucciones de la programación assembler y, también, se proponen prácticos para que ejercite sobre su PC. Como es lógico, a medida que se vaya utilizando, tanto comandos del Debugger, como instrucciones assembler, se hará una breve explicación de como funciona cada uno de ellos.

Habiendo ingresado al Debugger, ingrese A <Enter>. Debería visualizar algo semejante a lo siguiente:

-a
149B:0100_

Práctico 1 : Suma y resta de constantes con resultado positivo, utilizando la parte alta (AH) del registro AX

```
mov ah,8 <Enter>
add ah,3 <Enter>
sub ah,4 <Enter>
Enter>
```

Tomando en cuenta que el segmento y el desplazamiento pueden ser diferentes en su PC, debería ver:

```
-a 100
149B:0100 mov AH,08
149B:0102 add AH,03
149B:0105 sub AH,04
149B:0108 int 20
149B:010A
```

convoca la interpretación del
assembler a partir de la posición 100

Si desea visualizar el programa que ingresó ejecute el siguiente comando "u100 10A":

convoca la interpretación del código de máquina desde la dirección 100 hasta la 10A

-u100 10A
149B:0100 B408 MOV AH,08
149B:0102 80C403 ADD AH,03
149B:0105 80EC04 SUB AH,04
149B:0108 CD20 INT 20

El comando (U) permite desensamblar el código de máquina generado por el ensamblador, o lo que es lo mismo visualizar las instrucciones en lenguaje simbólico, además de la dirección de memoria -segmento:desplazamiento- y el código de máquina en hexadecimal. Este comando se puede solicitar de dos maneras:

- acompañándolo de la dirección desde/hasta (inclusive) que deseamos desensamblar - es el caso de "-u100 10A" -, o bien,
- dándole al comando la cantidad de bytes y la dirección a partir de la cual necesitamos que desensamble. De esta manera el comando -u estaría acompañado por la dirección l de memoria y por la longitud (length), en bytes que queremos desensamblar. Para el ejemplo anterior sería: -u 100 L A (A = 10 en hexadecimal).

Como mencionamos anteriormente, las instrucciones, en general, tienen el siguiente formato: *mnemónico, destino, fuente*, donde:

- mnemónico son 2 ó 3 letras que identifican a la instrucción,
- destino, es la dirección de una locación de memoria o de un registro, o el nombre de un registro donde se alojará el dato luego de ser operado,
- y fuente, es la dirección de una locación de memoria o de un registro, o el nombre de y fuente un registro donde se aloja el dato a ser operado.

En el caso que el destino o el fuente fueren direcciones, ya sea de memoria, o de registros, éstas tienen que ir encerradas entre corchetes "[]".

El programa anterior consta de cuatro líneas, cada una de ellas hace referencia a una instrucción diferente y le ordena a la UCP que ejecute distintas acciones.

MOV es el mnemónico de la instrucción MOVE (mover) que se utiliza para copiar información de un lugar a otro, ya sea entre registros, o de memoria a registro y viceversa. En el ejemplo, 08 es el dato fuente y será copiado en AH que es el registro de destino.

ADD es el mnemónico de la instrucción que permite la adición (addition). En este caso 03 será sumado al contenido anterior del registro AH, y el resultado quedará alojado en AH ($AH = AH + 03$).

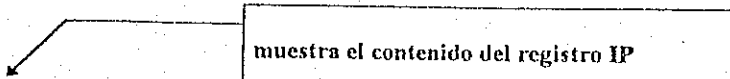
SUB es el mnemónico correspondiente a la resta o sustracción (subtract) y le indica que el valor 04 será restado del contenido del registro AH ($AH = AH - 4$).

Por último, INT 20 indica el fin del programa y le devuelve el control al Debug.

Odipert. vxd

Para poder verificar paso a paso el accionar de las instrucciones del programa anterior sobre cada uno de los registros, existe el comando TRACE (t). Este permite cada vez que se ingresa, ejecutar una instrucción de programa y visualizar el contenido de los registros. Tome en cuenta, antes de utilizarlo, que deberá verificar la dirección de inicio del programa en el registro IP; si no es la correcta deberá modificarla.

C:\>DEBUG



```
-r ip
IP 0100
:
-T

AX=0800 BX=0000 CX=0000 DX=0000 SP=FFEE BP=0000 SI=0000 DI=0000
DS=149B ES=149B SS=149B CS=149B IP=0102 NV UP EI PL NZ NA PO NC
149B:0102 80C403  ADD  AH,03
-T

AX=0B00 BX=0000 CX=0000 DX=0000 SP=FFEE BP=0000 SI=0000 DI=0000
DS=149B ES=149B SS=149B CS=149B IP=0105 NV UP EI PL NZ NA PO NC
149B:0105 80EC04  SUB  AH,04
-T

AX=0700 BX=0000 CX=0000 DX=0000 SP=FFEE BP=0000 SI=0000 DI=0000
DS=149B ES=149B SS=149B CS=149B IP=0108 NV UP EI PL NZ NA PO NC
149B:0108 CD20    INT  20
-q
```

- El comando q permite salir del debug sin guardar el archivo de prueba que se generó.

Práctico 2 :

Suma y resta de constantes con resultado negativo, utilizando la parte alta (AH) del registro AX.

1. Realice un programa que permita sumar dos constantes y restar la constante C, generando un resultado negativo ($AH = 8 + 3 - C$).
2. Justifique el valor de AH.
3. Represente el mapa parcial de memoria donde se aloja el último programa.
4. Haga el trace y represente el mapa parcial de memoria para el programa del Ejemplo 1, asignando los siguientes valores: $[0200] = 8$ y $[0203] = 3$.

Multiplicaciones

Los procesadores invocados en este laboratorio permiten dos tipos de instrucciones para la multiplicación:

MUL que multiplica entidades sin signo, e

IMUL que multiplica entidades con signo.

El multiplicando debe estar en el registro AL, si es de 8 bits (un byte) y en el caso de ser de 16 bits (una palabra) debe alojarse en AX.

El multiplicador se supone del mismo formato que el multiplicando y debe ser movido al registro CL o CX.

El resultado se obtendrá sobre AX si las entidades son de 8 bits y en el par conformado por los registros DX:AX si las entidades son de 16 bits.

$AL \cdot CL \rightarrow AX$

Ejemplo 3.1 :

Multiplicación de entidades enteras de 8 bits sin signo

```
C:\>debug
-a100
2974:0100 mov al,3
2974:0102 mov cl,4
2974:0104 mul, cl
2974:0106 int 20
2974:0108
-g106
```

```
AX=000C BX=0000 CX=0004 DX=0000 SP=FFEE BP=0000 SI=0000 DI=0000
DS=2974 ES=2974 SS=2974 CS=2974 IP=0106 NV UP EI PL NZ NA PE NC
2974:0106 CD20      INT    20
-q
```

Como resultado de la multiplicación de 3 por 4 se obtuvo en el registro AX una C, lo que es correcto, ya que en hexadecimal C es igual a 12 en decimal y en el registro CX el valor 4 que correspondía al multiplicador.

Ejemplo 3.2 :

Multiplicación de 2 entidades enteras de 8 bits sin signo (valores máximos)

```
C:\>debug
-A
2974:0100 mov al,ff
2974:0102 mov cl,ff
2974:0104 mul cl
2974:0106 int 20
2974:0108
-t
```

```
AX=00FF BX=0000 CX=0000 DX=0000 SP=FFEE BP=0000 SI=0000 DI=0000
DS=2974 ES=2974 SS=2974 CS=2974 IP=0102 NV UP EI PL NZ NA PO NC
2974:0102 B1FF      MOV    CL,FF
-t
```

```
AX=00FF BX=0000 CX=00FF DX=0000 SP=FFEE BP=0000 SI=0000 DI=0000
DS=2974 ES=2974 SS=2974 CS=2974 IP=0104 NV UP EI PL NZ NA PO NC
2974:0104 F6E1 MUL CL
-t
```

```
AX=FE01 BX=0000 CX=00FF DX=0000 SP=FFEE BP=0000 SI=0000 DI=0000
DS=2974 ES=2974 SS=2974 CS=2974 IP=0106 OV UP EI NG NZ AC PE CY
2974:0106 CD20 INT 20
-q
```

El producto de multiplicar los valores máximos que pueden soportar tanto AL como CL, siendo estos últimos de 255 en decimal o su equivalente FF en hexadecimal, es de 65.025 en decimal o su equivalente FED1 en hexadecimal. Esto último se verifica observando el valor del registro AX después del último trace.

Ejemplo 3.3 : Multiplicación de dos entidades enteras de 8 bits con signo

Imul permite la multiplicación entre dos entidades de 8 o 16 bits con signo, considerándose negativa una cantidad cuyo bit más significativo sea igual a 1.

```
C:\>debug
-a
2974:0100 mov al,4
2974:0102 mov cl,-1
2974:0104 imul cl
2974:0106 int 20
2974:0108
-g0106
```

```
AX=FFFC BX=0000 CX=00FF DX=0000 SP=FFEE BP=0000 SI=0000 DI=0000
DS=2974 ES=2974 SS=2974 CS=2974 IP=0106 NV UP EI NG NZ NA PO NC
2974:0106 CD20 INT 20
-q
```

De la ejecución del programa se verifica que el resultado correcto se encuentra alojado en AL, siendo éste igual a FC en hexadecimal, o sea -4 en decimal; y el multiplicador se encuentra almacenado en CL (FF_h = -1_d).

Práctico 3 :

Multiplicación de dos entidades enteras de 16 bits, con y sin signo

1. Desarrolle el programa que posibilite la multiplicación de dos entidades sin signo que contienen el máximo valor permisible en 16 bits (65.535). El resultado deberá ser 4.294.836.225 en decimal, equivalente a FFFE0001 hexadecimal. Examine el contenido del par DX:AX.
2. Desarrolle el programa que permita la multiplicación de las siguientes entidades de 16 bits con signo: 4 * -1. Verifique el contenido del par DX:AX.

Divisiones

En las divisiones, también existen dos tipos de instrucciones:

DIV que divide dos entidades sin signo, y

IDIV que divide dos entidades con signo.

En una división de entidades enteras, se supone que el dividendo siempre será mayor que el divisor. Por consiguiente, para un divisor de 8 bits el dividendo será de 16 bits y se alojará en AX; mientras que para un divisor de 16 bits el dividendo será de 32 bits y se almacenará en el par conformado por los registros DX:AX, siendo el bit más significativo de DX el más significativo del par y, a su vez, el bit menos significativo de AX el menos significativo de la doble palabra.

El divisor, si es un byte, deberá siempre estar en el registro BL, mientras que si tiene la longitud de una palabra deberá permanecer en BX.

El cociente, cuando el divisor es de 8 bits (byte), se almacenará en el registro AL y el resto o residuo en AH, mientras que si el divisor es de 16 bits (una palabra) el cociente se almacenará en AX y el resto en DX.

Ejemplo 4 :

División de dos entidades enteras de 8 bits sin signo

```
C:\>debug
-a100
2974:0100 mov ax,7
2974:0103 mov bl,3
2974:0105 div bl
2974:0107 int 20
2974:0109
-q
```

Práctico 4 :

División de dos entidades enteras

1. Ejecute el programa del ejemplo 4 con trace y analice los registros AL y AH.
2. Desarrolle un programa donde el dividendo sea igual a 1C y el divisor a -1 y analice los registros AL y AH.
3. Desarrolle el programa que posibilite la división de las siguientes entidades: FFFF y 2, considerando a BX como el registro que contiene el divisor de 16 bits. Examine el contenido del par DX:AX.
4. Desarrolle el programa que permita dividir FFFF por -1 y analice el contenido del par de registros DX:AX

Desplazamiento y rotación de bits en un registro

En un registro de 16 bits, por ejemplo, se pueden albergar 2 bytes o una palabra. En cualesquiera de los casos, los bits del registro se pueden desplazar o rotar dentro de la

misma entidad. El desplazamiento mueve los bits del registro en forma lineal, mientras que una rotación lo hace en forma circular.

Las instrucciones que permiten un desplazamiento de bits en el registro son:

SHL (shift left: desplazamiento lógico a la izquierda),

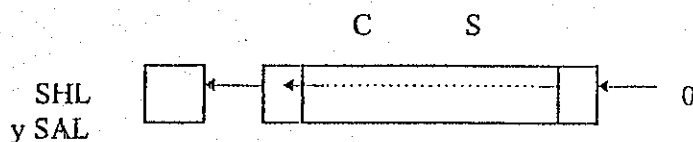
SHR (shift right: desplazamiento lógico a la derecha),

SAL (shift arithmetic left: desplazamiento aritmético a la izquierda, para valores signados), y

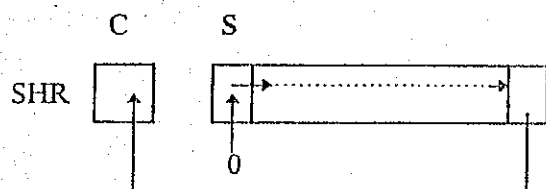
SAR (shift arithmetic right: desplazamiento aritmético a la derecha para valores signados).

Los desplazamientos lógicos no permiten conservar el signo del operando, razón por la cual para valores signados se deberán realizar desplazamientos aritméticos.

SHL y SAL desplazan todos los bits hacia la izquierda, menos el bit más significativo (bit 15 en el caso de una palabra y bit 7 en el caso de un byte) es insertado en el bit de acarreo del registro de banderas (Status Register). A su vez, se coloca un cero en el bit menos significativo (bit 0 en ambos casos), ya sea una palabra o un byte.

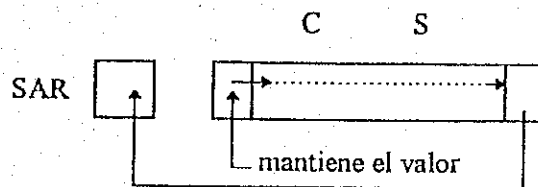


SHR desplaza todos los bits hacia la derecha, menos el bit menos significativo (bit 0), que lo inserta en el bit de acarreo y coloca un cero en el bit más significativo.



SAR desplaza todos los bits a la derecha, menos el bit menos significativo (bit 0), que lo inserta en el bit de acarreo y no modifica el bit más significativo (bit de signo), o sea, mantiene el valor del signo.

El desplazamiento aritmético a izquierda multiplica el contenido del registro por la base, en este caso por 2; mientras que el desplazamiento aritmético a derecha divide el contenido del registro por la base. En el caso de la instrucción SAR, la división puede producir un resto, lo que provoca un redondeo 'por defecto' del resultado. Si el número es positivo trunca el resto y si el número es negativo incrementa el resultado al número entero superior (ej.: si el resultado es -1,5 deja en registro el valor 2).



Ejemplo 5.1 :

Multiplica el contenido de un registro de 16 bits por la base

C:\>debug

-a

2974:0100 mov ax,a0

2974:0103 shl ax,1

2974:0105 int 20

2974:0107

-t

AX=00A0 BX=0000 CX=0000 DX=0000 SP=FFEE BP=0000 SI=0000 DI=0000
DS=2974 ES=2974 SS=2974 CS=2974 IP=0103 NV UP EI PL NZ NA PO NC
2974:0103 D1E0 SHL AX,1

-t

AX=0140 BX=0000 CX=0000 DX=0000 SP=FFEE BP=0000 SI=0000 DI=0000
DS=2974 ES=2974 SS=2974 CS=2974 IP=0105 NV UP EI PL NZ AC PO NC
2974:0105 CD20 INT 20

-q

El contenido del registro AX es A0 en hexadecimal, equivalente a 10100000 en binario y a 160 decimal. Luego de la instrucción SHL, sobre el registro AX encontramos el valor 0140h, equivalente a 101000000b y a 320d. Esto es equivalente a multiplicar el valor del byte o palabra por 2. Observe que la única bandera que ha modificado su valor es la de acarreo auxiliar, por efecto del propio desplazamiento.

Las instrucciones de desplazamiento deben tener un *contador de desplazamiento*. Si este contador está implícito significa que se hará sólo un desplazamiento; en caso contrario la cantidad de desplazamientos deberá ser cargada previamente en el registro CL.

Ejemplo 5.2 :

Divide el contenido de un registro de 8 bits por una cifra especificada en el registro CL

C:\>debug

-a

2974:0100 mov al,8

2974:0102 mov cl,4

2974:0104 shr al,cl

2974:0106 int 20

2974:0108

-t

AX=0008 BX=0000 CX=0000 DX=0000 SP=FFEE BP=0000 SI=0000 DI=0000
DS=2974 ES=2974 SS=2974 CS=2974 IP=0102 NV UP EI PL NZ NA PO NC
2974:0102 B104 MOV CL,04

-t

AX=0008 BX=0000 CX=0004 DX=0000 SP=FFEE BP=0000 SI=0000 DI=0000
 DS=2974 ES=2974 SS=2974 CS=2974 IP=0104 NV UP EI PL NZ NA PO NC
 2974:0104 D2E8 SHR AL,CL
 -t

AX=0000 BX=0000 CX=0004 DX=0000 SP=FFEE BP=0000 SI=0000 DI=0000
 DS=2974 ES=2974 SS=2974 CS=2974 IP=0106 NV UP EI PL ZR AC PE CY
 2974:0106 CD20 INT 20
 -q

Si AL contiene el valor 8d, equivalente a 1000h y se divide por el valor especificado en el registro CL (4), significa que se realizarán 4 corrimientos a derecha, que provocarán que en AX quede el valor cero. Note que se activan las banderas correspondientes a ZERO Y CARRY.

Las instrucciones que permiten rota un bit a la vez en un registro son:

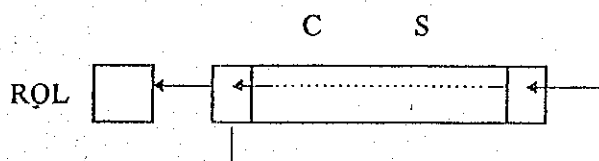
ROL (rotate left: rotación a la izquierda),

ROR (rotate right: rotación a la derecha),

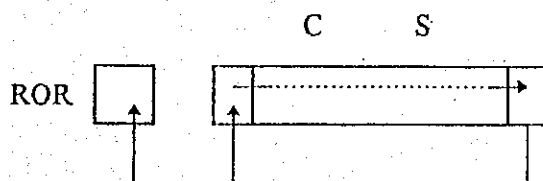
RCL (rotate left thru carry: rotación con acarreo a la izquierda), y

RCR (rotate right thru carry: rotación con acarreo a la derecha).

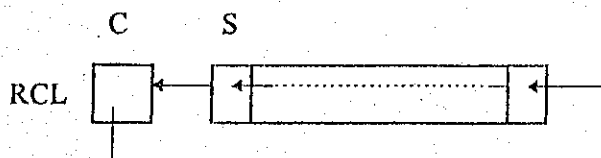
ROL rota los bits del registro, ya sea de 16 u 8 bits, a izquierda, colocando el bit más significativo en el bit de acarreo del registro de banderas y, también, en el bit menos significativo del registro.

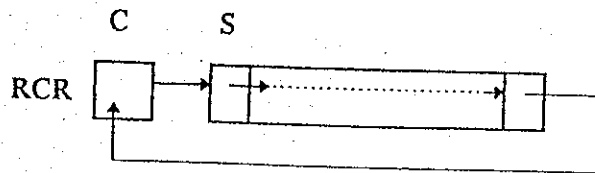


ROR rota los bits del registro a la derecha, insertando el bit menos significativo en el bit de acarreo del registro de banderas y en el bit más significativo del byte o palabra.



RCL y RCR utilizan el bit de acarreo del registro de banderas como una extensión del propio registro durante la rotación. Los bits de estas instrucciones rotan a izquierda o derecha respectivamente, arrastrando con ellos al bit de acarreo.





Práctico 5 : Desplazamientos y acarreos

1. Desplace aritméticamente a derecha el valor A0 en el registro AL y analice el contenido de los registros involucrados.
2. Realice el ejercicio anterior, pero utilice rotación a izquierda.
3. Realice el ejercicio anterior, pero utilice rotación a derecha.
4. Realice el ejercicio anterior, pero utilice rotación con acarreo a izquierda.
5. Realice el ejercicio anterior, pero utilice rotación con acarreo a derecha.

Práctico 6 : Bifurcaciones condicionales, incondicionales e iteraciones

Se puede decir que en casi todos los programas se utilizan instrucciones que permiten saltar de un lugar a otro del programa en forma incondicional (sin ninguna condición de por medio), hacer un salto condicional (esto es, sujeto a alguna condición o evento) o iterar una porción de programa (repetir una secuencia de instrucciones n cantidad de veces). Los ejemplos que se presentan a continuación, pretenden reflejar estas circunstancias y mostrarle que hay distintas formas de arribar a un mismo resultado.

Ejemplo 6.1 : Despliega una cadena 7 veces, utilizando un salto condicional y un registro que lleva la cuenta de cuantas veces fue desplegada.

```
C:\>debug
-a
2974:0100 jmp 130
2974:0102
-e 102 "QUE TENGAS UN FELIZ DIA!" 0d 0a "$"
-a 130
2974:0130 mov bx,7
2974:0133 mov dx,102
2974:0136 mov ah,9
2974:0138 int 21
2974:013A dec bx
2974:013B jnz 138
2974:013D int 20
2974:013F
-g
QUE TENGAS UN FELIZ DIA!
QUE TENGAS UN FELIZ DIA!
QUE TENGAS UN FELIZ DIA!
QUE TENGAS UN FELIZ DIA!
QUE TENGAS UN FELIZ DIA!
QUE TENGAS UN FELIZ DIA!
QUE TENGAS UN FELIZ DIA!
```

El programa ha finalizado con normalidad
-q

Este programa despliega 7 veces la cadena 'QUE TENGAS UN FELIZ DÍA!'.

La primera instrucción JMP (JUMP = salto incondicional) se usa para saltar sobre la definición de la cadena que se realiza a partir de la posición 102. Una cadena, siempre debe definirse entre comillas y necesita de ciertos parámetros:

como luego de ser visualizada la cadena se debe pasar a la siguiente 'línea', después de la cadena deberá ingresar un espacio y el código hexadecimal de control de carro (0d),

a continuación para apuntar a la nueva línea se necesita ingresar, luego de otro espacio, el código hexadecimal de línea nueva (0a) y

por último, para indicarle al ensamblador el final de la cadena, se debe ingresar un espacio y entre comillas el signo \$.

La siguiente instrucción a ejecutarse guarda en el registro BX la cantidad de veces que se desplegará la cadena.

INT 21 es la interrupción de DOS que se encargará de mostrar la cadena y para ello necesita saber cual de las funciones se le solicita. Esto se realiza cargando la dirección inicial de la cadena (102) en el registro DX y el número que identifica a la función (en este caso 9) en el registro AH.

Luego de haberse ejecutado INT 21 se decrementa el registro BX que lleva la cuenta de la cantidad de veces que se mostró la cadena (DEC BX).

La siguiente instrucción es un salto condicionado, salta si no es ZERO (esto quiere decir que salta si el flag Z es distinto de 0) a la dirección 120 para permitir que siga mostrándose la cadena.

Ejemplo 6.2 :

Despliega la cadena 7 veces, pero utiliza el registro CX para controlar la iteración y el salto condicional JCXZ, que salta si CX = 0

```
C:\>debug
-a
2974:0100 jmp 130
2974:0102
-e 102 "QUE TENGAS UN FELIZ DÍA!" 0d 0a "$"
-a 130
2974:0130 mov dx, 102
2974:0133 mov cx, 7
2974:0136 mov ah, 9
2974:0138 int 21
2974:013A dec cx
2974:013B jcxz 13f
2974:013D jmp 138
2974:013F int 20
2974:0141
-g
QUE TENGAS UN FELIZ DÍA!
QUE TENGAS UN FELIZ DÍA!
QUE TENGAS UN FELIZ DÍA!
QUE TENGAS UN FELIZ DÍA!
```

Ejemplo 6.3 :

Despliega la cadena 7 veces, reemplazando el salto condicional y el registro contador por la instrucción LOOP

```
C:\>debug
-a
2974:0100 jmp 130
2974:0102
-c 102 "QUE TENGAS UN FELIZ DIA!" 0d 0a "$"
-a 130
2974:0130 mov cx,7
2974:0133 mov dx,102
2974:0136 mov ah,9
2974:0138 int 21
2974:013A loop 138
2974:013C int 20
2974:013E
-g
QUE TENGAS UN FELIZ DIA!
QUE TENGAS UN FELIZ DIA!
QUE TENGAS UN FELIZ DIA!
QUE TENGAS UN FELIZ DIA!
QUE TENGAS UN FELIZ DIA!
QUE TENGAS UN FELIZ DIA!
QUE TENGAS UN FELIZ DIA!
```

El programa ha finalizado con normalidad
-q

La instrucción LOOP se utiliza para el control de iteraciones. Cada vez que se ejecuta decrementa automáticamente el registro contador CX y salta a la dirección 138 hasta que CX = 0. Con esta instrucción se utilizan dos instrucciones menos que en los ejemplos anteriores.

Práctico 7:**Intercambio de datos entre registros**

La instrucción XCHG cambia el contenido del registro fuente al destino y viceversa.

Ejemplo 7:

Intercambio del contenido de los registros AX y BX

```
C:\DEBUG
-a100
2974:0100 mov ax, 123
2974:0103 mov bx, ff
2974:0106 xchg ax,bx
2974:0108 int 20
2974:010A
-t
```

```
AX=0123 BX=0000 CX=0000 DX=0000 SP=FFEE BP=0000 SI=0000 DI=0000
DS=2974 ES=2974 SS=2974 CS=2974 IP=0103 NV UP EI PL NZ NA PO NC
2974:0103 BBFF00 MOV BX,00FF
```

-t

```
AX=0123 BX=00FF CX=0000 DX=0000 SP=FFEE BP=0000 SI=0000 DI=0000
DS=2974 ES=2974 SS=2974 CS=2974 IP=0106 NV UP EI PL NZ NA PO NC
2974:0106 87C3 XCHG AX,BX
```

-t

```
AX=00FF BX=0123 CX=0000 DX=0000 SP=FFEE BP=0000 SI=0000 DI=0000
DS=2974 ES=2974 SS=2974 CS=2974 IP=0108 NV UP EI PL NZ NA PO NC
2974:0108 CD20 INT 20
```

-q

Luego del último trace, se puede observar el intercambio entre los registros AX y BX.

Práctico 8 :

Almacenamiento de programas

Todo programa que se genera a través del Debugger, lo hace a partir de una locación de memoria. Si esas locaciones se escriben dos veces, por ejemplo, con otro programa, o si el computador se pone en OFF (de alguna manera pierde el suministro de energía), el contenido de la memoria se perderá y, obviamente, también el programa que se ingresó a ella. Para poder resguardar un programa, generado con el assembler del Debug, usted puede almacenarlo en un disco. De esta manera, podrá recuperarlo en otra oportunidad.

Para almacenar un programa en un disco siga los pasos que a continuación se detallan:

una vez ingresado el programa, averigüe la cantidad de bytes que ocupa. Esto se logra restando la dirección de la última instrucción del programa, de la dirección de la primera instrucción del mismo. Estas direcciones se expresan en hexadecimal. El comando H (hexadecimal) permite sumar y restar dos cantidades en hexadecimal. Las cantidades deben ser ingresadas en el orden correspondiente. En el ejemplo 8 se ingresó primero la última dirección y luego la primera. Como resultado se obtuvo en primer lugar la suma de ambas (0201) y a continuación la diferencia (000A) de las mismas;

1. ingrese el nombre de un archivo que identifique al programa que se almacenará en el disco. Esto se logra con el comando N (Name) y a continuación la ruta\nombre de archivo\extensión. Recuerde que la extensión debe ser .COM y que el nombre debe cumplir con las especificaciones del MS-DOS. En el ejemplo 'n a:\programa.com';
2. cargue en el registro CX la cantidad de bytes que ocupa el programa utilizando el comando rcx;
3. de la orden de escritura o grabación. Esto se logra utilizando el comando w (Write).

Una vez efectivizada la grabación, Debug mostrará la siguiente leyenda:

Writing xxx Bytes (escribiendo xxx bytes).

Ejemplo 8 :

Como almacenar un programa en un disco

```
C:\>debug
-a
2974:0100 mov ax,4
2974:0103 mov bx,2
2974:0106 sub ax,bx
2974:0108 int 20
```

```

2974:010A
-h 10A 100
020A 000A
-n a:\programa.com
-rcx
CX 0000
:000A
-w
Escribiendo 0000A bytes
-g108

```

```

AX=0002 BX=0002 CX=000A DX=0000 SP=FFEE BP=0000 SI=0000 DI=0000
DS=2974 ES=2974 SS=2974 CS=2974 IP=0108 NV UP EI PL NZ NA PO NC
2974:0108 CD20 INT 20
-q

```

Práctico 9 : Carga de programas

Si previamente usted almacenó un programa, es lógico que en algún momento, lo quiera volver a ejecutar. Para recuperarlo, o sea, cargarlo desde el disco donde se hallaba resguardado a la memoria, deberá seguir los siguientes pasos:

1. asegúrese que está operando bajo debugger, en caso contrario invóquelo;
2. indique el nombre del programa que se desea cargar utilizando el comando Name (n);
3. dé la orden de carga utilizando el comando Load (L). Debug siempre lo cargará en un segmento que se encuentre disponible y a partir del desplazamiento 0100.

Ejemplo 9 : Como cargar un programa alojado en disco a la memoria

```

C:\>debug
-n a:\programa.com
-L

```

Para verificar el programa cargado y comprobar su funcionamiento, desensamble a partir de la localidad 100H y luego pida el trace de cada una de las instrucciones.

```

-u 100 L 9
299B:0100 B80400 MOV AX,0004
299B:0103 BB0200 MOV BX,0002
299B:0106 29D8 SUB AX,BX
299B:0108 CD20 INT 20
-t
AX=0004 BX=0000 CX=000A DX=0000 SP=FFFE BP=0000 SI=0000 DI=0000
DS=299B ES=299B SS=299B CS=299B IP=0103 NV UP EI PL NZ NA PO NC
299B:0103 BB0200 MOV BX,0002
-t
AX=0004 BX=0002 CX=000A DX=0000 SP=FFFE BP=0000 SI=0000 DI=0000
DS=299B ES=299B SS=299B CS=299B IP=0106 NV UP EI PL NZ NA PO NC
299B:0106 29D8 SUB AX,BX

```

AX=0002 BX=0002 CX=000A DX=0000 SP=FFFE BP=0000 SI=0000 DI=0000
 DS=299B ES=299B SS=299B CS=299B IP=0108 NV UP EI PL NZ NA PO NC
 299B:0108 CD20 INT 20

Práctico 10: Concepto de pila

La pila es una zona de memoria, contenida en el SS (Stack Segment = Segmento de pila), que se estructura con criterio LIFO (Last in, First out = último en entrar, primero en salir). Usted puede imaginar la pila como un conjunto de platos (locaciones de memoria) apilados, donde para sacar el último, primero deberá sacar los que están por encima del mismo comenzando desde el que se encuentra más arriba. Ahora que entendió el concepto de pila, acepte la realidad, en memoria la pila se almacena en forma invertida, o sea, que el último plato que se ingresa a la pila es el que está más abajo, mientras que el primero que se ingresó es el que está más arriba.

Quien se encarga del manejo de la pila es la UCP y se utiliza, generalmente, para:

- almacenar la dirección de retorno (IP) y, eventualmente, el CS (segmento de código) cuando se llama a un procedimiento, también conocido como subrutina,
- almacenar el estado del procesador cuando se produce una interrupción. Alguno de los registros que apila son el CS, el IP y el estado de los flags (Status Register),
- para brindar un espacio de trabajo a un procedimiento,
- para pasar parámetros de un lenguaje de alto nivel al ensamblador.

El manejo de la pila se realiza a través de los registros SP y BP. El SP (Stack Pointer = Puntero de pila) es el registro que contiene la dirección del último elemento de la pila utilizado, por este motivo, cualquier acceso a la pila estará en función de la localidad especificada por SS:SP.

El almacenamiento o extracción de datos de la pila no es un procedimiento físico. Estas operaciones se simulan incrementando o decrementando el registro SP. Como la pila trabaja a nivel palabra (2 bytes), cada vez que usted opere con la misma, se autoincrementará ($SP = SP + 2$) o se autodecrementará ($SP = SP - 2$) en 2 unidades (posiciones de memoria). Esto nos lleva a pensar, que si el SP apunta al último registro de la pila utilizado, podemos decir que la pila crece de las locaciones más altas de memoria a las más bajas, esto es en sentido inverso.

Las instrucciones que, automáticamente, utilizan la pila son: CALL, INT, RET e IRET. Call y Ret son instrucciones que sirven para invocar y dar el retorno a un procedimiento o subrutina, mientras que Int e Iret cumplen la misma función cuando se invoca una interrupción.

Las instrucciones que permiten el manejo de la pila son:

PUSH que empuja o produce el almacenamiento de la palabra en la pila y luego decrementa el SP, y

POP que extrae o saca una palabra de la pila y desafecta las posiciones de memoria incrementando el SP.

En el programa del ejemplo 10, usted podrá observar que las primeras acciones corresponden a reemplazar el valor de las localidades -offset- de memoria 1100 y 1101 por los valores hexadecimales D7 y D8 respectivamente. En realidad estos comandos no inciden en el manejo de la pila, pero le servirá para visualizar, más adelante, como se transfieren esos contenidos de memoria (pertenecientes al SD = Segmento de datos) a la pila (SS = Segmento de pila).

Las tres primeras instrucciones del programa transfieren a los registros AX, BX y CX los valores hexadecimales D5D6, D3D4 y D1D2 respectivamente. A partir de este momento es cuando usted podrá ponerse en contacto con el movimiento de la pila.

Las cuatros instrucciones que siguen empujan a la pila el contenido de la dirección de memoria [1100] -y por arrastre la [1101]- y luego el contenido de los registros AX, BX y CX con sucesivos PUSH. Cuando posteriormente, usted analice con trace el programa verá que estos pasos se fueron cumplimentando sin lugar a errores.

Las subsecuentes instrucciones POP, extraen de la pila los cuatro valores almacenados en ella (recuerde que tiene que pensarlos como palabras invertidas) y los depositan en los registros AX, BX, CX y DX.

La última instrucción INT 20 le devuelve el control al Debug.

Ejemplo 10 :

El manejo de la Pila: PUSH y POP

```
C:\>debug
-e1100
2974:1100 26.D7
-e1101
2974:1101 83.D8
-a
2974:0100 mov ax,D5D6
2974:0103 mov bx,D3D4
2974:0106 mov cx,D1D2
2974:0109 push [1100]
2974:010D push ax
2974:010E push bx
2974:010F push cx
2974:0110 pop ax
2974:0111 pop bx
2974:0112 pop cx
2974:0113 pop dx
2974:0114 int 20
2974:0116
-t
```

```
AX=D5D6 BX=0000 CX=0000 DX=0000 SP=FFEE BP=0000 SI=0000 DI=0000
DS=2974 ES=2974 SS=2974 CS=2974 IP=0103 NV UP EI PL NZ NA PO NC
2974:0103 BBD4D3 MOV BX,D3D4
-t
```

```
AX=D5D6 BX=D3D4 CX=0000 DX=0000 SP=FFEE BP=0000 SI=0000 DI=0000
DS=2974 ES=2974 SS=2974 CS=2974 IP=0106 NV UP EI PL NZ NA PO NC
2974:0106 B9D2D1 MOV CX,D1D2
-t
```

```
AX=D5D6 BX=D3D4 CX=D1D2 DX=0000 SP=FFEE BP=0000 SI=0000 DI=0000
DS=2974 ES=2974 SS=2974 CS=2974 IP=0109 NV UP EI PL NZ NA PO NC
2974:0109 FF360011 PUSH [1100] DS:1100=D8D7
-t
```

```
AX=D5D6 BX=D3D4 CX=D1D2 DX=0000 SP=FFEC BP=0000 SI=0000 DI=0000
DS=2974 ES=2974 SS=2974 CS=2974 IP=010D NV UP EI PL NZ NA PO NC
2974:010D 50 PUSH AX
-t
```

```
AX=D5D6 BX=D3D4 CX=D1D2 DX=0000 SP=FFEA BP=0000 SI=0000 DI=0000
DS=2974 ES=2974 SS=2974 CS=2974 IP=010E NV UP EI PL NZ NA PO NC
2974:010E 53 PUSH BX
-t
```

```
AX=D5D6 BX=D3D4 CX=D1D2 DX=0000 SP=FFE8 BP=0000 SI=0000 DI=0000
DS=2974 ES=2974 SS=2974 CS=2974 IP=010F NV UP EI PL NZ NA PO NC
2974:010F 51 PUSH CX
-t
```

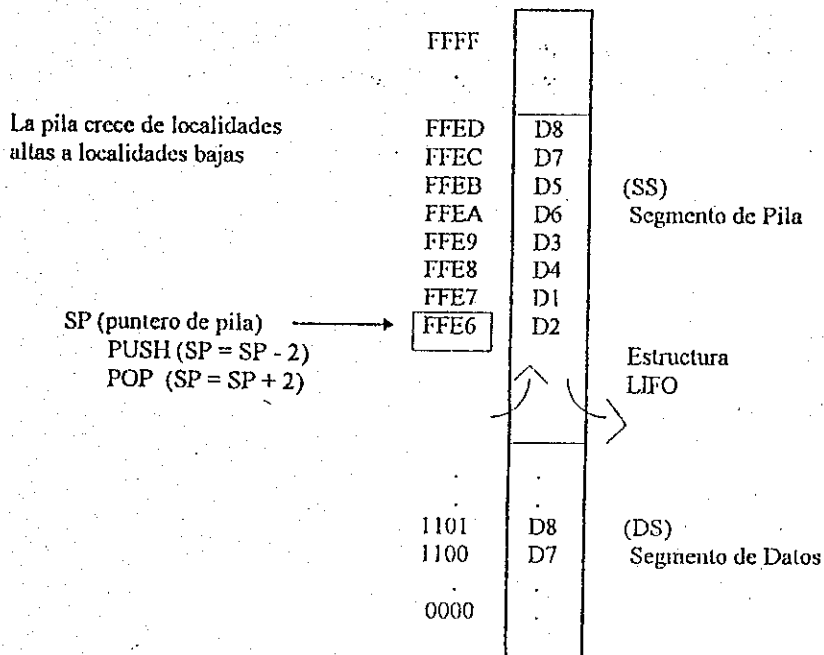
```
AX=D5D6 BX=D3D4 CX=D1D2 DX=0000 SP=FFE6 BP=0000 SI=0000 DI=0000
DS=2974 ES=2974 SS=2974 CS=2974 IP=0110 NV UP EI PL NZ NA PO NC
2974:0110 58 POP AX
```

En este momento, ya se han ejecutado todas las instrucciones para el llenado de la pila, por consiguiente, se puede observar el contenido de la pila de memoria con el comando D (Display). Para lograrlo utilicelo seguido del indicador de segmento de pila (SS), dos puntos (:) y el desplazamiento correspondiente, que en este caso es el valor del puntero de pila (SP).

```
-dss:ffe6
2974:FFE0          D2 D1 D4 D3 D6 D5 D7 D8 00 00 .....
2974:FFF0  F8 F8 F8 F8 F8 F8 F8 F8 F8 F8 F8 F8 F8 F8 F8 .....
           ←----- posiciones -----→
           E0 E1 E2 E3 E4 E5 E6 E7 E8 E9 EA EB EC DE EE EF
```

La línea anterior apunta a que se puedan reconocer las 16 posiciones que se muestran en la línea segmento:desplazamiento: 2974:FFE0.

Este es el momento de hechar una mirada al mapa parcial de memoria para comprender, si todavía no lo hizo, la disposición de los datos que se ingresaron a la memoria, teniendo en cuenta que si bien los registros de UCP son de 16 bits, las locaciones de memoria son sólo de 8 bits.



MAPA PARCIAL DE MEMORIA

Tenga en cuenta que el registro SP se va decrementando antes que se ingresan los datos a la pila (en total 8 bytes) y luego de extraerlos se incrementa hasta llegar, en este caso, al valor con que inició el procesamiento.

Advierta, también, la particularidad que se presenta en la palabra formada por las locaciones de memoria FFED y FFEC respecto de las subsecuentes. En este caso, el contenido de los bytes que la conforman no se encuentran invertidos. Esto se debe, simplemente, a que estos valores fueron ingresados individualmente (a nivel byte) a ambas posiciones de memoria y no a través de un registro (nivel palabra), que se almacena en memoria en forma invertida. Continuamos con el análisis de las últimas instrucciones:

-t

AX=D1D2 BX=D3D4 CX=D1D2 DX=0000 SP=FFE8 BP=0000 SI=0000 DI=0000
 DS=2974 ES=2974 SS=2974 CS=2974 IP=0111 NV UP EI PL NZ NA PO NC
 2974:0111 5B POP BX

-t

AX=D1D2 BX=D3D4 CX=D1D2 DX=0000 SP=FFEA BP=0000 SI=0000 DI=0000
 DS=2974 ES=2974 SS=2974 CS=2974 IP=0112 NV UP EI PL NZ NA PO NC
 2974:0112 59 POP CX

-t

AX=D1D2 BX=D3D4 CX=D5D6 DX=0000 SP=FFEC BP=0000 SI=0000 DI=0000
 DS=2974 ES=2974 SS=2974 CS=2974 IP=0113 NV UP EI PL NZ NA PO NC
 2974:0113 5A POP DX

-T

AX=D1D2 BX=D3D4 CX=D5D6 DX=D8D7 SP=FFEE BP=0000 SI=0000 DI=0000
 DS=2974 ES=2974 SS=2974 CS=2974 IP=0114 NV UP EI PL NZ NA PO NC
 2974:0114 CD20 INT 20

-q

Si observa con detenimiento el manejo de la pila podrá comprender por qué estos registros han intercambiado su valor y, por último, por qué el contenido de la palabra [1100] se halla en el registro DX.

Como conclusión y debido al criterio de estructuración que utiliza la pila (LIFO), los valores que se ingresan a la pila, se extraen en orden inverso respecto del que tenían cuando se ingresaron.

Interrupciones y excepciones

Si durante el procesamiento de un programa, que se encuentra en estado de ejecución se produce algún acontecimiento, interno o externo, que provoque que la ejecución del programa se detenga para atenderlo, ese acontecimiento se denomina *interrupción*. Dicho de otra manera, una interrupción provoca la transferencia del control de un programa en estado de ejecución, a una rutina de servicio, que a su término, devolverá el control al programa que fue suspendido para que prosiga con su ejecución.

La actividad de la UCP debe poder ser controlada de alguna manera y consecuentemente, la UCP debe estar 'atenta' para poder dar *servicio a una interrupción*, es por eso que realiza consultas, en forma permanente, para detectar una solicitud de interrupción.

Cuando el programa en estado de ejecución se suspende por algún acontecimiento erróneo, de carácter interno al mismo, esta suspensión se denomina *excepción*.

La cantidad de interrupciones y excepciones diferentes que se pueden producir, en general, son 256, razón por la cual, se mantiene en memoria una 'Tabla de vectores de interrupciones' de 256 entradas, una para cada tipo de interrupción, que se encuentra alojada a partir de la posición 0 de memoria baja y ocupa los primeros 1024 bytes (4 bytes para cada entrada, con el formato desplazamiento : segmento - 2 bytes para cada uno -). Estos vectores señalan el inicio de otra zona de memoria (RAM o ROM) donde se encuentra la rutina que se encargará de atender o dar servicio a la causa que provocó la llamada de atención a la UCP. Estas rutinas reciben el nombre de *Manejadores de Interrupciones (Interrupt Drivers)*.

El S.O., antes de proceder a la atención, tanto de las interrupciones como de la excepciones, en general, guarda en la pila de memoria la dirección de retorno (CS:IP) y, eventualmente, el registro de banderas; posibilitando así, restaurar los valores resguardados en la pila y retornar el control al programa que la solicitó, luego de haber sido cumplimentada la misma. La dirección de retorno en el caso de las interrupciones es la instrucción siguiente a la que produjo la interrupción, mientras que en el caso de las excepciones, es la dirección de la instrucción que produjo la excepción.

Podemos realizar una clasificación de las interrupciones y de las excepciones.

Interrupciones:

Externas o Hardware: Son convocadas asincrónicamente, en cualquier momento. No se encuentran bajo el control del programa.	No enmascarables (NMI). La UCP es avisada por una señal llamada NMI.	Siempre es atendida. Se consideran de máxima importancia. Ej.: errores en la memoria o falta de energía.
	Enmascarables (INTR). La UCP es avisada por una señal llamada INTR.	Cuando se produce según una condición. Por ejemplo, si la bandera de interrupción del registro de estado está activada (EI), la interrupción es atendida por la UCP. De ser así, un circuito externo especial, llamado Controlador Programable de Interrupciones, decide la prioridad que tiene sobre las demás peticiones de interrupción posibles y le da curso a la consulta del vector de interrupciones correspondiente. Ej.: atención de un periférico de entrada, recepción de un byte en un puerto, etc. Si, en el caso anterior, la bandera de interrupción está desactivada (DI), la UCP no la tiene en cuenta y no se ejecuta.

Internas o Software:

Se convocan en forma sincrónica en el programa.

Se pueden producir por la ejecución de una instrucción específica dentro de un programa, para solicitar una interrupción que cumpla con alguna función determinada.

La forma de convocarla es INT n, donde n corresponde al número de entrada de la Tabla de interrupciones.

Ej.: INT 20 actúa como fin de programa, cediendo el control al programa que lo convocó.

INT 21 ejecuta una función del DOS.

INT 25 lee sectores de un disco.

INT 26 graba sectores de un disco.

Excepciones:

Faltas o errores

Son las que se pueden detectar y corregir antes que se produzca la ejecución de una instrucción determinada.

Ej.: se produce un fallo al invocar una página de memoria, que no está en Memoria Principal. La atención de esta excepción provoca que el S.O. localice la página faltante en la Memoria Virtual y la cargue en la Principal.

Otra falta común puede ser el código de operación no válido.

Son las que se detentan una vez ejecutada la instrucción que las provoca, por ejemplo, las que de alguna manera, incorpora el usuario en el programa.

Ej.: Sobreflujo.

Son las que se detectan sin poder localizar la instrucción que las provoca, abortando el procesamiento del programa.

Ej.: Desborde del segmento originado por el coprocesador.

Doble fallo, que se produce cuando en el intento de dar servicio a una excepción, se produce otra excepción.

Las interrupciones software (INT#), también suelen dividirse en :

- Interrupciones del BIOS (Basic Input/Output System: Sistema básico de entrada/salida): normalmente, se encargan de controlar los periféricos, pero también, controlan algunas funciones del disco.
- Interrupciones del DOS (Disk Operating System, usualmente conocido como Sistema Operativo -S.O.-): se encargan de administrar el disco y la memoria. Una de las más comunes es la INT 21, que permite realizar múltiples funciones (utilizar las funciones del DOS), cargando el número que identifica a la misma en el registro AH.

Para poder encontrar las descripciones de las interrupciones que controlan, tanto la BIOS como el DOS, se debe recurrir al Manual de Referencia Técnicas de la BIOS y al Manual Técnico de DOS, pertenecientes a los componentes de su instalación.

Ejemplo 11 :

Búsqueda de la Tabla de Vectores de Interrupción que se encuentra ubicada en las posiciones más bajas de la memoria (primeras 1024 posiciones)

```
C:\>debug
-d0:0
0000:0000 0A 2B 57 1C 65 04 70 00-16 00 1B 10 65 04 70 00 .+W.e.p....e.p.
0000:0010 65 04 70 00 54 FF 00 F0-4C E1 00 F0 6F EF 00 F0 e.p.T...L...o...
0000:0020 00 00 6F 20 28 00 1B 10-6F EF 00 F0 6F EF 00 F0 ..o(...o...o...
0000:0030 0B 3A E2 17 6F EF 00 F0-9A 00 1B 10 65 04 70 00 ...o.....e.p.
0000:0040 0B 00 F4 20 4D F8 00 F0-41 F8 00 F0 47 25 5D FD ...M...A...G%}.
0000:0050 39 E7 00 F0 5A 05 A5 0A-2D 04 70 00 28 0A 31CC 9...Z...-p.(.1.
0000:0060 D4E3 00 F0 2F 00 DF 10-6E FE 00 F0 04 06 31CC .../...n....l.
0000:0070 1D 00 6F 20 A4 F0 00 F0-22 05 00 00 C2 D7 FF FF ..o....".....
```

Este es un vuelco de memoria (dirección 0000:0000 - 0000:007F) que permite visualizar el contenido de una porción de la Tabla de Vectores de Interrupción. Para obtener el vuelco de toda la tabla (1024 bytes), sería necesario repetir 7 veces más el comando D.

Si toma en cuenta que cada vector usa 4 bytes, para encontrar la dirección de memoria que corresponde al manejador de una interrupción (Interruption Driver) deberá seguir los siguientes pasos:

1. multiplicar el número de interrupción por 4,
2. tomar los 4 bytes que se encuentran en esa localidad, que especifican desplazamiento: segmento,
3. convertirlos a segmento : desplazamiento y, finalmente,
4. invertir los bytes de cada palabra, ya que en memoria el almacenamiento es en orden inverso.

Si, por ejemplo, usted quiere encontrar la dirección de memoria donde se encuentra el manejador de la interrupción #10 (interrupción de la BIOS para el manejo del video), debe multiplicar 10h por 4 (que es la cantidad de bytes por vector). Obtendrá como resultado 40h. Tome los dos bytes que se encuentran a partir de esa posición 0B00 y

F420 y conviértalos a segmento:desplazamiento, esto es, F420:0B00. Por último, invierta los bytes de cada palabra. No olvide que los datos en memoria, siempre se encuentran en orden inverso: 20F4:000B. Pida un vuelco de esa dirección.

```
-d20F4:000B
20F4:0000          FB 84 E4 74 14 .G[...]c.....t
20F4:0010 3D 02 4F 74 0A 80 FC 4F-74 0A EA 00 00 74 20 53 =.Ol...Ol...t S
20F4:0020 50 8B C3 3D 50 50 25 7F-7F 83 F8 03 76 0B 83 F8 P..=PP%.....v...
20F4:0030 07 74 06 58 E8 D0 FF EB-E1 58 9C 0E E8 DB FF E8 .tX.....X.....
20F4:0040 C5 FF CA 02 00 FF 1E 52-04 5A 5B C3 66 50 67 89 .....R.Z[.IPg.
20F4:0050 0C 33 67 89 44 33 02 66-C1 E8 10 67 88 44 33 04 .3g.D3.f...g.D3.
20F4:0060 67 88 64 33 07 0A F6 74-05 67 88 54 33 07 66 58 g.d3...t.g.T3.fX
20F4:0070 C3 53 66 56 57 B4 08 FF-1E 52 04 0B C0 0F 84 B3 .SfVW....R.....
-q
```

Si bien, a simple vista, no podría entender el contenido de esta porción de memoria (para hacerlo necesitaría descifrar el código de máquina que encierra), se puede asegurar que ésta, por lo menos, es una parte del programa que maneja o se encarga de ejecutar la interrupción #10.

Ahora, observe los siguientes ejemplos que utilizan interrupciones del BIOS y del DOS.

Ejemplo 11.1 :

Vizualización por pantalla de las teclas que se digitan en el teclado, mediante interrupciones del DOS

Tenga en cuenta que cuando usted oprime una tecla en su teclado, se emite una señal que le avisa al sistema operativo que usted quiere "hacer algo". El sistema operativo chequea continuamente la vía de comunicación con el usuario y cuando encuentra esa señal ejecuta una interrupción que acepta el valor de la tecla digitada. Este aviso al sistema operativo puede invocar la realización de diversas funciones, tales como, reconocer y ejecutar comandos del DOS, permitir que se ingresen datos a un programa, etc.

En este ejemplo, nos interesa, no sólo determinar el mecanismo por el cual se ingresa el valor de la tecla que se digita, sino también el que nos permite ver en la pantalla la representación de la tecla pulsada. Estas dos tareas se realizan con la misma interrupción INT 21 (Function Request), que permite acceder a distintas funciones del DOS, en este caso:

08 = Wait for console input without echo (ingresa por teclado sin eco o vizualización) y
02 = Display output (muestra la salida).

C:\>debug

-a

```
2974:0100 mov ah,8
2974:0102 int 21
2974:0104 mov ah,2
2974:0106 mov dl,al
2974:0108 int 21
2974:010A cmp al,0d
```

```

2974:010C jnz 100
2974:010E int 20
2974:0110
-g
"Este es un ejemplo"

```

El programa ha finalizado con normalidad
-q

La primera llamada a la interrupción #21, utiliza la función 8 que lee el teclado y la segunda utiliza la función 2 que permite visualizar en la pantalla lo que se había leído previamente. En el ejemplo, se ingresaron en forma sucesiva los caracteres "Este es un ejemplo". Si el dato leído es un retorno de carro (CR), provocado al digitar <enter> y representado por los caracteres hexadecimales 0D, finaliza el programa.

Ejemplo 11.2 : Carga del primer sector del disco de booteo o de arranque en la memoria

```

C:\>debug
-a
2974:0100 mov dx,9000
2974:0103 mov es,dx
2974:0105 xor bx,bx
2974:0107 mov ax,0201
2974:010A mov cx,1
2974:010D mov dx,180
2974:0110 int 13
2974:0112 int 20
2974:0114

```

La INT 13 (Fixed Disk BIOS), posibilita, en este caso, la confección de un programa que lea sólo el primer sector del disco de booteo (o de arranque) y lo cargue a partir de la dirección 9000:0000 del segmento extra de memoria, suponiendo que el disco desde donde arranca el computador sea la unidad "C".

Esta interrupción necesita los siguientes parámetros para su funcionamiento:

AH = número de función requerida (en este caso, 02 = Read the desired sectors into memory),

AL = número de sectores,

BX = dirección de un buffer de memoria, necesariamente, del ES = Extra Segment (ES:DX),

CH = número de cilindro,

CL = número de sector de comienzo,

DH = número de cabeza o cara,

DL = número que identifica al driver (80 para C; 81 para D ; 0 para A ; 1 para B).

Las dos primeras instrucciones de este programa, le asignan al ES (Segmento Extra) una porción de memoria que comienza en la dirección 9000. Se utiliza el registro DX como intermediario, ya que no se le puede asignar, en forma directa, un valor al registro ES. La tercera instrucción, pone a 0 el registro BX, cuyo contenido será asignado como

valor de desplazamiento del ES. A partir de la cuarta instrucción, se seleccionan los valores de los parámetros:

AX = 0201, donde 02 corresponde a la lectura de los sectores indicados y su copia en la memoria, y 01 la cantidad de sectores a leer,

CX = 0001, donde 00 es el número de pista o cilindro que se selecciona y 01 el número de sector a partir del cual se hace la selección,

DX = 0180, donde 01 es el número de cabeza o cara seleccionada y 80 es el número identificador del driver ("C").

Las dos últimas instrucciones son INT 13, que llama a la interrupción FIXED DISK BIOS para que sea ejecutada y finalmente, INT 20 que devuelve el control al DEBUG.

La carga en memoria del sector que se ubica en la posición 2000:0000 (sector 0) del disco rígido y su posterior vuelco, le permite acceder al contenido del Area de Sistema del mismo. Si, posteriormente, ejecuta el programa y pide un dump de la posición 9000:0000 podrá verificar que la carga fue correcta. Analice el contenido de ambas áreas.

```
-L 2000:0000 2 0 1
-d 2000:0000
2000:0000 EB 3E 90 4D 53 57 49 4E-34 2E 30 00 0210 01 00 >.MSWIN4.0.....
2000:0010 02 00 02 00 00 F8 C9 00-33 00 10 00 33 00 00 00 .....3...3...
2000:0020 FD 8F 0C 00 80 00 29 77-79 45 1D 20 20 20 20 20 .....)wyE.
2000:0030 20 20 20 20 20 20 46 41-54 31 36 20 20 20 F1 7D ... FAT16 .}
2000:0040 FA 33 C9 8E D1BC FC 7B-16 07 BD 78 00 C5 76 00 ...{...x.v.
2000:0050 1E 56 16 55 BF 22 05 89-7E 00 89 4E 02 B1 0B FC .V.U."~N....
2000:0060 F3 A4 06 1F BD 00 7C C6-45 FE 0F 8B 46 18 88 45 .....|E...F..E
2000:0070 F9 FB 38 66 24 7C 04 CD-13 72 3C 8A 46 10 98 F7 ..8f$|...r<F...
-d
2000:0080 66 16 03 46 1C 13 56 1E-03 46 0E 13 D1 50 52 89 f.F..V..F...PR.
2000:0090 46 FC 89 56 FE B8 20 00-8B 76 11 F7 E6 8B 5E 0B F..V...v....^
2000:00A0 03 C3 48 F7 F3 01 46 FC-11 4E FE 5A 58 BB 00 07 ..H...F..N.ZX...
2000:00B0 8B FB B1 01 E8 94 00 72-47 38 2D 74 19 B1 0B 56 .....rG8-t...V
2000:00C0 8B 76 3E F3 A6 5E 74 4A-4E 74 0B 03 F9 83 C7 15 .v>...^UNl.....
2000:00D0 3B FB 72 E5 EB D7 2B C9-B8 D8 7D 87 46 3E 3C D8 ;r...+...|F><
2000:00E0 75 99 BE 80 7D AC 98 03-F0 AC 84 C0 74 17 3C FF u...}.....t.<
2000:00F0 74 09 B4 0E BB 07 00 CD-10 EB EE BE 83 7D EB E5 t.....}-
-d
2000:0100 BE 81 7D EB E0 33 C0 CD-16 5E 1F 8F 04 8F 44 02 ..).3..^...D.
2000:0110 CD 19 BE 82 7D 8B 7D 0F-83 FF 02 72 C8 8B C7 48 .....}.)....r...H
2000:0120 48 8A 4E 0D F7 E1 03 46-FC 13 56 FE BB 00 07 53 H.N....F..V....S
2000:0130 B1 04 E8 16 00 5B 72 C8-81 3F 4D 5A 75 A7 81 BF .....[r..?MZu...
2000:0140 00 02 42 4A 75 9F EA 00-02 70 00 50 52 51 91 92 ..BJu....p.PRQ..
2000:0150 33 D2 F7 76 18 91 F7 76-18 42 87 CA F7 76 1A 8A 3..v...v.B...v..
2000:0160 F2 8A 56 24 8A E8 D0 CC-D0 CC 0A CC B8 01 02 CD ..V$.....
2000:0170 13 59 5A 58 72 09 40 75-01 42 03 5E 0B E2 CC C3 .YZXr.@u.B.^....
-d
2000:0180 03 18 01 27 0D 0A 44 69-73 63 6F 20 64 65 20 73 ...!...Disco de s
2000:0190 69 73 74 65 6D 61 20 69-6E 63 6F 72 72 65 63 74 istema incorrect
2000:01A0 6F FF 0D 0A 45 72 72 6F-72 20 45 2F 53 FF 0D 0A o...Error E/S...
2000:01B0 43 61 6D 62 69 65 20 65-6C 20 20 64 69 73 63 6F Cambie el disco
2000:01C0 20 79 20 70 72 65 73 69-6F 6E 65 20 75 6E 61 20 y presione una
2000:01D0 74 65 63 6C 61 0D 0A 00-49 4F 20 20 20 20 20 20 tecla...IO
2000:01E0 53 59 53 4D 53 44 4F 53-20 20 20 53 59 53 80 01 SYSMSDOS SYS..
```

2000:01F0 00 57 49 4E 42 4F 4F 54-20 53 59 53 00 00 55 AA .WINBOOT SYS..U.

-8

El programa ha finalizado con normalidad

-d9000:0

```
9000:0000 EB 3E 90 4D 53 57 49 4E-34 2E 30 00 0210 01 00 >.MSWIN4.0.....
9000:0010 02 00 02 00 00 F8 C9 00-33 00 10 00 33 00 00 00 .....3...3...
9000:0020 FD 8F 0C 00 80 00 29 77-79 45 1D 20 20 20 20 20 .....)wyE.
9000:0030 20 20 20 20 20 20 46 41-54 31 36 20 20 20 F1 7D .FAT16 .)
9000:0040 FA 33 C9 8E D1BC FC 7B-16 07 BD 78 00 C5 76 00 .3.....{...x..v.
9000:0050 1E 56 16 55 BF 22 05 89-7E 00 89 4E 02 B1 0B FC .V.U."...N....
9000:0060 F3 A4 06 1F BD 00 7C C6-45 FE 0F 8B 46 18 88 45 .....|E...F..E
9000:0070 F9 FB 38 66 24 7C 04 CD-13 72 3C 8A 46 10 98 F7 ..8f$|...r<F...
-d
9000:0080 66 16 03 46 1C 13 56 1E-03 46 0E 13 D1 50 52 89 f..F..V..F...PR.
9000:0090 46 FC 89 56 FE B8 20 00-8B 76 11 F7 E6 8B 5E 0B F..V...v....^
9000:00A0 03 C3 48 F7 F3 01 46 FC-11 4E FE 5A 58 BB 00 07 ..H...F..N.ZX...
9000:00B0 8B FB B1 01 E8 94 00 72-47 38 2D 74 19 B1 0B 56 .....rG8-t...V
9000:00C0 8B 76 3E F3 A6 5E 74 4A-4E 74 0B 03 F9 83 C7 15 .v>..^JNt.....
9000:00D0 3B FB 72 E5 EB D7 2B C9-B8 D8 7D 87 46 3E 3C D8 ;r...+...}F><.
9000:00E0 75 99 BE 80 7D AC 98 03-F0 AC 84 C0 74 17 3C FF u...}.....t<.
9000:00F0 74 09 B4 0E BB 07 00 CD-10 EB EE BE 83 7D EB E5 t.....}..
-d
9000:0100 BE 81 7D EB E0 33 C0 CD-16 5E 1F 8F 04 8F 44 02 ..j..3...^...D.
9000:0110 CD 19 BE 82 7D 8B 7D 0F-83 FF 02 72 C8 8B C7 48 .....).}...r...H
9000:0120 48 8A 4E 0D F7 E1 03 46-FC 13 56 FE BB 00 07 53 H.N....F..V....S
9000:0130 B1 04 E8 16 00 5B 72 C8-81 3F 4D 5A 75 A7 81 BF .....{r..?MZu...
9000:0140 00 02 42 4A 75 9F EA 00-02 70 00 50 52 51 91 92 ..BJu....p.PRQ..
9000:0150 33 D2 F7 76 18 91 F7 76-18 42 87 CA F7 76 1A 8A 3..v...v.B...v..
9000:0160 F2 8A 56 24 8A E8 D0 CC-D0 CC 0A CC B8 01 02 CD ..V$.....
9000:0170 13 59 5A 58 72 09 40 75-01 42 03 5E 0B E2 CC C3 .YZXr.@u.B.^...
-d
9000:0180 03 18 01 27 0D 0A 44 69-73 63 6F 20 64 65 20 73 ....Disco de s
9000:0190 69 73 74 65 6D 61 20 69-6E 63 6F 72 72 65 63 74 istema incorrect
9000:01A0 6F FF 0D 0A 45 72 72 6F-72 20 45 2F 53 FF 0D 0A o...Error E/S...
9000:01B0 43 61 6D 62 69 65 20 65-6C 20 20 64 69 73 63 6F Cambie el disco
9000:01C0 20 79 20 70 72 65 73 69-6F 6E 65 20 75 6E 61 20 y presione una
9000:01D0 74 65 63 6C 61 0D 0A 00-49 4F 20 20 20 20 20 20 tecla...IO
9000:01E0 53 59 53 4D 53 44 4F 53-20 20 20 53 59 53 80 01 SYSMSDOS SYS..
9000:01F0 00 57 49 4E 42 4F 4F 54-20 53 59 53 00 00 55 AA .WINBOOT SYS..U.
```

El DOS reserva en todo disco formateado, la pista exterior o cilindro (0), de la primera cara o cabeza, para el *Area de Sistemas*. En esta pista se encuentran:

el *Registro de Booteo o de Arranque*, que forma parte de un área mayor, denominada *Tabla de particiones*. La *Tabla de Particiones*, de un disco rígido (los flexibles no pueden particionarse), permite dividirlo en zonas o "particiones" y hace que el DOS asuma cada una de ellas como discos diferentes,

la *Tabla de Asignación de Archivos* (FAT - File Allocation Table) y

el *Directorio Raíz*.

El Registro de Booteo contiene el *Bloque de Parámetros de la BIOS (BPE)*, que da información sobre las características físicas del disco, para ser usadas por los controladores de los mismos:

- instrucción de salto a una dirección de memoria donde comienza el programa de arranque (el primer byte es el código de salto, el segundo es el desplazamiento de la dirección y el tercero es el código de no operación -NOP-) 3 bytes
- versión del DOS usada para formatear el disco 8 bytes
- número de bytes por sector 2 bytes
- nro. de bytes por cluster, por pista y por disco o partición del disco duro 1 byte
- número de sectores reservados para el Area de Sistemas 2 bytes
- número de copias de la FAT 1 byte
- número de entradas al directorio raíz 2 bytes
- número de sectores de disco 2 bytes
- indicador de tipo de soporte o disco 1 byte
- número de sectores por FAT (varian con la capacidad del disco) 2 bytes
- número de sectores por pista (varian en función del tamaño del disco) 2 bytes
- número de cabezas o caras 2 bytes
- número de sectores ocultos (son del área del sistema) 2 bytes
- los 2 últimos octetos no tienen información 2 bytes.

Cada vez que usted arranca su PC, la pone en ON (léase la enciende), la resetea o digita Ctrl+ALT+Del un programa de inicialización se pone en marcha. Este programa se encuentra en memoria ROM y testea el hardware del ordenador, para asegurarse que todos sus componentes están en óptimas condiciones. Si esto es así, carga en memoria el Registro de Booteo, que se encuentra alojado en un disco (ya sea el que está ubicado en la unidad "A" o en el rígido "C") y que contiene el BPB. Si la PC no cuenta con unidades de disco, normalmente, se activa el intérprete de BASIC. A continuación, se copian del disco a memoria, los archivos del sistema DOS (IO.SYS o IBMBIO.COM y MSDOS.SYS o IBMDOS.COM) y el COMMAND.COM. Por último, toma el control el COMMAND.COM que carga los archivos CONFIG.SYS, el cual declara todos los controladores a ser utilizados, y el AUTOEXEC.BAT, que es un *archivo de procesamiento por lotes* que habilita, entre otras cosas, la ejecución de aquellos programas que tienen que permanecer residentes en memoria, como por ejemplo, Windows. Cuando la carga llega a su fin, se visualiza el prompt del DOS, que indica que su PC se encuentra disponible para ser utilizada. Si por alguna circunstancia, accidental, los archivos mencionados del DOS no pueden ser localizados, el programa de inicialización, se lo hará saber con un mensaje de error y permanecerá expectante hasta que usted coloque un Disco de Sistema (disco de booteo) en la disquetera "A".

Si desensambla, a partir de cualquiera de las direcciones de segmentos utilizados, podrá ver la dirección a la cual debe saltar para encontrar el programa que se encarga de "arrancar" su PC.

```

-u 2000:0000 L 3
2000:0000 EB3E    JMP    0040
2000:0002 90      NOP

```

Desensamble a partir de la dirección que se indica en su vuelco. En este ejemplo la dirección es 0040.

```

-u2000:0040
2000:0040 FA      CLI
2000:0041 33C9    XOR     CX,CX
2000:0043 8ED1    MOV     SS,CX
2000:0045 BCFC7B  MOV     SP,7BFC
2000:0048 16      PUSH    SS
2000:0049 07      POP     ES
2000:004A BD7800  MOV     BP,0078
2000:004D C57600  LDS     SI,[BP+00]
2000:0050 1E      PUSH    DS
2000:0051 56      PUSH    SI
2000:0052 16      PUSH    SS
2000:0053 55      PUSH    BP
2000:0054 BF2205  MOV     DI,0522
2000:0057 897E00  MOV     [BP+00],DI
2000:005A 894E02  MOV     [BP+02],CX
2000:005D B10B    MOV     CL,0B
2000:005F FC      CLD

```

Esta, es sólo una parte del programa de arranque. Si desea seguir investigando, continúe hasta encontrar el fin del programa.

Práctico 11 : Interrupciones del DOS y de la BIOS

1. Utilice la INT 25 (Absolute Disk Read) para realizar un programa que emule la instrucción Load (L) del Debugger y cargue, a partir de la dirección 200, del segmento de memoria con el que está trabajando, sólo el contenido del sector número 0 de la unidad de diskettes "A". Ejecute el programa, haga un vuelco de la dirección 200 antes y después de la ejecución del mismo y verifique, además, su funcionamiento comparándolo con el vuelco que produce la carga del mismo sector a partir de la dirección 0400 con la instrucción Load. Para más referencias sobre el Load, consulte el Manual del usuario y referencias del MS-DOS.

Esta interrupción necesita los siguientes datos para su funcionamiento:

- AL = número de drive (0 para A),
 - CX = número de sectores a leer,
 - DX = número de sector lógico de comienzo,
 - DS:BX = dirección de transferencia.
2. Modifique el color de fondo de la pantalla (background), a partir de la interrupción del BIOS #10 (Video BIOS). Para lograrlo, el registro AH debe contener el número de función a utilizar, siendo 0B el correspondiente a SET COLOR PALETTE (ajustar el color) y el registro BL indica el número correspondiente al

color que se desea ingresar. En el registro BH se indica con 0 o con 1 si se modifica el color del background (fondo) o del foreground (primer plano) de la pantalla, respectivamente.

Disco Magnético. El Soporte Magnético De Mayor Uso.

Los discos magnéticos pueden ser rígidos o flexibles, dispuestos en unidades de cabezas fijas o móviles, en grupos de uno o más platos. Más allá de todos estos detalles debemos considerar que un disco está constituido por una base, recubierta de material magnetizable con forma de circunferencia que gira alrededor de un eje dispuesto en la unidad y que es accedido por una o más cabezas lecto/grabadoras.

Una cabeza registra los bits en círculos concéntricos denominados pistas (tracks). Una pista es una división lógica y no física de la superficie producto de la acción de rotación del soporte y de la posición fija de la cabeza al momento de grabación o lectura. La cantidad de pistas en la superficie depende del tipo de unidad a la que está asociado el soporte, y del sistema operativo que lo gestione. Si la unidad graba las dos caras de un plato y/o cuenta con más de uno habrá muchas pistas de igual número (e igual distancia respecto del eje) en las distintas caras. Podemos imaginarnos que por efecto de rotación se describen tantos cilindros concéntricos como pistas halla. Por cada cara grabable habrá una cabeza lecto-grabadora. Todas ellas acceden simultáneamente como si fueran los dientes de un peine a un cilindro (cuya dirección está dada por un número relativo a la distancia de radio entre las cabezas y el eje). El concepto es importante dado que la información se graba por cilindro, es decir, verticalmente. Considere que cada movimiento de cabeza es mecánico y, por lo tanto, resulta entonces óptimo grabar todas las pistas de igual número sin modificar su posición. Cuando se completa un cilindro se pasa al siguiente.

Si bien la unidad de referencia que hace el proceso a un archivo es, en la mayoría de los casos, el registro lógico, la unidad de acceso al soporte es una agrupación de ellos, denominada registro físico o bloque.

En un disco, cada registro físico se graba en un sector de pista, siendo ésta la unidad de acceso al soporte. Todos los sectores de una pista almacenan la misma cantidad de bits, y a su vez los sectores de pistas distintas conservan el mismo criterio, por lo tanto, se deduce que la densidad de grabación aumenta sucesivamente respecto de la distancia del eje central (cuanto más cerca del eje, mayor densidad de grabación).

Si decimos que la cantidad de bits por sector se mantiene constante resulta, entonces, que la cantidad de registros lógicos por sector también lo es. La determinación del volumen de información por sector es responsabilidad del sistema operativo. Ciertos programas del sistema operativo tienen la responsabilidad de dar una estructura lógica que permita un ágil reconocimiento de la ubicación de cada información solicitada. Estos programas se dedican a organizar y administrar la gestión de almacenamiento y captación de bits en el disco de la manera más eficiente. Así, cada sistema operativo reconoce el territorio sobre el soporte a su manera (no tiene sentido profundizar todas ellas).

Originalmente, un disco es un territorio sin límites y se denomina virgen. Se dice que un programa que da una estructura productiva es un formateador o inicializador del soporte.

Los parámetros que determinan la ubicación de una entidad para su acceso son el número de pista o cilindro, el número de cara y el número de sector. El "formateo" (del inglés "format" nombre que se denomina al programa de inicialización) de un disco implica la grabación de labels (o marcas) que identifican estos parámetros, de este modo se establece donde se grabarán los archivos. El proceso de "formateo" es, entonces, imprescindible para poder trabajar con el soporte.

A modo de ejemplo, considere una estructura de disco en PC bajo DOS. Como los PC pueden administrarse bajo distintos sistemas operativos (CP/M-86, UCS DP-System, XENIX, PC-LX) se han mantenido algunas formas estandares comunes a todos ellos y algunas formas variables convenientes a cada sistema operativo en particular.

El programa FORMAT del DOS genera en el disco un área para la gestión del soporte y un área de datos para los archivos del usuario. La primera se conoce como área de sistema y contiene un sector de arranque (BOOT), una tabla índice que indica el "domicilio" de cada archivo grabado (FAT) y, - una tabla de atributos pertenecientes a cada archivo (DIRECTORIO).

Consideremos como ejemplo un soporte de 5¼ pulgadas con 1.2 Mbytes de capacidad, estos discos son divididos lógicamente en 80 pistas numeradas de 0 a 79 cada pista se divide a su vez en 15 sectores. Cada sector aloja 512 octetos.

Cada sector se referencia internamente con la dirección formada por su referencia física (número de cilindro, cara, sector) pero el DOS los numera en forma correlativa y los denomina clusters.

Como ya mencionamos, el primer sector de la pista 0 se denomina sector de arranque (boot sector), allí se almacena entre otras cosas una rutina que carga los distintos programas del sistema operativo. La rutina de arranque es un software autosuficiente que contiene las funciones básicas del DOS y se utiliza además para cargar otros programas del sistema operativo de uso frecuente que quedarán residentes en memoria principal. En el momento de "formateo" se indica si este disco podrá "bootear" el sistema operativo. Si bien el sector de arranque sólo tiene sentido en un solo disco, el resto mantendrá este espacio asignado, aún cuando carezca de una copia de los programas de "booteo", si se pretende arrancar el sistema de un disco "no bootable" el sistema enviará un mensaje (NON SYSTEM- DISK) indicando que no se pueden "cargar" los residentes en memoria.

Se puede también analizar la información almacenada en el sector de con comandos de debug:

Práctico 12 : Inicialización de un disco bajo DOS

1. Ingrese un diskette virgen en el drive A y tipee el siguiente comando
FORMAT A: /S <ENTER>
2. Cuando le pida Nro. de Label tipee DISCO2 <ENTER>

Microsoft(R) Windows 95
(C)Copyright Microsoft Corp 1981-1995.

C:\WINDOWS>format a: /s
Inserte un nuevo disco en la unidad A:
y presione ENTRAR cuando esté listo...

Verificando el formato del disco.
Verificando 1.2M
Formateo completado.
Sistema transferido

¿Nombre del volumen? (11 caracteres, ENTRAR para ninguno)?DISCO2

1.213.952 bytes de espacio total en disco
391.168 bytes utilizados por el sistema
822.784 bytes disponibles en disco

512 bytes en cada unidad de asignación.
1.607 unidades de asignación disponibles en disco.

El número de serie del volumen es 1E23-15CE

¿Desea formatear otro disco (S/N)?

- 3 El objetivo a alcanzar ahora es rastrear la información grabada en el disco por el programa FORMAT. A continuación bajo debug ingrese:

C:\DEBUG <ENTER>
-L 1000:0 0 0 1
-D 1000:0 <ENTER>
-D <ENTER>
-D <ENTER>
-D <ENTER>

Del mismo modo que en el ejemplo 11.2 se hace posible la carga del primer sector con un programa ensamblar, el comando L (load) permite cargar en la dirección de memoria 1000:0000 (que es una dirección tomada al azar) de la unidad 0 (drive a) a partir del sector número 0 un solo sector

-L 1000:0 0 0 1
- cantidad de sectores
---- número de sector inicial
----- número de unidad
----- dirección de memoria
----- L(oad)

El comando D (dump) permite volcar en su pantalla el contenido de 128 octetos a partir de la posición de memoria 1000:0

D 1000:0

----- dirección de memoria
----- D(ump)

Como el sector completo tiene 512 octetos repetimos el comando tres veces más. (si no se especifica la posición de memoria asume la siguiente al último dump). Para más información acerca de estos comandos consulte el manual de DOS de la versión que tiene en su computador en la sección de DEBUG. De este modo se logra el vuelco de los 512 octetos en su pantalla para poder visualizarlo en forma completa presione la tecla <Print Screen> cada vez que se le llene la pantalla. Aparecerá un listado similar al que sigue :

```
C:\WINDOWS>DEBUG
-L 1000:0000
-D 1000:0
1000:0000 EB 3E 90 28 37 66 7C 65-49 48 43 00 02 01 01 00 >.(7f)HC.....
1000:0010 02 E0 00 60 09 F9 07 00-0F 00 02 00 00 00 00 00 .....
1000:0020 00 00 00 00 00 00 29 CE-15 23 1E 44 49 53 43 4F .....).#.DISCO
1000:0030 32 20 20 20 20 20 46 41-54 31 32 20 20 20 F1 7D 2 FAT12 .}
1000:0040 FA 33 C9 8E D1 BC FC 7B-16 07 BD 78 00 C5 76 00 .3.....{...x..v.
1000:0050 1E 56 16 55 BF 22 05 89-7E 00 89 4E 02 B1 0B FC .V.U."..~.N....
1000:0060 F3 A4 06 1F BD 00 7C C6-45 FE 0F 8B 46 18 88 45 .....|.E...F..E
1000:0070 F9 FB 38 66 24 7C 04 CD-13 72 3C 8A 46 10 98 F7 ..8f3|...r<F...
-D
1000:0080 66 16 03 46 1C 13 56 1E-03 46 0E 13 D1 50 52 89 f..F..V..F...PR.
1000:0090 46 FC 89 56 FE B8 20 00-8B 76 11 F7 E6 8B 5E 0B F..V..v....^
1000:00A0 03 C3 48 F7 F3 01 46 FC-11 4E FE 5A 58 BB 00 07 ..H...F..N..ZX...
1000:00B0 8B FB B1 01 E8 94 00 72-47 38 2D 74 19 B1 0B 56 .....rG8-t...V
1000:00C0 8B 76 3E F3 A6 5E 74 4A-4E 74 0B 03 F9 83 C7 15 .v>..^JNl.....
1000:00D0 3B FB 72 E5 EB D7 2B C9-B8 D8 7D 87 46 3E 3C D8 ;r...+...}.F><
1000:00E0 75 99 BE 80 7D AC 98 03-F0 AC 84 C0 74 17 3C FF u...}.....t.<
1000:00F0 74 09 B4 0E BB 07 00 CD-10 EB EE BE 83 7D EB E5 t.....}..
-D
1000:0100 BE 81 7D EB E0 33 C0 CD-16 5E 1F 8F 04 8F 44 02 ..}.3...^....D.
1000:0110 CD 19 BE 82 7D 8B 7D 0F-83 FF 02 72 C8 8B C7 48 ....}.}....r...H
1000:0120 48 8A 4E 0D F7 E1 03 46-FC 13 56 FE BB 00 07 53 H.N....F..V....S
1000:0130 B1 04 E8 16 00 5B 72 C8-81 3F 4D 5A 75 A7 81 BF .....[r..?MZu...
1000:0140 00 02 42 4A 75 9F EA 00-02 70 00 50 52 51 91 92 ..BJu....p.PRQ..
1000:0150 33 D2 F7 76 18 91 F7 76-18 42 87 CA F7 76 1A 8A 3..v...v..B...v..
1000:0160 F2 8A 56 24 8A E8 D0 CC-D0 CC 0A CC B8 01 02 CD ..VS.....
1000:0170 13 59 5A 58 72 09 40 75-01 42 03 5E 0B E2 CC C3 .YZXr.@u.B.^....
-D
1000:0180 03 18 01 27 0D 0A 44 69-73 63 6F 20 69 6E 63 6F ...!..Disco inco
1000:0190 72 72 65 63 74 6F 20 20-20 FF 0D 0A 45 72 72 6F rrecto ...Erro
1000:01A0 72 20 45 2F 53 20 20 20-20 20 FF 0D 0A 43 61 6D r E/S ...Cam
1000:01B0 62 69 65 20 65 6C 20 64-69 73 63 6F 20 79 20 70 bie el disco y p
1000:01C0 72 65 73 69 6F 6E 65 20-75 6E 61 20 74 65 63 6C resione una tecl
1000:01D0 61 20 20 20 20 0D 0A 00-49 4F 20 20 20 20 20 20 a ...IO
1000:01E0 53 59 53 4D 53 44 4F 53-20 20 20 53 59 53 80 01 SYMSDOS SYS..
1000:01F0 00 57 49 4E 42 4F 4F 54-20 53 59 53 00 00 55 AA .WINBOOT SYS..U.
```

Esta es el Area de boot.

- 4 El ejercicio consiste en analizar la información encontrada en su propia área de boot, según la información de la página 42.

Una particularidad del DOS es considerar unidades lógicas llamadas CLUSTERS, que son unidades que agrupan uno o varios sectores y se utilizan como unidad de asignación de espacio a los ficheros. Para el caso particular analizado un cluster es igual a un sector; si bien en otros discos la cantidad de sectores por cluster es mayor, siendo siempre una potencia exacta de 2.

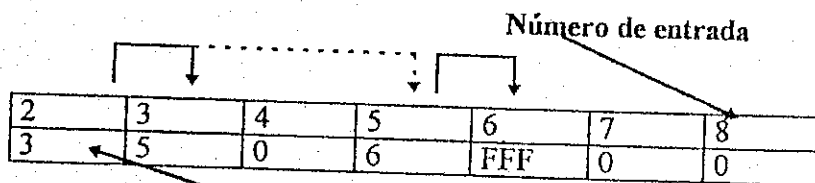
	Capacidad	Cilindros	Sectores x pista	Cabezas	Sectores x Cluster
diskette 5¼ de	360KB	40	9	2	2
	1.2MB	80	15	2	1
3½ de	720KB	80	9	2	2
	1.44MB	80	18	2	1
disco duro de	44 MB	732	17	7	4

Esta Tabla solo considera algunos modelos de soporte de disco usted puede encontrar diferencias según su modelo de disco y las distintas versiones de sistema operativo. Tenga cuidado si su curiosidad lo lleva a analizar el sector de arranque de su disco duro, no modifique nada ya que de este modo podría perder la capacidad de acceder al resto de la información en el soporte.

Práctico 13 : Tabla de Asignación de Archivos bajo DOS (FAT).

La FAT Es una tabla índice (File Allocation Table), en cada entrada se representa un cluster del área de datos (entrada 3 = cluster 3) que a su vez contiene un número que actúa de puntero a otro cluster (entrada 3 contiene un 6). Un archivo está asociado a un cluster de comienzo (definido en el directorio como veremos más adelante), supongamos que en el directorio dice que el archivo comienza en el cluster 2; con ese número se accede a la entrada 2, la entrada señalada contiene el número de cluster donde continúa el archivo, es decir que el archivo continúa en el cluster 3, consultando la entrada 3 se deduce que el archivo continúa en el cluster 5, así las sucesivas entradas se encadenan hasta que el cluster final del archivo se marca con la entrada de la FAT conteniendo todos unos. Si un cluster de datos está vacío su correspondiente entrada en la FAT contiene el número es cero.

En realidad este es un esquema muy simplificado de la organización de la FAT.



Después del sector de arranque se ubican dos copias de la FAT y luego una copia del directorio; siguiendo con el ejemplo anterior se supone que si el sector de boot ocupa el primer sector a partir del sector 1 y hasta el sector 7 (7 sectores) encontraremos la primer copia de la FAT (su tamaño tiene relación con la capacidad del soporte).

Cantidad de Sectores ocupados por las copias de la FAT

	Sector de Boot	Sectores de FAT
diskette 5¼ de 360KB	1	4
1.2MB	1	14
3½ de 720 KB	1	6
1.44MB	1	18

Ahora ingrese el comando DEBUG

-L 1000:0 0 0 1

y obtendrá su propio vuelco de la FAT.

-L 1000:0 0 1 1

-D 1000:0

```

1000:0000 F9 FF FF 03 40 00 05 60-00 07 80 00 09 A0 00 0B .....@..
1000:0010 C0 00 0D E0 00 0F 00 01-11 20 01 13 40 01 15 60 .....@..
1000:0020 01 17 80 01 19 A0 01 1B-C0 01 1D E0 01 1F 00 02 .....
1000:0030 21 20 02 23 40 02 25 60-02 27 80 02 29 A0 02 2B !.#@%'.').+
1000:0040 C0 02 2D E0 02 2F 00 03-31 20 03 33 40 03 35 60 .../.1.3@.5'
1000:0050 03 37 80 03 39 A0 03 3B-C0 03 3D E0 03 3F 00 04 ..7.9...=.?.
1000:0060 41 20 04 43 40 04 45 60-04 47 80 04 49 A0 04 4B A.C@.E'.G.I.K
1000:0070 C0 04 4D E0 04 4F 00 05-51 20 05 53 40 05 55 60 ..M.O.Q.S@.U'

```

-D

```

1000:0080 05 57 80 05 59 A0 05 5B-C0 05 5D E0 05 5F 00 06 ..W.Y.[.]._...
1000:0090 61 20 06 63 40 06 65 60-06 67 80 06 69 A0 06 6B a.c@.e'.g.i.k
1000:00A0 C0 06 6D E0 06 6F 00 07-71 20 07 73 40 07 75 60 ...m.o.q.s@.u'
1000:00B0 07 77 80 07 79 A0 07 7B-C0 07 7D E0 07 7F 00 08 ..w.y..{.}.....
1000:00C0 81 20 08 83 40 08 85 60-08 87 80 08 89 A0 08 8B .....@..
1000:00D0 C0 08 8D E0 08 8F 00 09-91 20 09 93 40 09 95 60 .....@..
1000:00E0 09 97 80 09 99 A0 09 9B-C0 09 9D E0 09 9F 00 0A .....
1000:00F0 A1 20 0A A3 40 0A A5 60-0A A7 80 0A A9 A0 0A AB ...@..

```

--D

```

1000:0100 C0 0A AD E0 0A AF 00 0B-B1 20 0B B3 40 0B B5 60 .....@..
1000:0110 0B B7 80 0B B9 A0 0B BB-C0 0B BD E0 0B BF 00 0C .....
1000:0120 C1 20 0C C3 40 0C C5 60-0C C7 80 0C C9 A0 0C CB ...@..
1000:0130 C0 0C CD E0 0C CF 00 0D-D1 20 0D D3 40 0D D5 60 .....@..
1000:0140 0D D7 80 0D D9 A0 0D DB-C0 0D DD E0 0D DF 00 0E .....
1000:0150 E1 20 0E E3 40 0E E5 60-0E E7 80 0E E9 A0 0E EB ...@..
1000:0160 C0 0E ED E0 0E EF 00 0F-F1 20 0F F3 40 0F F5 60 .....@..
1000:0170 0F F7 80 0F F9 A0 0F FB-C0 0F FD E0 0F FF 00 10 .....

```


-D

```

1000:0180 01 21 10 03 41 10 05 61-10 07 81 10 09 A1 10 0B .I.A.a.....
1000:0190 C1 10 0D E1 10 0F 01 11-11 21 11 13 41 11 15 61 .....I.A.a
1000:01A0 11 17 81 11 19 A1 11 1B-C1 11 1D E1 11 1F 01 12 .....
1000:01B0 21 21 12 23 41 12 25 61-12 27 81 12 29 A1 12 2B II.#A.%a'.).+.
1000:01C0 C1 12 2D E1 12 2F 01 13-31 21 13 33 41 13 35 61 .../..11.3A.5a
1000:01D0 13 37 81 13 39 A1 13 3B-C1 13 3D E1 13 3F 01 14 .7..9...=.7..
1000:01E0 41 21 14 43 41 14 45 61-14 47 81 14 49 A1 14 4B AI.CA.Ea.G.I.K
1000:01F0 C1 14 4D E1 14 4F 01 15-51 21 15 53 41 15 55 61 ..M.O.QI.SA.Ua

```

En el disco duro el tamaño de la FAT depende de cómo esté particionado el disco. Tenga en cuenta el el sistema operativo numera los clusters de dato en forma secuencial observe el vuelco obtenido con el utilitario Norton. Sin embargo los utilitarios de Norton tiende a hacernos la cosas muy simples, y de hecho eso está muy bien !!. Pero piense en que ocurriría si Norton no actualiza sus utilitarios nunca más o si usted se encuentra frente a otros computadores que no los puedan ejecutar. Lo mejor para usted es que tenga su propia experiencia en esta tediosa tarea de encontrar las cosas con la única herramienta del rastreo, no importa que esta vez sea en un PC las tecnologías cambian pero cierta filosofía para implementarlas no cambian nunca. Para conocer la FAT mejor observe el siguiente vuelco :

El primer octeto identifica el tipo de medio o soporte F9 significa doble cara, alta densidad. Esta FAT pertenece a un disco flexible por lo tanto cada entrada está definida por 1½ octeto, es decir que dos entrada ocupan 3 octetos. (En los discos duros cada entrada a la FAT es de 2 octetos).

La tabla se lee así :

Divida el vuelco en grupos de tres octetos y numérelas así 0-1 2-3 4-5 6-7 Cada grupo identifica dos sectores o lo que es lo mismo en un en éste caso dos clusters.

Empiece con el grupo 2-3

- asígnele el primer par de octetos al cluster par (2) 03 40
- dé vuelta la palabra 40 03
- elimine el dígito de orden superior 0 03

003 HEXADECIMAL es 3 en decimal es decir que el archivo continúa en el cluster 3. (si el valor obtenido es 000 se identifica un cluster vacío)

- asígnele el segundo par de octetos al cluster impar (3) 40 00
- dé vuelta la palabra 00 40
- elimine el dígito de orden inferior 00 4

004 HEXADECIMAL es 4 en decimal es decir que el archivo continúa en el cluster 4 (que es el siguiente)

HEX	CLUSTER	OCTETOS INVERTIDOS	OCTETOS HEX	NRO. CLUSTER DECIM
2	2	03 40	40 03	3
3	3	40 00	00 40	4
4	4	05 60	60 05	5
5	5	60 00	00 60	6
6	6	07 80	80 07	7
7	7	80 00	00 80	8
8	8 ...			

Como usted verá cada cluster apunta al siguiente esto ocurre normalmente con un disco en el que no se han removido archivos como este que ha sido recién formateado, la tendencia es que un archivo ocupe clusters sucesivos.

El Directorio Raíz y los Subdirectorios

El directorio es un archivo que va actualizando el DOS, imprescindible para el rastreo de los restantes archivos en el disco. Cada archivo tiene una entrada asignada en el directorio que indica su nombre y atributos en distintos campos. Para DOS el nombre es una cadena de ocho caracteres seguidos de un punto y tres caracteres más denominados estos últimos 'extensión'. La extensión determina el tipo de archivo, por ejemplo DOCUMENT.TXT indicará que el archivo DOCUMENT es un archivo de texto.

El área de archivos o área de datos está dividida en clusters de igual tamaño (tamaño que varía según el tamaño del disco). Inicialmente los archivos se almacenan en clusters sucesivos a partir del número 2 pero cuando el usuario elimina ciertos archivos quedan huecos que podrán ser utilizados por un archivo nuevo. Es común que el hueco no sea lo suficientemente grande para alojar el nuevo archivo y, por lo tanto, se producirá un salto al próximo cluster vacío, de manera tal que permita alojar el resto de los datos del archivo que se desea almacenar. Este procedimiento se repite una y otra vez, quedando los archivos seccionados físicamente. Si bien es lógico y desable que los archivos aprovechen los huecos vacíos del soporte, recuperar un archivo particionado demora más que un archivo alojado en clusters sucesivos.

Como dijimos para poder acceder a un archivo guardado en un disco. En el directorio se guarda el nombre del archivo en el disco, y un número del cluster de comienzo que permita su ubicación en el soporte.

El DOS permite que un directorio contenga además de archivos otros directorios (que dependen del primero) estableciendo así una organización jerárquica que se puede identificar como un árbol invertido. De este modo se pueden separar archivos utilizados en aplicaciones distintas en directorios distintos.

Como en todo árbol el directorio principal se denomina raíz (root). De él dependen archivos u otros directorios hijos del raíz; el comando TREE permite ver la estructura jerárquica de un disco flexible o rígido

Práctico 14 :**Edición de un Archivo y Análisis de la Información del Directorio**

- Ingrese el comando de DOS MD (Make Directory)

A:>MD TPSUBDIR <ENTER>

- Realice las siguientes tareas :

A:>C:

C:>EDIT <ENTER>

con la tecla <ESC> elimine el mensaje de su editor

con la tecla <ALT> vaya al menú y seleccione la opción Archivo o File

con la tecla de flecha hacia abajo seleccione la opción Nuevo o New

Ingrese la siguiente leyenda :

ESTE ES UN ARCHIVO ASCII. CREADO CON EL EDIT.

con la tecla <ALT> vaya al menú y seleccione la opción Guardar como.. o Save as...

ingrese A:/TPFILE <ENTER>

con la tecla <ALT> vaya al menú y seleccione la opción Salir o Exit

Usted ha creado un archivo de tipo texto con la ayuda del editor de DOS EDIT

- Ingrese el siguiente comando

C:\TREE A: /F<ENTER>

- Ingrese el siguiente comando

C:\DIR A: /A/S

Deberá aparecer en su pantalla :

El volumen de la unidad A es DISCO2

El número de serie del volumen es 1E23-15CE

Directorio de A:\

IO	SYS	223.148	24/08/95	9:50	IO.SYS
MSDOS	SYS	9	24/08/95	9:50	MSDOS.SYS
COMMAND	COM	95.334	24/08/95	9:50	COMMAND.COM
DRVSPACE	BIN	71.559	24/08/95	9:50	DRVSPACE.BIN
TPSUBDIR	<DIR>		31/07/96	14:34	TPSUBDIR
		4 archivo(s)	390.050 bytes		

Directorio de A:\TPSUBDIR

.	<DIR>	31/07/96	14:34	.
..	<DIR>	31/07/96	14:34	..
TPFILE	46	31/07/96	14:37	TPFILE
		1 archivo(s)	46 bytes	

Número total de archivos en la lista:

5 archivo(s) 390.096 bytes

3 directorio(s) 821.760 bytes libres

C:\DOS>

El comando DIR del DOS permite ver información adicional acerca de sus directorios y archivos, el parámetro /P le permite controlar la edición del comando DIR de una pantalla por vez. Este comando es atendido por un servicio del DOS que consulta el archivo directorio.

Como su nombre lo indica en realidad es un archivo-tabla cada entrada a la tabla representa un archivo o un subdirectorio con 32 bytes de información que se deben analizar así :

00H-07H Nombre del archivo.

08H-0AH Nombre de la extensión.

0BH Byte de atributos.

0CH-15H Area reservada.

16H-17H Hora de última modificación.

18H-19H Fecha de última modificación.

1AH-1BH Número de cluster donde comienza el archivo

1CH-1FH Tamaño del archivo

NOMBRE Contiene la representación ASCII del nombre asignado al archivo cuando fue creado. Como se puede ver la longitud es de 8 octetos. El primer octeto de este nombre puede tener significados que indiquen el estado del archivo, por ejemplo cuando el Archivo comienza con el caracter ASCII :

E5 el archivo fue borrado.

2E es la primera entrada de la tabla-directorio de un subdirectorio

2E2E es la segunda entrada de la tabla-directorio de un subdirectorio

EXTENSION Contiene la representación ASCII de 3 caracteres de extensión que representan una característica del archivo; un archivo fuente assembler tiene la extensión .ASM, uno basic .BAS, un ejecutable .EXE; la extensión se declara también al momento de creación del archivo separada del nombre con un punto EJEMPLO.ASM.

ATRIBUTOS. Los Archivos son estructuras que admiten operaciones propias de archivos abrir, cerrar, leer, escribir, crear, borrar (open , close, read, write, create, delete). Estas operaciones pueden ser reguladas por los atributos asociados al archivo. El byte atributo es una combinación de 8 bits ; cada uno representa uno atributo.

8	7	6	5	4	3	2	1
---	---	---	---	---	---	---	---

1. Read-Only El archivo puede ser abierto cerrado, leído pero no puede ser borrado ni escrito.
2. Hidden (Oculto) El archivo existe pero no se muestra en las búsquedas normales dentro del directorio.
3. System pertenece al DOS y está oculto para búsquedas normales.
4. Indica que este archivo identifica el soporte es decir que este archivo es una cadena de caracteres denominada "volume" este archivo pertenece al directorio raíz.
5. Indica que este no es un archivo sino un subdirectorio.
6. Este bit vale 1 cuando el archivo ha sido creado o sufre una escritura, un programa llamado BACKUP que se utiliza para copiar
7. archivos de resguardo lo vuelve a cero cuando produjo la copia, para no volver a BACKUParlo si no hay una nueva modificación.
8. Se ignoran.

0CH-15H AREA RESERVADA.

HORA DE ULTIMA MODIFICACION Contiene su valor y representa horas (0-23), minutos (0-60), segundos (0-29 cada S equivale a dos segundos)

HHHHHMMMMMMSSSSS

FECHA DE ULTIMA MODIFICACION. Contiene su valor y representa año (0-119), que se le suman a la base 1980 para obtener el año real; mes (1-12) y día (1-31).

AAAAAAMMMMMDDDDD

NÚMERO DE CLUSTER donde comienza el archivo

TAMAÑO DEL ARCHIVO se declara en estos octetos con el formato de bytes invertidos.

Para completar el estudio del directorio se analiza el contenido del directorio en DISCO2, siendo importante que usted analice en la tabla cada entrada y verifique la información obtenida con el comando DIR A:

- Ingrese:

```
A:\>C:
C:\>DEBUG <ENTER>
L 1000:0 0 F 1
D 1000:0 <ENTER>
D <ENTER>
D <ENTER>
D <ENTER>
```

presione la tecla <Print Screen> a medida que vaya completándose su pantalla

Aparecerá un listado así :

```
C:\>debug
-L 1000:0 0 F 1
-D 1000:0
1000:0000 49 4F 20 20 20 20 20 20 53 59 53 27 00 00 00 00 IO SYS'....
1000:0010 00 00 00 00 00 00 40 4E-18 1F 02 00 AC 67 03 00 .....@N....g..
1000:0020 4D 53 44 4F 53 20 20 20 20 53 59 53 27 00 00 00 00 MSDOS SYS'....
1000:0030 00 00 00 00 00 00 40 4E-18 1F B6 01 09 00 00 00 .....@N.....
1000:0040 43 4F 4D 4D 41 4E 44 20-43 4F 4D 20 00 00 00 00 COMMAND COM ....
1000:0050 00 00 00 00 00 00 40 4E-18 1F B7 01 66 74 01 00 .....@N....fl..
1000:0060 44 52 56 53 50 41 43 45-42 49 4E 27 00 00 00 00 DRVSPACEBIN'....
1000:0070 00 00 00 00 00 00 40 4E-18 1F 72 02 87 17 01 00 .....@N..r....
-d
1000:0080 44 49 53 43 4F 32 20 20-20 20 20 28 00 00 00 00 DISCO2 ^ (...
1000:0090 00 00 00 00 00 00 58 70-FF 20 00 00 00 00 00 00 .....Xp. ....
1000:00A0 54 50 53 55 42 44 49 52-20 20 20 10 00 73 54 74 TPSUBDIR ..sTt
1000:00B0 FF 20 FF 20 00 00 54 74-FF 20 FE 02 00 00 00 00 ...Tt. ....
1000:00C0 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 .....
1000:00D0 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 .....
1000:00E0 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 .....
1000:00F0 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 .....
```

-q

Ahora :

- Obtenga su propio vuelco.
- Determine el nombre y la extensión de cada uno de los archivos y el nombre de los subdirectorios que encuentre.
- Determine los atributos de algunos de ellos.
- Determine la fecha y hora de creación de los mismos.
- Determine el cluster de inicio de cada uno de ellos. Considere que este número está relacionado con el número de sector pero no lo es p.e. si el número de cluster es 0002 (primero del área de datos) referencia al sector 1D (29), no olvide que el sector 0 y los sectores 1 a 28 fueron tomados para guardar las FAT's y el directorio raíz.
- Rastree en el vuelco de su FAT algún archivo mencionados en su directorio hasta encontrar su cluster de fin de archivo marcado con FFF. Luego en función de la cantidad de clusters ocupados calcule cuantos octetos ocupó el archivo y verifíquelo con un comando DIR.
- Investigue el formato lógico para un disquete 3.5

Otra particularidad a tener en cuenta es la siguiente. El sistema operativo actúa de mediador en cada acceso a disco pedido por el programa que se está ejecutando, esto es así debido a que el procedimiento de acceso requiere de ciertas operaciones, que de no existir su intervención, deberían definirse en el programa cada vez que tuviera lugar una lectura o grabación. Podemos resumir estas operaciones según sea una creación de archivo, una lectura o grabación o su eliminación del soporte.

Creación:

Se accede secuencialmente al directorio comparando el nombre del archivo con cada una de sus entradas. Si el archivo existe no podrá ser creado y como consecuencia existirá la opción de reemplazo de uno por el otro, previa consulta al usuario. Si el archivo no existe se le asigna una entrada al directorio y se busca en la FAT un cluster con número cero (vacío) para que le sea asignado. Se graba en el área de datos el primer cluster y se busca en la FAT otra entrada con cero actualizando, con este número de entrada, la entrada anterior para producir el encadenamiento, así hasta el final del archivo marcando una entrada a la FAT con todos unos.

Lectura/grabación:

Se accede secuencialmente al directorio comparando el nombre del archivo a leer o grabar. Si se quiere leer y no existe provoca un error avisando al usuario (FILE NOT FOUND), si existe se accede a los clusters de datos consultando alternativamente la FAT.

Remoción de un archivo:

Se accede secuencialmente al directorio comparando el nombre del archivo con cada una de sus entradas, si no ocurre equiparamiento se produce un mensaje del error al usuario y en caso contrario se marca la entrada del directorio indicando que ésta queda libre y a su vez, se liberan las entradas de la FAT que le estaban asignadas. Es importante que le quede claro el concepto de que el archivo "no se borra" físicamente del soporte, sino que los territorios ocupados quedan liberados gracias a la actualización del directorio y las entradas de la FAT regrabadas con el número cero.

Lo más lógico en todos los soportes es que el área de sistema se aloje en el cilindro que permita el menor tiempo de acceso a las cabezas lecto/grabadoras dado que ésta es siempre consultada para acceder al soporte. Este es el cilindro cuya distancia física de las cabezas es menor, mientras que los clusters de datos ocupan el resto de los cilindros.

Es importante recalcar que lo explicado hasta ahora es un ejemplo representativo de un disco específico, organizado lógicamente por un sistema operativo particular (el DOS). Otros soportes del mismo tipo se organizan de otra manera.

Este ha sido, quizás, el primer acercamiento que usted ha tenido con el reconocimiento interno del quehacer de su computadora. Queda ligado a su interés, seguir investigando el alcance de esta poderosa herramienta, que no dude, le dará un sinfín de satisfacciones.