# UNIVERSIDAD NACIONAL AUTONOMA DE MEXICO

## FACULTAD DE INGENIERIA

## DISEÑO DE UN CONTROLADOR DIGITAL CON UNIDAD DE DISCO

T E S I S

Que para obtener el Título de

**INGENIERO MECANICO ELECTRICISTA**

P r e s e n t a n

**ABEL GONZALEZ CANCELA**

**JAIME ALBERTO GASPAR ELIZONDO HUERTA**

**JOSE ANTONIO MAZA MAGNUSSEN**

FACULTAD DE INGENIERIA

U N A M

Director de Tesis: M. I. Lauro Santiago Cruz

MEXICO, D. F. 1995

# DISEÑO DE UN CONTROLADOR DIGITAL CON UNIDAD DE DISCO

*Agradecemos el apoyo y la amistad siempre brindados por:*

*Ing. Rodolfo Peters L.*

*e*

*Ing. Lauro Santiago C.*

*A mi mamá por el ejemplo que me dejó,*

*Con mucho cariño para mi papá por su enorme apoyo,*

*A Claudia, con todo mi amor, por su apoyo, comprensión y empuje,*

*A mis hermanos,*

*A Samuel.*

*Alberto Elizondo*

*A ti papá, porque nunca has dejado de estar conmigo.*

*A mi mamá por su amor.*

*A mi fiel esposa Gaby.*

*A Tabé, Mábel y Lenchita.*

*A Beto, Claudia, Jorge y Pepe por su amistad.*

*Abel González*

*A Gabriela por su amor y apoyo, por tu ayer, tu hoy y tu mañana,*

*A María Isabel,*

*A mi padres por su ejemplo, cariño y consejo,*

*A Bestemor y Bestefar por sus consejos,*

*A mis hermanos y amigos.*

*Jose Antonio Maza*

# INDICE

**INTRODUCCION**

**CAPITULO I PLANTEAMIENTO Y CONFIGURACION DEL SISTEMA**

**I.1) Planteamiento**

**I.2) Características del controlador**

**I.3) Enlace con dispositivos de medición**

**I.4) Interfaz con una unidad de discos flexibles**

I.4.1) Conceptos Generales de la Interfaz
I.4.2) Requerimientos de la Interfaz
I.4.3) Control de la Interfaz

**I.5) Interfaz con el usuario**

**I.6) Diagrama general de bloques**

**CAPITULO II MANEJO DE ARCHIVOS EN MS DOS**

**II.1) Conceptos generales de la estructura de un disco**

**II.2) Estructura en MS-DOS**

II.2.1) Area de "Boot"
II.2.2) Area de "Fat"
II.2.3) Area del Directorio Raíz
II.2.4) Area de Archivos

**CAPITULO III DISEÑO DEL CONTROLADOR**

**III.1) Descripción general del controlador**

**III.2) Principales componentes del controlador**

**III.3) Diagramas de la tarjeta controladora**

**CAPITULO IV PROGRAMACION DEL CONTROLADOR**

**IV.1) Diagrama general de flujo**

**IV.2) Rutina de inicio**

**IV.3) Manejador de disco y carga de programa**

IV.3.1) Rutina de despliegue de directorio
IV.3.2) Rutina para leer y ejecutar un archivo
II.2.3) Rutina para escribir a disco

**IV.4) Interfaz con el usuario**

**IV.5) Lineamientos para la programación del sensor**

**CONCLUSIONES**

**BIBLIOGRAFIA**

**ANEXOS**

# INTRODUCCION

El origen de esta tesis fue la necesidad de desarrollar un controlador digital flexible que controlará desde un sensor hasta una estación de muestreo almacenando los datos recabados por ésta.

Este trabajo surgió debido a que en el Instituto de Ingeniería de la UNAM se diseñan y desarrollan sistemas digitales de control con una finalidad específica, que al no ser flexibles no son aprovechados en proyectos nuevos.

El objetivo principal de este proyecto es el hacer una tarjeta flexible de bajo costo, con independencia en cuanto a capacidad de almacenamiento y por otro lado, compatible con los estándares del mercado para permitir el procesamiento de la información almacenada.

Este trabajo no intenta desarrollar los elementos para el control del sensor o estación de muestreo en específico ni para la captura de la información, que dependerá directamente del diseño de dicho dispositivo, sino más bien, crear la infraestructura de hardware y software necesarias para poder implementar el software de control y procesamiento que se desee en función de las necesidades particulares del sistema de medición y muestreo de que se trate.

La principal característica de este controlador será el poder almacenar información en un medio magnético (floppy disk) con formato MS-DOS y recibir información del mismo. Esto permitirá darle flexibilidad al diseño, ya que se podrá procesar la información almacenada en cualquier PC compatible y por otro lado, el hecho de que el controlador posea un medio magnético, permitirá hacer manejable su programación.

El "Device Driver" para el manejo de disco, será incluido en la ROM del sistema y se podrá tener acceso a él mediante una rutina de interrupción, de tal forma que para los desarrollos del software de control de un sensor o estación de muestreo, solamente será necesario llamar a dichas interrupciones para lograr el acceso a disco.

# CAPITULO I

# PLANTEAMIENTO
# Y
# CONFIGURACIÓN DEL SISTEMA

## I.1) PLANTEAMIENTO

Los sistemas de adquisición de datos de bajo costo que operan autónomamente son los instrumentos más eficientes y precisos de obtener información del medio ambiente que nos rodea.

Dado que el uso de estos sistemas autónomos de adquisición se ha multiplicado mucho en los últimos años, el Instituto de Ingeniería de la UNAM ha desarrollado controladores para la obtención de perfiles de distribución de agua y estaciones de muestreo para registrar información sísmica y climatológica.

Estos sistemas se adecuan a las necesidades específicas de estos proyectos pero carecen de flexibilidad al no poder ser utilizados para otro fin.

Todos los proyectos tienen en común la necesidad de controlar algún dispositivo externo que sense el entorno, así como almacenar esta información para poder ser analizada posteriormente.

A su vez, todos estos diseños tienen la limitante de no poder almacenar grandes cantidades de información y que el proceso de recuperación de la misma se realiza suspendiendo la obtención de datos.

La información obtenida por estos sensores era traducida y enviada vía serial hacia un PC., lo que implicaba transportar una PC al lugar de la adquisición y desarrollar y ejecutar un programa en la PC para la recepción de la información.

Por lo anterior, se planteó la necesidad de desarrollar un controlador flexible que pudiera adecuarse a las características de estos y otros proyectos permitiendo el almacenamiento y recuperación de grandes volúmenes de información.

## I.2) CARACTERISTICAS DEL CONTROLADOR

La idea básica de este diseño es contar con los elementos de salida para controlar el movimiento de sensores o estaciones de muestreo, tener la capacidad de recibir datos del mismo, y poder procesar y finalmente almacenar la información en un medio magnético con formato compatible con MS-DOS. También se prevee la posibilidad de recibir o enviar información vía serie a través de un puerto.

Además, el controlador contará con un teclado y un "display" que permita la comunicación elemental con el usuario, de tal manera que se pueda accesar un archivo y desplegarlo en el "display" o pedirle que tome instrucciones de algún archivo en disco.

La posibilidad de almacenar información en un medio magnético le da flexibilidad al sistema. Por un lado, le da independencia al permitir almacenar grandes cantidades de información, y por otro, ser versátil en su programación, ya que puede programársele en cualquier momento.

Se pensó en darle la característica de guardar la información en formato MS-DOS, ya que esto facilitaría el procesamiento de la información almacenada en una PC compatible a través de cualquier hoja de cálculo comercial, lo que haría aun más versátil su funcionalidad.

## I.3) ENLACE CON DISPOSITIVOS DE MEDICION

El controlador debe contar con elementos de enlace que le permitan controlar un dispositivo de medición o sensor. El diseño del controlador contempla elementos comunes de hardware para el control del uno o varios sensores diferenciándolos entre sí por software.

De esta forma, el controlador no sólo podrá determinar cuando mover o no un sensor, sino cuando efectuar una lectura a través de este dispositivo.

Esta tesis no contempla en su alcance, el desarrollo del software para controlar un sensor o estación de muestreo en específico, ya que esto depende de las características particulares de estos dispositivos.

## I.4) INTERFAZ CON UNA UNIDAD DE DISCOS FLEXIBLE

El punto más importante de esta tesis es la interfaz con una unidad de disco flexible en formato compatible con el sistema operativo MS-DOS.

Se consideraron dos alternativas para darle solución a esto. La primera de ellas es la opción de desarrollar todo el hardware necesario para hacer interfaz con una unidad de disco flexible. Esta opción además de resultar cara, implica alcances no contemplados en este proyecto, así como el limitar severamente la posibilidad de actualización de dicha interfaz, ya que se tendría que modificar tanto hardware como software.

La segunda opción, por la cual se decidió, es utilizar una tarjeta controladora de disco flexible para PC XT compatible, de tal forma que sólo se desarrolle el software de interfaz con dicha tarjeta controladora. Esto permite la posibilidad de reemplazar dicha tarjeta con la unidad de disco en caso de querer hacer una actualización al sistema.

### I.4.1) CONCEPTOS GENERALES DE LA INTERFAZ

A continuación se describen los requerimientos para hacer la interfaz con una tarjeta controladora de unidades de disco flexible de baja densidad (5 ¼ in.). Estos requerimientos son iguales para cualquier controladora de discos utilizada en una PC XT. Adicionalmente, se describirán las características de interfaz para una tarjeta controladora que maneje discos de alta densidad (3 1/2 in.) normalmente utilizados en las PC AT dejando la posibilidad en un futuro, si así lo requiere las necesidades del proyecto, de utilizar un medio magnético de mayor capacidad de almacenamiento.

### I.4.2) REQUERIMIENTOS DE LA INTERFAZ

Las tarjetas controladoras de discos flexibles basadas en el 82072 ocupan un "slot" sobre los canales de Entrada/Salida (E/S) de las computadoras personales.

Como es sabido, el canal de E/S de una PC AT difiere del canal de una PC XT, pero el conjunto de señales utilizado por la tarjeta controladora es el mismo en ambos casos.

La tarjeta controladora de disco requiere los recursos del procesador central, del DMA y del controlador de interrupciones.

El sistema de control de unidades de disco consiste en 5 elementos funcionales:

1) Interfaz entre la tarjeta controladora y el canal de E/S.
2) Generación de "Chip-Select"
3) Generación de Reloj para el 82072
4) Interfaz con el DMA
5) Interfaz con la unidad de discos

Los puntos 3 y 5 no son del interés de este trabajo, ya que internamente dentro de la tarjeta controladora se encuentra todo el hardware necesario para el manejo de la unidad de disco flexible así como la generación del reloj para el 82072.

La interfaz con el canal de E/S requiere de las siguientes líneas:

- 10 líneas de direcciones (A0 hasta A9)
- 8 líneas de datos (D0 hasta D7)
- 4 líneas para el control de escritura/lectura que son: MEMW^, MEMR^, IOW^, IOR^.
- 6 líneas para el control que son: RESET, T/C, AEN, DRQ2, DACK2^, IRQ6.

Las señales que tengan un " ^ " significa que son verificadas en estado bajo.

La interfaz con el DMA requiere de tres señales básicas que son T/C (Terminal Count), DRQ2 (Data Request en el canal 2 del DMA) y DACK2^ que es la línea de reconocimiento para dicho canal.

Cabe hacer notar que en un ambiente de PC, la transferencia de información de disco o hacia disco tiene que hacerse en modo byte-byte. Sin embargo en la tarjeta controladora como en otro tipo de sistemas se puede realizar transferencias de bloques completos de información (modo "burst").

La señal IRQ6, es utilizada por la tarjeta controladora para interrumpir al microprocesador. Más adelante veremos la función específica de dicha interrupción.

Para soportar las diferentes capacidades de "drivers" y los diferentes tiempos de transferencia de datos, existen 3 registros incorporados en la tarjeta controladora de discos flexibles. Estos tres registros son los siguientes:

- DRR (Data Rate Register)
- DIR (Digital Input Register)
- DOR (Digital Output Register)

**DRR (Data Rate Register)**
**(Controladoras de diskette de alta densidad exclusivamente).**

Este registro de sólo escritura y seleccionado en la dirección 3F7H especifica la unidad de disco y el tipo de disco utilizado. El valor de este registro es utilizado por la tarjeta controladora de alta densidad para seleccionar la velocidad de transferencia de datos y la velocidad de giro de la unidad de disco. A continuación se muestra una tabla con la decodificación para este registro.

| D1 | D0 | DESCRIPCION |
|---|---|---|
| 1 | 1 | RESERVADO |
| 1 | 0 | DD DRIVE DD DISKETTE |
| 0 | 1 | QD DRIVE DD DISKETTE |
| 0 | 0 | QD DRIVE QD DISKETTE |

DD = Doble Densidad y QD = Cuádruple Densidad

**Tabla I.1**

**DOR (Digital Output Register)**

Este registro es de sólo escritura y se encuentra en la dirección 3F2H. A continuación se presenta una tabla con el significado de cada uno de los bits que conforman este registro

| BITS | DEFINICION |
|---|---|
| 0 | Selección de Drive: 0 en este bit indica que el drive A está seleccionado, 1 indica que el drive B está seleccionado. |
| 1 | Reservado: Puede ser 0 ó 1 |
| 2 | Función de Reset: Cuando este bit es puesto a 1, la función de reset es deshabilitada |
| 3 | Habilita DMA e Interrupción: Cuando este bit es puesto a 1, las líneas para DMA y la de interrupción son habilitadas |
| 4 | Habilitador del Motor del Drive A: Cuando este bit es puesto a 1, la señal de habilitar el motor A es activada. Un "timer" es dedicado a realizar la función de deshabilitar el motor. El "timer" es inicializado en función del comando seleccionado. |
| 5 | Habilitador del Motor del Drive B: igual que el bit anterior |
| 6 | Reservado: Igual que el bit 1 |
| 7 | Reservado: Igual que el bit 1 |

**Tabla I.2**

**DIR (Digital Input Register)**

**(Controladoras de diskette de alta densidad exclusivamente).**

Este registro es de sólo lectura y el único bit usado por la tarjeta controladora de discos flexibles es el 7.

Este bit es utilizado exclusivamente cuando se está manejando un drive de alta densidad. La función primordial de este bit es informar al sistema que un

diskette ha sido cambiado (cuando está puesto a 1). Este bit es automáticamente puesto a 0 cuando se realiza una operación de búsqueda (Seek).

A continuación se presenta una tabla mostrando el mapa de direcciones con los registros mencionados.

| DIRECCION | TIPO DE ACCESO | REGISTRO | MODELO |
|---|---|---|---|
| [illegible] | | NO USADO | |
| [illegible] | | NO USADO | |
| [illegible] | ESCRITURA | DIGITAL OUTPUT REG. | XT/AT |
| [illegible] | | NO USADO | |
| [illegible] | LECTURA | MAIN STATUS REG.* | FDC |
| [illegible] | ESC./LEC. | DATA REGISTER * | FDC |
| [illegible] | | NO USADO | |
| [illegible] | ESCRITURA | DATA RATE REGISTER | AT |
| [illegible] | LECTURA | DIGITAL INPUT REG. | AT |

FDC = Floppy Disk Controler
* Estos registros son utilizados tanto en una XT ó AT

**Tabla I.3**

### I.4.3) CONTROL DE LA INTERFAZ

El FDC (Floppy Disk Controler) controla los motores de la unidad de discos y las cabezas, además de manejar el flujo de datos de y hacia los sectores del disco. Una tarjeta controladora puede manejar hasta 4 unidades de disco.

El FDC maneja 15 operaciones en total, de las cuales utilizaremos 8 :

- SPECIFY
- SENSE DRIVE STATUS
- SENSE INTERRUPT STATUS
- SEEK
- RECALIBRATE
- READ DATA
- WRITE DATA
- READ ID

**SPECIFY**: Este comando coloca los valores iniciales para cada uno de los tres temporizadores o timers. El "Head Unload Time" (HUT), el "Step Rate Time" (SRT) y el "Head Load Time" (HLT).

**SENSE DRIVE STATUS**: Este comando se utiliza cuando el procesador desea obtener el estado de las unidades de disco.

**SENSE INTERRUPT STATUS**: Una señal es generada por el Floppy Disk Controler dependiendo de situaciones específicas que pueden ser de utilidad para el programador.

**SEEK**: Este comando permite posicionar la cabeza en un "track" ó cilindro determinado.

**RECALIBRATE**: Este comando es utilizado para colocar la cabeza de la unidad de discos al "track 0" y tener un punto de inicio.

**WRITE DATA**: Comando utilizado para escribir información a disco.

**READ DATA**: Comando utilizado para leer información de disco.

**READ ID**: Este comando es utilizado para dar la posición actual de la cabeza en función del agujero índice que traen todos los discos.

Las operaciones en el Floppy Disk Controler se realizan en tres fases: la fase de comando, la de ejecución y la de resultado.

En la fase de comando, uno o más bytes son enviados al "Data Register" del FDC. La secuencia de bytes es estricta y varia dependiendo del comando. El FDC reconoce el comando e inicia la fase de ejecución. Finalmente, durante la fase de resultado, un número de bytes se genera para informarle al procesador el estado de la operación. Es imperativo que no exista error en el número de bytes enviados o leídos a o del "Data Register".

El número de bytes enviados o leídos depende de la operación que realice el FDC. Para el caso de lectura o escritura a un sector, se necesitan 7 bytes adicionales al comando en sí. Estos siete bytes indican track, cabeza, sector y cuatro bytes adicionales. El primero de los cuales es el número de bytes por sector (en el caso más utilizado es 2 para 512 bytes por sector). El segundo indica el número de sectores por track (9 para un floppy de 360 kbytes). Los

últimos dos bytes indican el "Gap Lenght" y el "Data Lenght" (Normalmente 2AH y FFH respectivamente).

A continuación se describe los pasos a seguir por el procesador para leer un sector determinado en un disco.

1) Enviar la señal para encender el motor y esperar a que estabilice su velocidad.
2) mover la cabeza del drive a la posición de inicio de lectura mediante la operación de búsqueda ("seek").
3) Monitorear el fin de la operación de búsqueda a través del cambio de estado del bit 7 del byte de "status" de la operación seek .
4) Inicializar al DMA para mover los datos a memoria.
5) Enviar las instrucciones al FDC .
6) Leer la información del FDC.
7) Apagar el motor.

En páginas anteriores mencionamos que la tarjeta controladora de discos flexibles utiliza la interrupción No.6 (IRQ6) para indicarle al procesador que terminó la operación de búsqueda a través de modificar el estado del bit 7 del byte de "status" de la operación seek en la BIOS DATA AREA localidad 0040:003E.

La IRQ6 es utilizada para evitar que el procesador esté continuamente leyendo el "Status Register" del FDC , ya que la operación de búsqueda es la que más tiempo ocupa en el acceso a disco y además su tiempo varía dependiendo

de donde se encuentre posicionada la cabeza,. Esto le permite al procesador realizar actividades síncronas en paralelo al movimiento de la cabeza del drive.

## I.5) INTERFAZ CON EL USUARIO

La interfaz contemplada para este diseño consta de un "display" alfanumérico de cristal líquido y un teclado estándar de matriz. Estos tienen la función de permitir la comunicación elemental entre el usuario y la tarjeta controladora de tal manera que se pueda realizar las siguientes operaciones:

| | | |
|---|---|---|
| 1) | **DIR** | -Desplegar nombres de archivo en disco- |
| 2) | **LEE Y EJECUTA** | -Pasar información de disco a memoria y ejecutar a partir de esa localidad- |
| 3) | **ESCRIBE** | -Pasar información de memoria a disco- |
| 4) | **RELOJ** | -Programar el reloj de tiempo real- |

Para lograr la máxima flexibilidad en el diseño, la interfaz con el usuario no se limitan al teclado y al "display", sino que abarca la comunicación vía serial con una computadora personal.

Si es que se requiere, se puede sustituir el teclado y "display" local de este controlador por el monitor y el teclado de la PC. Esto nos permite graficar y procesar la información en el lugar y en el momento de la adquisición de la misma.

## I.6) DIAGRAMA GENERAL DE BLOQUES

Integrando los elementos presentados en este capítulo, el diagrama general de bloques del controlador digital tiene las siguientes caracterísiticas.


**FIGURA I.1**

# CAPITULO II

# MANEJO DE ARCHIVOS EN MS-DOS

## II.1) CONCEPTOS GENERALES DE LA ESTRUCTURA DE UN DISCO

El almacenamiento en disco está basado en dos cosas principalmente: tecnología de grabado y un esquema de rápido acceso.

La tecnología utilizada es el grabado magnético. El grabado digital magnético es hecho en una superficie de material magnéticamente sensible, usualmente de una forma de óxido de cobre, lo cual le da su color característico café. El recubierto magnético es muy delgado y usualmente cubre a un material de soporte plástico de tipo "mylar".

La superficie del disco es tratada como una matriz de puntos. Cada uno de estos puntos son caracterizados como bits y podrán ser puestos a su equivalente magnético "0" ó "1". Debido a que la localización de estos puntos no está precisamente determinada, los esquemas de almacenamiento involucran marcas para reconocer posiciones. Esta es una de las razones por las cuales los discos tienen que ser "formateados" previamente antes de ser usados.

En lo que respecta al esquema de rápido acceso, existe una diferencia entre el concepto de cinta magnética, -que es escencialmente lineal porque la información tiene que ser almacenada en ésta del frente hacia atrás y no hay manera sencilla de saltar al centro de la misma-, y un disco rotatorio.

El disco rotatorio tiene dos características importantes que hacen posible el llegar a cualquier parte de su superficie rápidamente.

La primera es la rotación del disco. El disco gira sobre sí mismo a 300 R.P.M., es decir que cada 120 milésimas de segundo se tendrá acceso al mismo lugar en sentido circunferencial.

La segunda es el movimiento de la cabeza grabadora, la cual atraviesa el disco en forma radial de la parte más externa hasta la parte más interna del disco y viceversa. Este movimiento se realiza en un promedio de 100 milisegundos. Utilizando estos dos factores es como se puede llegar a cualquier parte del disco en un tiempo muy corto. Esta es la razón por la cual los discos son llamados unidades de almacenamiento de acceso aleatorio, ya que podemos accesar cualquier dato directamente sin necesidad de pasar a través de toda la información de manera secuencial como lo haríamos en una cinta magnética.

Para el uso en una computadora personal, el disco es dividido en anillos concéntricos, llamados cilindros o "tracks". El número de "tracks" varía dependiendo del tipo de disco.

Independientemente del número de "tracks" que posea un disco, estos son siempre identificados por un número, empezando por el "0" que es el track más alejado del centro del disco.

Al igual que la superficie del disco es dividida en anillos concéntricos, cada anillo concéntrico es dividido a su vez en partes llamadas "sectores" que son de un tamaño fijo.

Todas las lecturas o escrituras a disco son hechas en términos de sectores completos. Los sectores son identificados mediante números asignados a ellos.

Para los propósitos particulares de este proyecto, se utilizarán discos de doble densidad, es decir, el grabado al disco se realiza alternamente en ambas caras. Lo que implica contar con dos cabezas lectoras.

Los discos de doble densidad cuentan con 40 "tracks" por lado, 9 sectores por "track" y cada sector es de 512 bytes (figura II.1)

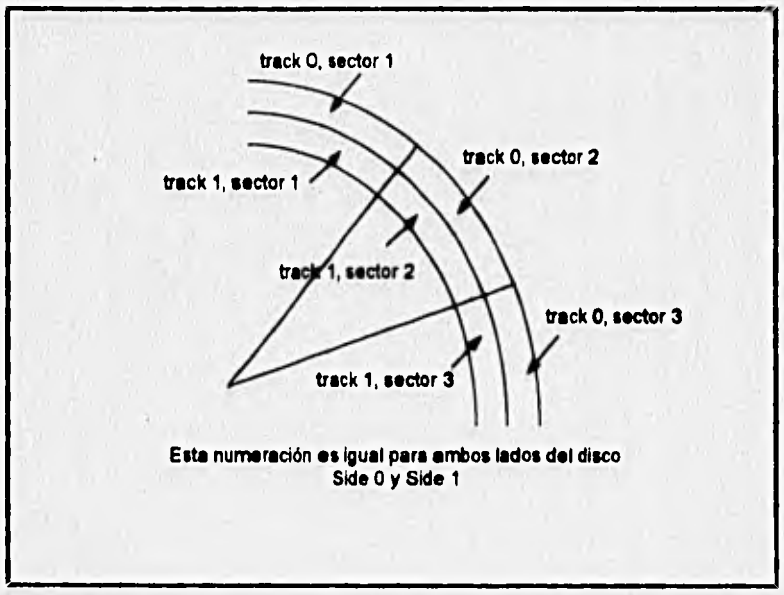


**Figura II.1**

## II.2) Estructura en MS-DOS

Como ya mencionamos anteriormente, el sector en un disco es la unidad fundamental para las operaciones del mismo. Para el sistema operativo MS-DOS, no existen tracks, ni lados, solamente sectores. DOS numera los sectores de un disco de manera secuencial empezando con "0" para el primer sector del track 0 del lado 0 y continuando con el segundo sector del mismo track y del mismo lado, cuando termina un track continua con el mismo track pero del otro lado del disco (figura II.2).

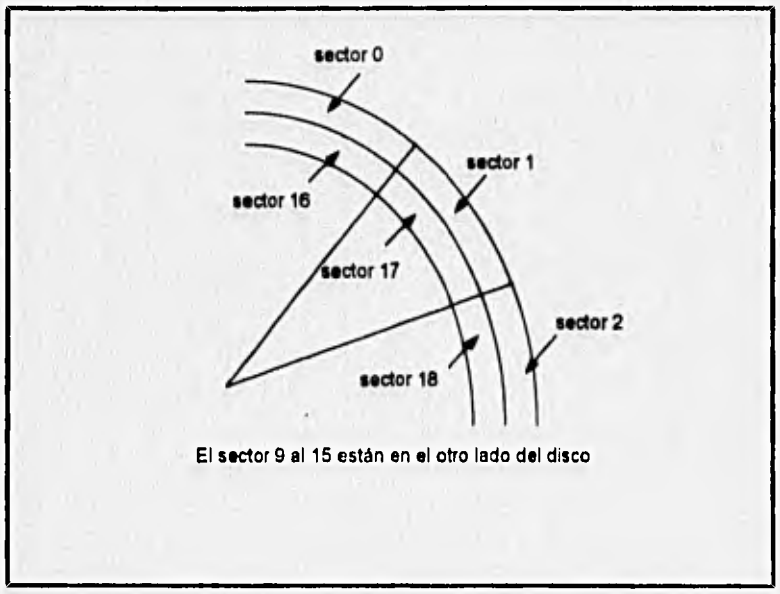


**Figura II.2**

Para organizar la información en un disco, DOS divide a éste en dos partes: Una parte pequeña, la cual DOS utiliza para mantener registro de información clave acerca del disco, y la zona mayor para los datos en sí. La primera es

conocida como área del sistema y a su vez es dividida en tres partes: área de "boot", área de "FAT" y área del directorio Raíz (Root).

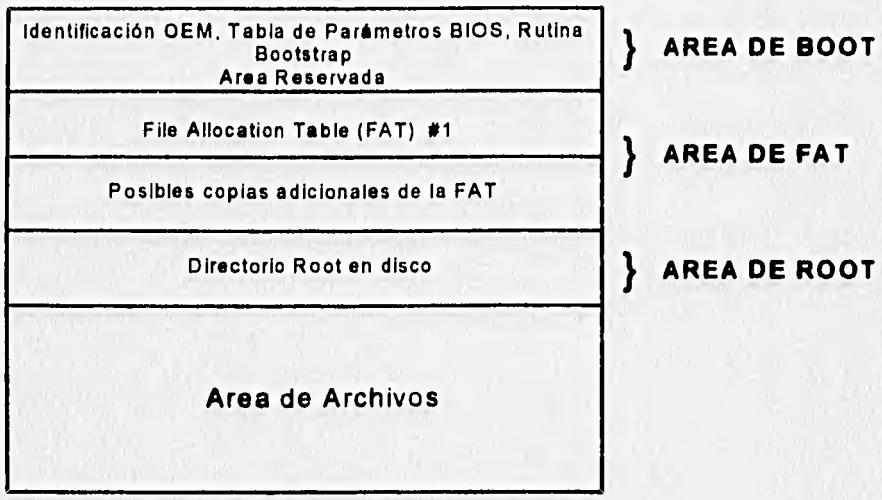


**Figura II.3**

### II.2.1) AREA DE BOOT

El área de "boot", contenida en el sector lógico "0", contiene toda la información indispensable acerca de las características del medio magnético (ver figura II.4).

El primer byte en el sector es siempre una instrucción de salto, que puede ser dentro del segmento (opcode 0E9H) seguida de un desplazamiento de 16 bits o un salto corto (opcode 0EBH) seguido de un desplazamiento de 8 bits y

después por un NOP (opcode 90H). Si ninguno de estos dos saltos está presente implica que el disco no está formateado.

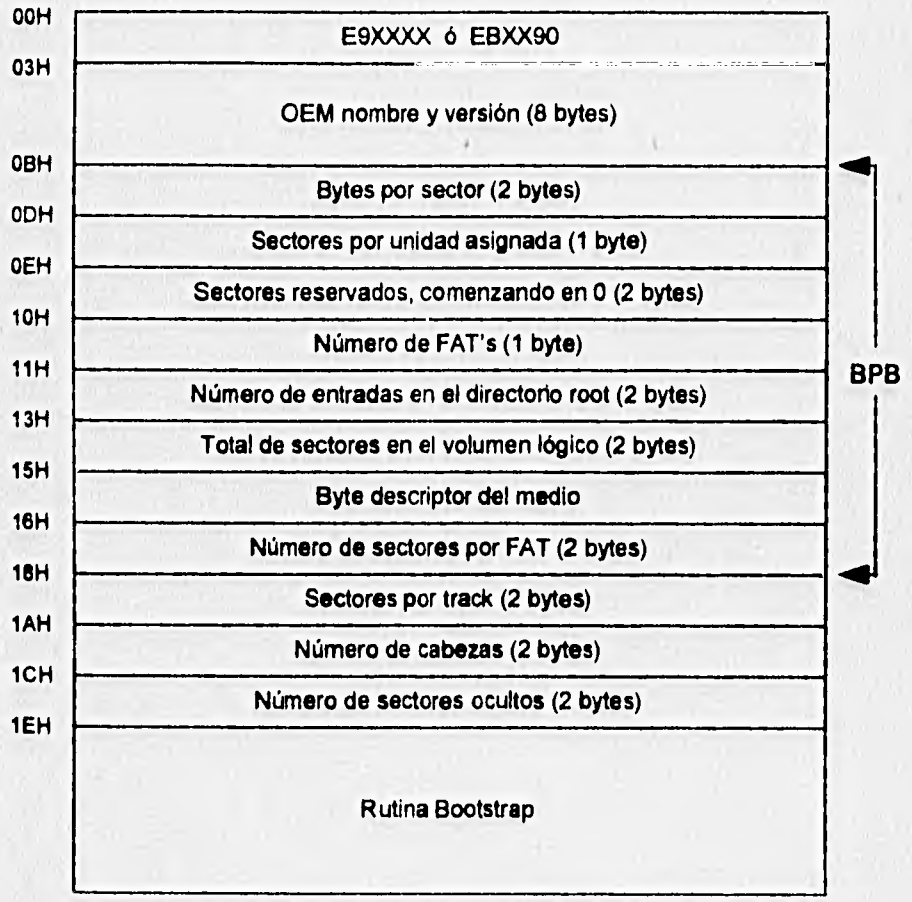


**Figura II.4**

Después de la instrucción inicial de salto, en el byte 03h, encontramos un campo de 8 bytes reservado para la identificación del programa que formateó el disco.

El tercer componente de esta sección es el Bios Parameter Block (BPB) localizada entre los bytes 0BH y el 17H. Esta estructura describe las características físicas del disco y permite al "device driver" calcular las direcciones físicas para un sector lógico dado. También el BPB contiene información que es usada para calcular la dirección y tamaño de cada una de las áreas de control del disco (área de FAT y área del directorio Raíz).

Después de BPB se encuentran tres palabras (offset 18H, 1AH y 1CH) que le especifican al "device driver" el número de secotres por FAT, de cabezas y de sectores ocultos.

El último elemento que compone el sector de boot (offset 1EH) es el programa de "bootstrap" que se encarga primitivamente de transferir el control al Basic Input Output System "BIOS" de máquinas compatibles con MS-DOS.

El sector de boot es parte del área reservada que puede ser de varios sectores. El tamaño de esta área reservada está escrito en el BPB en el desplazamiento 0EH.

### II.2.2) AREA DE "FAT"

Esta área debe sus siglas a "File Allocation Table", y es de suma utilidad para que el sistema operativo mantenga información acerca del área de datos. Esta área es simplemente una tabla de números, donde se relacionan para cada archivo, los sectores del área de datos asociados a éste.

Un archivo, al ser creado o modificado, se le asignan o reasignan sectores del disco en potencias de 2 conocidas como "allocation units" o clusters. El número de sectores por cluster está definido en el BPB en el byte 0DH.

La FAT está dividida en campos que corresponden directamente a clusters asignados en el disco. Estos campos son de 12 bits o de 16 bits según la versión del sistema operativo y de la capacidad del medio magnético de que se trate (12 bits si es disco flexible y 16 bits para discos duros).

Los primeros dos campos en la FAT son reservados, en el caso de discos flexibles donde la FAT es de campos de 12 bits, serán los tres primeros bytes. Para discos duros cuya FAT es de 16 bits, los dos primeros campos ocupan cuatro bytes. En ambos casos, los primeros 8 bits del primer campo reservado contiene una copia del "media descriptor byte" que también se encuentra en el BPB. El resto de los bits del primer campo y del segundo campo son 1.

Por el número almacenado en cada entrada de la FAT, se determina el estado que tienen cada uno de los clusters, que a su vez corresponden a sectores del disco. En la tabla II.1 se describe el posible contenido de los campos.

| VALOR | SIGNIFICADO |
|---|---|
| [illegible] | CLUSTER DISPONIBLE |
| [illegible] | CLUSTER RESERVADO |
| [illegible] | CLUSTER DAÑADO |
| [illegible] | ULTIMO CLUSTER DEL ARCHIVO |
| [illegible] | SIGUIENTE CLUSTER DEL ARCHIVO |

**Tabla II.1**

Cada entrada de archivo en el directorio del disco contiene su cluster de inicio. Este cluster inicial nos indica la posición dentro de la tabla de la FAT donde está almacenado el siguiente cluster del archivo. Esto se repite hasta localizar el cluster que señala el fin de archivo.

La forma en como se interpreta el contenido del área de Boot y de la File Allocation Table (FAT) se muestra a continuación.

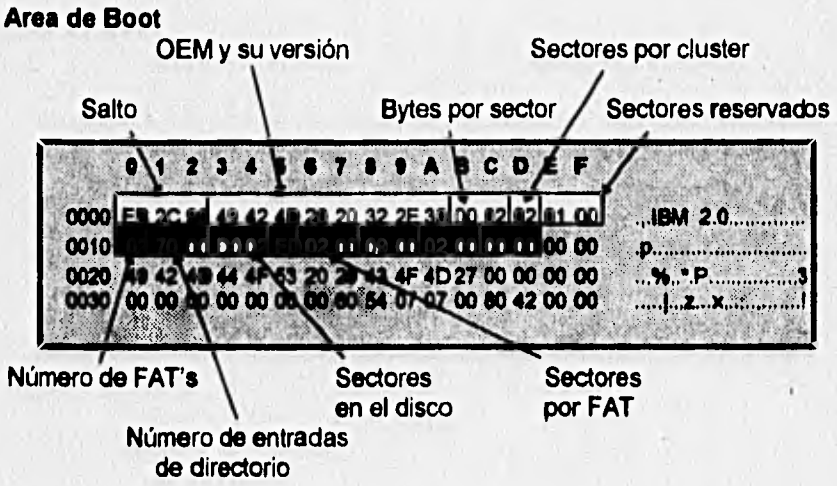


**Figura II.5**

Del Bios Parameter Block (BPB) se obtiene:

- 512 Bytes por sector (0200h)
- 2 sectores por cluster (02h)
- 2 sectores por FAT (0002h)
- 2 FAT's (02h)
- 112 entradas de directorio (0070h)

De esta información podemos calcular el número del sector de inicio lógico de cada una de las áreas de control del disco y del área de archivos contruyendo una tabla como sigue:

| AREA | LONGITUD | NUMERO DE SECTOR |
|---|---|---|
| [illegible] | 1 sector | 00 |
| [illegible] | 2 sectores | 01 - 04 |
| [illegible] | 7 sectores | 05 - 0Bh |
| [illegible] | | 12 sectores (0Ch) |

Por lo tanto, el primer sector del área de archivos es el 12 (0Ch)

**Tabla II.2**

Al leer el disco en el sector correspondiente al inicio de la FAT (sector 01) obtenido en la tabla anterior se puede observar la secuencia de los cluster de algunos archivos (Figura II.6)

**Area de FAT**

Reservado

```
     0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F
        yz cx ab yz cx ab yz  x
0000 FD FF FF 03 40 00 05 60 00 FF 8F 00 09 A0 00 0B   ...#.'..........
0010 C0 00 0D E0 00 0F 00 01 11 20 01 13 40 01 15 60   ..........  .#.....'
0020 01 17 F0 FF 19 A0 01 1B C0 01 1D E0 01 1F 00 02   ................
0030 21 20 02 23 40 02 25 60 02 27 80 02 29 F0 FF 00   ................
0040 00 00 00 00 00 00
```

**Figura II.6**

En la figura anterior se observa que los primeros dos clusters (formados por los 3 primeros bytes) son reservados. El primer archivo inicia a partir de ellos en el cluster 2, y siguendo la secuencia "xyz" y "abc" se relacionan los siguientes clusters como se muestra en la siguiente tabla:

| CLUSTER | 02 | 03 | 04 | 05 | 006 |
|---|---|---|---|---|---|
| CONTENIDO | 003 (xyz) | 004 (abc) | 005 (xyz) | 006 (abc) | FFF (xyz) |

**Tabla II.3**

### II.2.3) AREA DEL DIRECTORIO RAIZ

El directorio raíz puede ser visto como un catálogo que describe el contenido de un disco. Esta área lleva los nombres de los archivos dentro del área de datos, así como el nombre de los archivos especiales llamados subdirectorios.

El tamaño y la posición del directorio raíz es fija y está determinada por el formato del disco. El número de entradas en el directorio raíz y la localización del mismo pueden ser determinados mediante la información del BPB como se mostró con el ejemplo anterior.

Cada disco tiene un único directorio raíz, en el cual puede haber desde ningún subdirectorio hasta tantos como el disco pueda almacenar. Estos

subdirectorios a su vez pueden contener tantos subdirectorios como se deseen (figura II.7)

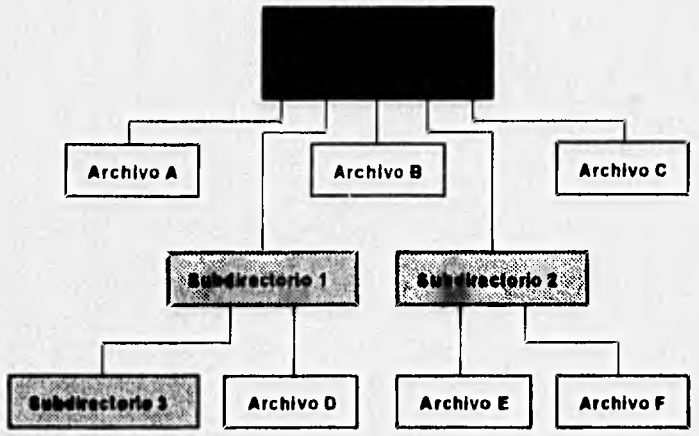


**Figura II.7**

Cada entrada del directorio raíz, ya sea nombre de archivo o nombre de subdirectorio, tiene un espacio único de 32 bytes. En la siguiente figura vemos las características de esta entrada.

| BYTE | DESCRIPCION | COMENTARIOS |
|---|---|---|
| [illegible] | NOMBRE DEL ARCHIVO | NOTA 1 |
| [illegible] | EXTENSION | |
| [illegible] | ATRIBUTOS ARCHIVO | NOTA 2 |
| [illegible] | RESERVADO | |
| [illegible] | TIEMPO | NOTA 3 |
| [illegible] | FECHA | NOTA 4 |
| [illegible] | CLUSTER INICIAL | NOTA 5 |
| [illegible] | TAMAÑO ARCHIVO | 4 BYTES |

**Tabla II.4**

Notas a la Tabla II.4:

**NOTA 1:** El primer byte del campo del nombre de archivo puede contener la siguiente información:

| VALOR | SIGNIFICADO |
|---|---|
| 00H | LA ENTRADA A ESTE DIRECTORIO NUNCA HA SIDO USADA, FIN DEL DIRECTORIO |
| 05H | EL PRIMER CARACTER DEL NOMBRE DE ARCHIVO ES ACTUALMENTE E5H |
| 2EH | LA ENTRADA ES UN ALIAS PARA EL ACTUAL SUBDIRECTORIO. SI EL SIGUIENTE BYTE ES 2EH, ENTONCES EL CAMPO DEL CLUSTER CONTIENE EL NUMERO DE CLUSTER DEL DIRECTORIO PREDECESOR (CERO SI EL DIRECTORIO PREDECESOR ES EL DIRECTORIO RAIZ) |
| E5H | EL ARCHIVO HA SIDO BORRADO |

**NOTA 2**: El byte de atributos es codificado como sigue:

| BIT | SIGNIFICADO |
|---|---|
| 0 | SOLO LECTURA |
| 1 | ARCHIVO OCULTO |
| 2 | ARCHIVO DEL SISTEMA |
| 3 | ETIQUETA DEL VOLUMEN |
| 4 | SUBDIRECTORIO |
| 5 | ARCHIVO (ENCENDIDO CUANDO EL ARCHIVO ES MODIFICADO) |
| 6 | RESERVADO |
| 7 | RESERVADO |

**NOTA 3**: El campo del tiempo se codifica como sigue:

| BITS | CONTENIDO |
|---|---|
| 00H-04H | NUMERO BINARIO DE MULTIPLOS DE 2 SEGUNDOS (0-29, CORRESPONDE A 0-58 SEGUNDOS) |
| 00H-04H | NUMERO BINARIO DE MINUTOS (0-59) |
| [illegible] | NUMERO BINARIO DE HORAS (0-23) |

**NOTA 4:** La codificación del campo de la fecha es:

| BITS | CONTENIDO |
|---|---|
| 00H-04H | DIA DEL MES (0-31) |
| 00H-04H | MES (1-12) |
| 00H-[illegible] | AÑO (ASOCIADO A 1980) |

**NOTA 5**: El campo del tamaño del archivo es interpretado como un entero de 4 bytes, con los dos bytes menos significativos almacenados primero.

Si el disco es utilizado para cargar el sistema operativo, las dos primeras entradas en el directorio raíz siempre son para los archivos del MS-DOS BIOS y del MS-DOS KERNEL.

También el directorio raíz puede contener una entrada especial llamada etiqueta del volumen, la cual da el nombre a todo el disco.

En la figura II.8 se muestra el primer sector de un directorio raíz:

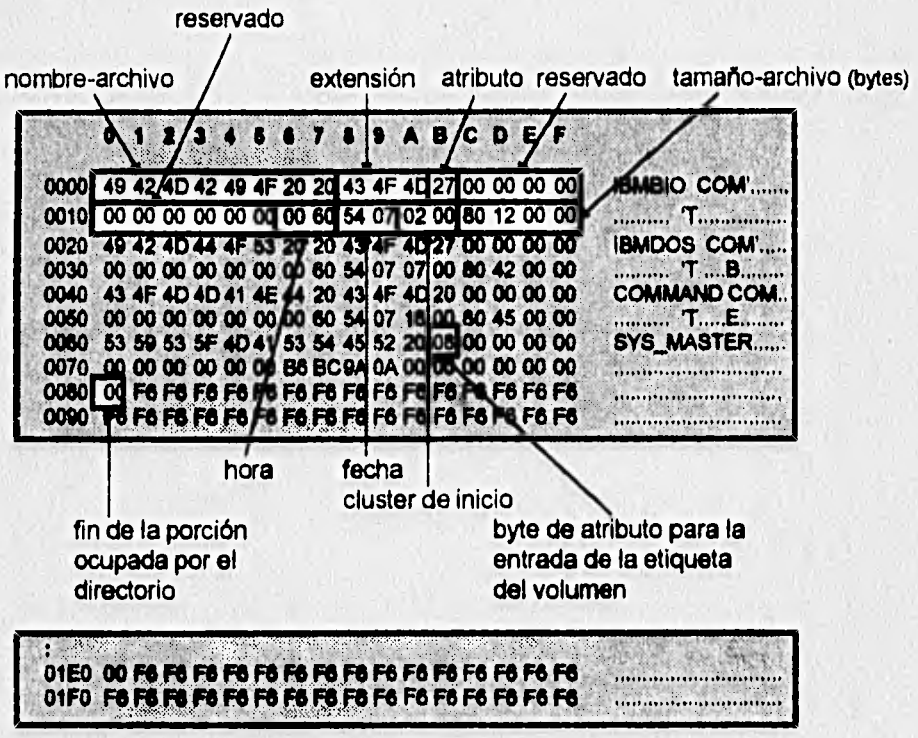


**Figura II.8**

Como mencionamos, los subdirectorios son un tipo de archivo especial que pueden contener descriptores de archivos de datos o apuntadores a otros subdirectorios. Una entrada de este tipo de archivo se caracteriza por tener el bit 4 puesto en 1 del byte de atributos y 0 en el tamaño de archivo. El campo del número de cluster apunta al primer cluster que implementa el subdirectorio.

Los cluster adicionales sólo pueden ser encontrados siguiendo la cadena de clusters a través de la FAT. En la siguiente figura se muestra la entrada a un subdirectorio.

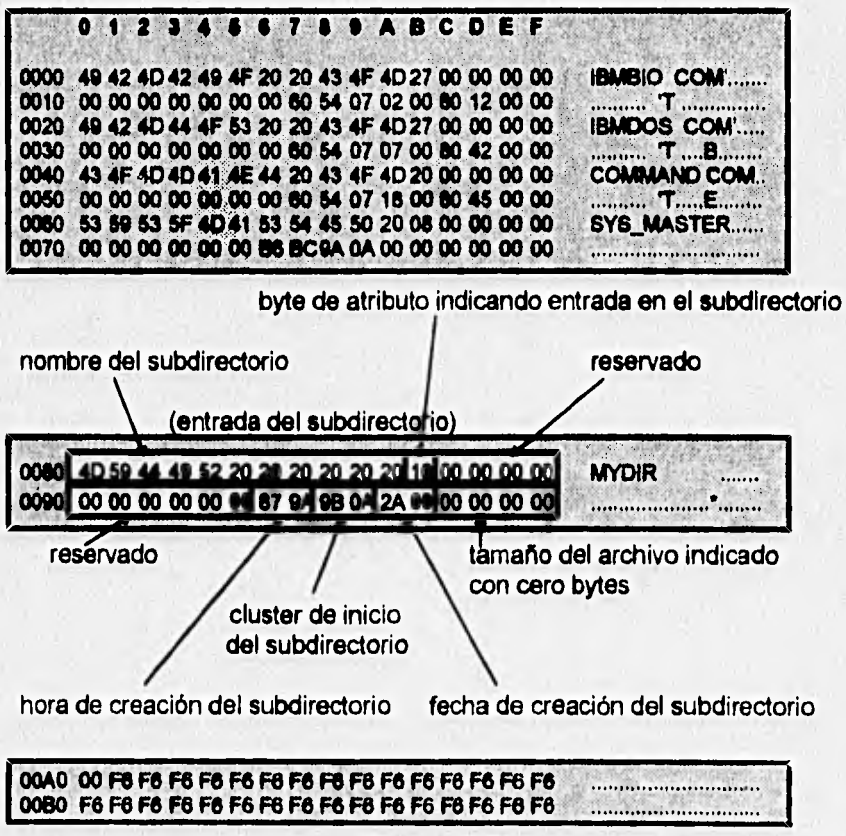


**Figura II.9**

Dentro de un subdirectorio, el formato de entrada para un archivo o para otro subdirectorio es exactamente igual que en el directorio raíz, es decir que las entradas son únicas y de 32 bytes.

Existen dos entradas especiales "." y "..", que todos los subdirectorios tienen y que se generan al momento de su creación y no pueden ser borradas. El punto se refiere al subdirectorio actual con la característica de que el campo de cluster apunta al cluster en el cual el subdirectorio se encuentra. El doble punto se refiere al subdirectorio predecesor donde el campo del cluster apunta al primer cluster del directorio predecesor. Si el directorio predecesor es el raíz, este campo es cero. A continuación se muestra el primer block de un subdirectorio.

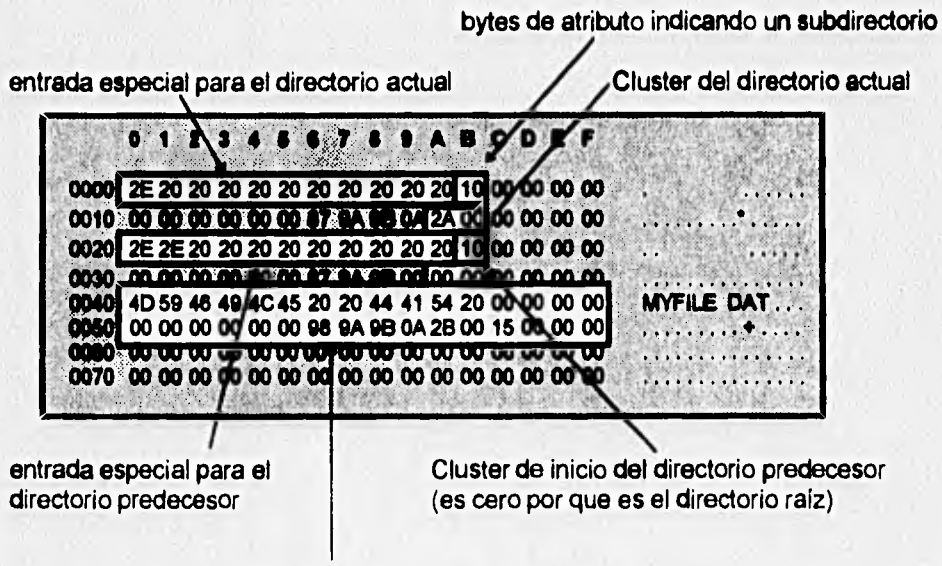


**Figura II.10**

# CAPITULO III

# DISEÑO DEL CONTROLADOR

## III.1) DESCRIPCION GENERAL DEL CONTROLADOR

El controlador está compuesto por un procesador 8088, controladores de acceso directo a memoria, controladores de interrupciones y del "driver". Además cuenta con dispositivos de entrada y salida en serie y paralelo, memorias ROM y RAM y un reloj de tiempo real. Así como elementos para interactuar con el usuario: un "display" de cristal líquido y un teclado de matriz.

En la figura III.1 se muestra un diagrama de bloques del controlador:.

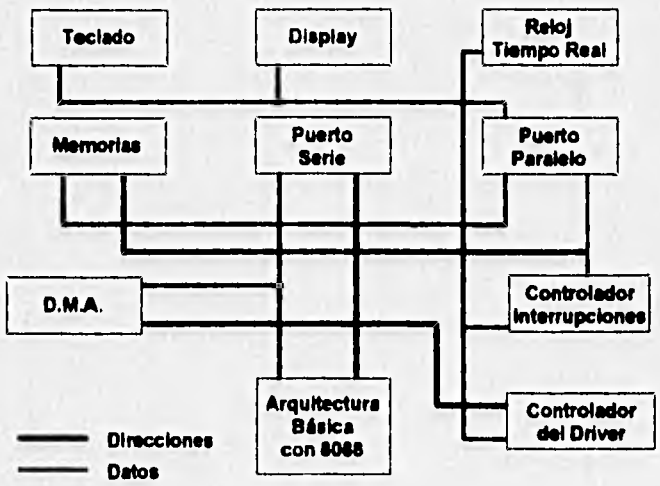


**Figura III.1**

Aunque en la figura III.1 no se muestra una computadora personal, se puede utilizar ésta para sustituir al "display" y al teclado de matriz en la interacción con los usuarios.

El controlador se diseñó usando básicamente integrados de los denominados "LSI" (Integración en gran escala) y complementando con integrados de menor complejidad, encargados de "adaptar" las señales provenientes de integrados de familias distintas.

En contraste con los componentes mayores, que son integrados LSI o VLSI construidos típicamente con tecnología MOS, o alguna pequeña variación de ésta, los integrados que realizan la "adaptación" suelen ser constituidos con tecnología bipolar TTL, con baja o media escala de integración (SSI o MSI).

Gracias a esta tecnología, el sistema puede codificar y decodificar las señales de control y separar y combinar señales de direcciones y de datos provenientes de los chips procesadores y controladores.

En los sistemas actuales se hace imprescindible el poder combinar estas señales (control, datos y direcciones) debido a que, en general, hay más señales necesarias que líneas para transportarlas.

## III.2) PRINCIPALES COMPONENTES DEL CONTROLADOR

### Microprocesador 8088

El microprocesador que se escogió para desarrollar este sistema fue el 8088 ya que es de bajo costo, está ampliamente disponible en el mercado nacional y reúne las características necesarias para el diseño.

El 8088 es un microprocesador con atributos tanto de procesadores de 8 como de 16 bits. Está diseñado para operar autogenerando sus señales del bus de control (modo mínimo) o para depender de un controlador de buses externo como el 8288 (modo máximo).

La configuración que se utiliza es la mínima, porque ésta soporta los requerimientos del controlador además de simplificar el diseño.

Entre las principales características de este procesador es que tiene un bus de direcciones de 20 bits, lo que le permite accesar hasta un millón de bytes direccionándolos de 00000(H) a FFFFF(H) divididas en segmentos de 16 bits (64 kbytes).

Todos los accesos a un segmento de memoria están referenciadas a la dirección base donde inicia dicho segmento y a incrementos tomados a partir de la base.

Otra característica es que este procesador comparte el bus de direcciones con el bus de datos (A0-A7 con D0-D7), esto se hace mediante un multiplexado en el tiempo, es decir, se envian señales distintas en tiempos diferentes por las mismas líneas. Por ejemplo, durante el acceso a memoria, la dirección se envia en el primer ciclo de reloj y los datos en los ciclos posteriores.

Sin embargo, como las memorias requieren que la dirección y el dato se presenten simultáneamente, es necesario contar con lógica intermedia que separe y guarde temporalmente tanto el dato como la dirección, y usar

determinadas señales del procesador para diferenciar a las direcciones de los datos y si estos se leen o escriben a un puerto o memoria. Estas señales de control que genera el procesador son seis y se agrupan en el bus de control del procesador.

De estas señales: Address Latch Enable (ALE) y Data Enable (DEN) se utilizan para que la lógica intermedia pueda determinar si la información enviada es una dirección o un dato. Gracias a ellas, parte de la circuitería externa al CPU puede trabajar con las direcciones e ignorar los datos, mientras que otra parte de la misma circuitería ignora las direcciones para guardar la información sobre los datos.

Las otras señales que componen el bus de control son: Data Transmition/Reception (DT/R^), Input Output/Memory (IO/M^), Read (RD^) y Write (WR^). Todas ellas utilizadas para controlar el flujo de datos y direcciones de y hacia el procesador.

Las señales que utiliza el procesador para ceder ordenadamente el control del bus del sistema al DMA ante una solicitud de éste son HOLD y HLDA.

El microprocesador reconoce dos tipos de interrupciones, las de software y las de hardware. Esta última, se clasifica en mascarables y no mascarables. La no mascarable está ligada a la señal NMI y tienen una prioridad superior a las mascarables que se reciben a través de la señal Interrup Request (INTR).

Las interrupciones por software se reciben a través de código máquina, relacionándolas con una tabla de hasta 256 elementos de 4 bytes cada uno, que contienen el apuntador a la dirección donde se localiza el programa de servicio a esa interrupción. Esta tabla se localiza en las localidades absolutas de 0 a 3FFH que es el primer kbyte de memoria.

## Generador de Pulsos de Reloj 8284

El Generador de Pulsos de Reloj 8284 genera cuatro señales importantes para el controlador.

La primera, CLK, que son los pulsos de reloj que determinan la velocidad de funcionamiento del sistema (los pulsos generados tienen un ciclo de trabajo de 33%). Esta señal es generada a una frecuencia de 4.77 MHz por medio de un cristal de 14.318 MHz conectado al generador.

La de RESET, que inicializa al procesador, y ejecuta las instrucciones empezando con la localidad de memoria predeterminada (FFFF0H). En algunos casos, permite interrumpir un programa que ha entrado en un bucle infinito.

La tercera, READY, que sincroniza al procesador con los dispositivos de más lento acceso.

Y la de PCLK (Reloj Periférifo), cuya frecuencia es la mitad de la de CLK, con un ciclo de trabajo de 50% del período.

## Controlador Programable de Acceso Directo a Memoria 8257

El 8257 es un controlador programable de cuatro canales de acceso directo a memoria, que permite la tranferencia de bloques de información de hasta 64 kbytes entre la memoria y un dispositivo periférico (un puerto o el controlador del "driver") sin la intervención del procesador.

Este componente recibe las solicitudes del dispositivo periférico a través de las señales de entrada DRQ (0-3). El 8257 solicita el control del bus del sistema al procesador con la señal HRQ, que el procesador recibe vía HOLD, y espera respuesta de éste a través de HLDA.

El dispositivo periférico es avisado de que su solicitud será atendida con las señales DACK (0-3).

El 8257, atiende las solicitudes en base a su programación inicial, que establece la forma en que serán atendidas las solicitudes. Estas pueden ser de una manera cíclica o de modo fijo, es decir, asignando siempre mayor prioridad al canal 0 y menor prioridad al canal 3.

En base a la prioridad predefinida, suministra al sistema la dirección base de la memoria que cada canal tiene asignado y genera las señales de control de escribir o leer tanto a memoria como al periférico para que la transferencia directa de información se lleve a cabo.

El 8257 mantiene el control del bus del sistema y repite la sequencia de transferencia de información mientras el periférico mantenga su solicitud o hasta que se hallan transmitido un número específico de bytes, que es cuando activa la señal TC (Terminal Count), informándole al periférico que la transferencia ha sido realizada.

Para programar este circuito, primero hay que activar su CS^, lo que lo habilita para trabajar en modo programación o "esclavo". Después, hay que inicializar el registro de dirección y de cuenta terminal para el canal que se va a utilizar y por último enviar la palabra de control que establece el modo de operación del canal. Esto se realiza en el programa de inicialización del controlador.

Los tres posibles modos de operación de cada canal son: lectura del DMA (transfiere datos de memoria a un periférico), escritura (transferencia del periférico a memoria), y verificación (validación de cualquiera de las operaciones anteriores). El modo de operación es selecccionado a través de los bits más significativos del registro de cuenta terminal del canal.

Cabe aclarar que para facilitar la programación de la tarjeta, se puede activar el AUTO LOAD a través de la palabra de control. Esto permite que el canal 2 sea utilizado en repetidas operaciones sin necesidad de volver a programar el registro de dirección. Esto lo hace internamente el DMA guardando las referencias del canal dos en el tres.

El 8257 mantiene el control del bus del sistema y repite la sequencia de transferencia de información mientras el periférico mantenga su solicitud o hasta que se hallan transmitido un número específico de bytes, que es cuando activa la señal TC (Terminal Count), informándole al periférico que la transferencia ha sido realizada.

Para programar este circuito, primero hay que activar su CS^, lo que lo habilita para trabajar en modo programación o "esclavo". Después, hay que inicializar el registro de dirección y de cuenta terminal para el canal que se va a utilizar y por último enviar la palabra de control que establece el modo de operación del canal. Esto se realiza en el programa de inicialización del controlador.

Los tres posibles modos de operación de cada canal son: lectura del DMA (transfiere datos de memoria a un periférico), escritura (transferencia del periférico a memoria), y verificación (validación de cualquiera de las operaciones anteriores). El modo de operación es seleccionado a través de los bits más significativos del registro de cuenta terminal del canal.

Cabe aclarar que para facilitar la programación de la tarjeta, se puede activar el AUTO LOAD a través de la palabra de control. Esto permite que el canal 2 sea utilizado en repetidas operaciones sin necesidad de volver a programar el registro de dirección. Esto lo hace internamente el DMA guardando las referencias del canal dos en el tres.

Este circuito también comparte el bus de direcciones con el bus de datos mediante un multiplexado en el tiempo. A diferencia del procesador, que comparte A0-A7 con D0-D7, el DMA lo hace en A8-A15 con D0-D7. También se utiliza lógica externa y las señales del bus de control del DMA para reconocer el tipo de información que está enviando.

Las señales de control son: Input/Output Read (I/OR), I/O Write (I/OW), Memory Read (MEMR), Memory Write (MEMW), Address Enable (AEN) y Address Strobe (ADSTB).

## Memorias RAM 2016 y HM62256

El controlador utiliza memorias de acceso aleatorio de tipo estático, es decir, que no requieren de un proceso de reescritura (refresco) para asegurar la integridad de la información.

Entre las características de la tarjeta controladora, se consideró el uso de memorias de acceso aleatorio de dos capacidades diferentes. Una de 1 kbyte por 8 bits utilizada para almacenar información propia del funcionamiento de la tarjeta controladora o de los dispositivos de medición. De esta forma el programador del sistema de adquisición, no necesita mezclar datos procedente de los dispositivos de medición, con información necesaria para su control.

Adicionalmente, se contemplan dos memoria de 32 kbytes por 8 bits para almacenar temporalmente la información proveniente de los sensores o estaciones de muestreo mientras ésta es guardada en el disco flexible.

Ambas memorias son de rápido acceso (100 ns), lo que asegura que no se requieran señales para su sincronización.

### Memoria EEPROM 27512

La memoria de sólo lectura tiene una capacidad de 64 kbyte por 8 bits, para almacenar el programa de inicialización y control básico de toda la tarjeta, así como el "device driver" para el manejo del controlador de disco. Esta memoria es el único elemento de la tarjeta controladora que debe reprogramarse en el caso de que se cambie el medio magnético de almacenamiento.

### Controlador de Interrupciones 8259

El diseño de este sistema requiere que los dispositivos de entrada salida como el teclado, "display", los puertos serie y paralelo y el controlador del drive sean atendidos eficientemente por el procesador, es decir, que el procesador minimice el tiempo de atención a estos dispositivos.

El método más común para atender este tipo de necesidades es el poleo. En este enfoque el procesador debe recorrer secuencialmente a todos los

dispositivos preguntándoles si alguno requiere ser atendido. Es fácil observar que al aumentar el número de dispositivos conectados a la tarjeta aumenta el tiempo que el procesador debe dedicar a barrer todos los dispositivos, disminuyendo, en la misma proporción, la velocidad de ejecución del programa principal.

Un método más deseable es aquel que permite que el procesador esté ejecutando su programa principal y que sólo lo suspenda para ejecutar la rutina de servicio de un dispositivo cuando el mismo dispositivo así lo solicita, volviendo posteriormente, a la rutina principal en el punto donde suspendió su ejecución. De esta manera, el número de dispositivos a atender puede aumentar sin afectar el tiempo de respuesta del sistema y sólo requiere que a todos los dispositivos se les asigne un orden con el cual serán atendidos por el procesador. Este método es llamado interrupción.

En la tarjeta utilizamos el controlador programable de interrupciones 8259 que puede operar y priorizar hasta 8 interrupciones solicitadas al procesador. Internamente esta compuesto de dos registros: el registro de solicitudes de interrupción (IRR) sobre el que se determina la prioridad de las interrupciones y el registro de atención de interrupciones (ISR) que informa al procesador de las interrupciones que están siendo atendidas

Este integrado opera de la siguiente manera: recibe la solicitud de interrupción de uno de los dispositivos activándose su bit del IRR, lo analiza y envia la señal de INTR al procesador, éste, a su vez, al reconocerlo responde con un pulso de INTA^ activando el bit que le corresponde en el registro ISR del

dispositivos preguntándoles si alguno requiere ser atendido. Es fácil observar que al aumentar el número de dispositivos conectados a la tarjeta aumenta el tiempo que el procesador debe dedicar a barrer todos los dispositivos, disminuyendo, en la misma proporción, la velocidad de ejecución del programa principal.

Un método más deseable es aquel que permite que el procesador esté ejecutando su programa principal y que sólo lo suspenda para ejecutar la rutina de servicio de un dispositivo cuando el mismo dispositivo así lo solicita, volviendo posteriormente, a la rutina principal en el punto donde suspendió su ejecución. De esta manera, el número de dispositivos a atender puede aumentar sin afectar el tiempo de respuesta del sistema y sólo requiere que a todos los dispositivos se les asigne un orden con el cual serán atendidos por el procesador. Este método es llamado interrupción.

En la tarjeta utilizamos el controlador programable de interrupciones 8259 que puede operar y priorizar hasta 8 interrupciones solicitadas al procesador. Internamente esta compuesto de dos registros: el registro de solicitudes de interrupción (IRR) sobre el que se determina la prioridad de las interrupciones y el registro de atención de interrupciones (ISR) que informa al procesador de las interrupciones que están siendo atendidas

Este integrado opera de la siguiente manera: recibe la solicitud de interrupción de uno de los dispositivos activándose su bit del IRR, lo analiza y envia la señal de INTR al procesador, éste, a su vez, al reconocerlo responde con un pulso de INTA^ activando el bit que le corresponde en el registro ISR del

controlador de interrupciones y borrando el del IRR. Al mismo tiempo el 8259 determina la interrupción con la más alta prioridad.

Cuando el procesador genera el segundo pulso INTA^, el 8259 libera una palabra de 8 bit que contiene la dirección de inicio de la rutina de servicio al bus de datos de donde lo lee el procesador. Esto completa el ciclo de interrupción entre el procesador y el controlador de interrupciones, que borra el bit del registro ISR de la interrupción atendida. El procesador, por su parte, guarda la posición del programa que estaba ejecutando e inicia la ejecución del programa de la interrupción solicitada.

Para programar este integrado se debe habilitar la señal de CS^ y enviarle de dos a cuatro bytes con los comandos de inicialización (ICW).

El comando de inicialización uno (ICW1) le informa al controlador la forma de operar al cambio de las señales de solicitud de interrupción (nivel o transición), si operará en cascada con otros 8259's y si requiere identificar al procesador que lo controla. En nuestro caso, operará al cambiar el nivel de la señal de solicitud de interrupción (y no por transición), no se le enviará la palabra ICW3 por no operar en cascada con otros 8259 y que debe esperar el comando ICW4 para identificar al procesador como 8088.

El comando de inicialización dos (ICW2), le indica al controlador la dirección de inicio de las rutinas de servicio de cada interrupción. En cada palabra los 5 bits más significativos son las dirección (A15-A11) y los tres bits menos significativos identifican a la interrupción.

El cuarto comando (ICW4), le informa al controlador que el procesador del sistema es un 8088, que active la terminación automática de interrupciones (AEOI) en vez de esperar que la rutina de servicio genere la señal de interrupción terminada (EOI).

El controlador permite a través del uso de las palabras de control de operación (OCW) que se inhiban determinadas interrupciones, en nuestro caso habilitamos la interrupción 6 que permite que el controlador del "driver" interrumpa al procesador y la interrupción 7 que se efectua cuando el puerto serie recibe la secuencia de cambio de interfaz con el usuario.

## Interfaz programable Serie 8251

En el diseño del sistema se incluyó un interfaz programable para la transmisión en serie, ya que resulta la manera más sencilla de conectarlo con una computadora. Normalmente se utiliza la transmisión en serie pues requiere un menor número de cables, permite transmisiones a grandes velocidades con menos errores debidos al ruido.

El 8251 es un receptor/transmisor con operación síncrona o asíncrona (USART) que acepta los datos en paralelo que le envía el procesador y los convierte en una secuencia de datos para su transmisión en serie o viceversa, recibir en serie para transformarlos a paralelo. Este circuito le avisa al procesador cuándo puede recibir otro dato en paralelo para su transformación y transmisión en serie o cuándo ya convirtió un dato recibido en serie a paralelo.

Puede operar en dos modos: en el modo síncrono, donde una vez iniciada la comunicación continuamente existe intercambio de información protocolaria entre el emisor y el receptor en bloques que se envían como una unidad a una frecuencia de transmisión dada. Cabe aclarar que en el caso de que no haya dato disponible a transmitirse, el 8251 automáticamente genera y transmite caracteres de sincronia.

En el modo asíncrono, se envían unidades de hasta 10 bits (bit de inicio, bits de datos, bit de paridad y bit de fin), únicamente cuando existe información a ser transmitida. La ventaja de este modo de transmisión es que sólo se distrae al procesador para la recepción de información y no para procedimientos de sincronización.

Para simplificar la conexión entre el controlador y la PC, se decidió trabajar con el modo de transmisión asíncrono.

Antes de inciar la recepción o transmisión de información, el 8251 debe ser programado por dos palabras de control generadas por el procesador. Estas dos palabras definen completamente el modo de operación. Las palabras de control están divididas en dos formatos: la de instrucción y la de comando.

El formato de instruccion define las caracterísitcas generales de operación, es decir, si opera en modo síncrono o asincrono, si generará y verificará bits de paridad, la longitud de la palabra y sincronía externa para el modo síncrono y el número de bits de stop para el modo asíncrono.

En el formato de comando se controla la operación del modo de comunicación seleccionado, a través de las funciones para habilitar la transmisión/recepción, el reset de errores y el control del modem.

Para que este integrado pueda interrumpir al procesador a través del controlador de interrupciones 8259 es necesario conectar la señal de RxD (Receiver Ready) al pin de la interrupción 7 del 8259.

### Generador de Baud Rate MC14411

Para generar la velocidad de transmisión serial se utiliza un Generador de Baud Rate MC 14411. Este integrado es un generador de bases de tiempo programable. Tiene 16 salidas que nos proporciona una determinada frecuencia dependiendo de la selección del rango deseado. Este circuito tiene cuatro rangos a escoger dependiendo del factor a multiplicar la frecuencia del cristal (1.8432MHz) y hasta 35 diferentes frecuencias. En nuestro caso se optó por tomar el primer rango de frecuencias por ser estas las normalmente utilizadas.

### Interfaz programable Paralelo 8255

Las transmisiones en paralelo son útiles en todas aquellas aplicaciones que requieren una gran velocidad y utilicen dispositivos no demasiado alejados entre sí (requiere de un mayor número de líneas para la transmisión de información).

No hay ninguna sincronización especial en las transmisiones en paralelo. La velocidad máxima de transmisión está limitada por la rapidez con que el sistema pueda operar los datos.

Este integrado consta de tres puertos paralelos (A, B y C), cada uno de 8 bits. El puerto C se puede dividir en dos puertos de 4 bits cada uno.

Existen tres modos de operación que pueden ser seleccionados por software para definir el funcionamiento de los puertos: Modo 0 (Basic Input/Output), Modo 1 (Strobed Input/Output) y Modo 2 (Bi-Directional Bus).

En el Modo 0, los puertos A, B y C pueden funcionar como entradas o salidas donde las salidas son "latched" y las entradas no. Como el puerto C puede ser dividido en dos puertos de cuatro bits, este modo de operación genera 16 posibles combinaciones.

En el Modo 1, se tienen dos grupos de operación 1 o 2. Cada uno de ellos contiene un puerto de 8 bits de datos (A ó B) que puede ser programado como entrada o salida donde ambas son "latched" y otro de 4 (la mitad de C) para la información de control y estatus en forma bidireccional.

El modo 2 se establece al puerto A como un bus bidireccional de 8 bits y ocupa 5 líneas del puerto C para control del mismo.

En el diseño se utiliza el modo de operación 0, donde A y C son puertos de salida y B es un puerto de entrada.

### Reloj de tiempo real RTC 58321

El reloj de tiempo real proporciona la fecha y la hora para la grabación de los archivos de información proveniente de los dispositivos de medición.

Este reloj tiene la ventaja de contar con un oscilador integrado, por lo que no requiere un cristal externo para su base de tiempo.

Entre sus características primordiales está el poder multiplexar su bus de direcciones con el de datos.

Durante su programación se le habilita como memoria para leer o escribir un dato que puede ser la fecha o la hora y como dispositivo de entrada/salida para identificar el dato a utilizar.

Es posible programar segundos, minutos, horas, dia, semana, mes y año (lleva el control de años bisiestos).

## III.3) DIAGRAMAS DEL CONTROLADOR

### III.3.1) Arquitectura básica con 8088 y DMA

En la figura III.2 se muestra el diagrama de conexión entre el procesador 8088, el generador de pulsos de reloj 8284 y el DMA 8257.

Como ya se mencionó, el 8088 opera en modo mínimo, es decir, autogenera las señales de control del bus de datos y direcciones por lo que el pin 33 es conectado a Vcc.

Existen dos señales para el manejo de interrupciones mascarables, Interrup Request (INTR) e Interrup Acknowledge (INTA). Ambas se conectan al Controlador de Interrupciones (8259) como se muestra en la figura III.5. El diseño de esta tarjeta no contempla el uso de interrupciones no mascarables, por lo que el pin 17 (NMI) está conectado a tierra para deshabilitar su función.

La señal de TEST que normalmente es utilizada en conjunto con la instrucción de WAIT, para detener temporalmente la ejecución de un programa no será utilizada en este diseño, por lo cual se conecta a tierra.

Las señales que recibe del generador de pulsos 8284 son CLK, RESET y READY.

El generador de pulsos de reloj 8284 divide en tres la frecuencia de oscilación tanto de un cristal conectado entre sus pines 16 y 17 o de una señal que se reciba por el pin 14. En este caso, se decidió utilizar un circuito oscilador por lo que la señal Frequency/Cristal select (F/C^) está conectada a Vcc. Las resistencia R1 y R2 se utilizan para darle estabilidad al circuito de oscilación.

La señales para sincronizar al procesador con otros dispositivos más lentos (AEN1^, AEN2^, RDY1, RDY2, ASYNC^, ) no son utilizadas al contar con circuitos integrados de rápido acceso. Las señales de External Frequency (EFI) y de Clock

Synchronization (CSYNC) tampoco son utilizadas. Para deshabilitarlas estas señales, se conectan como se muestra en la figura III.2

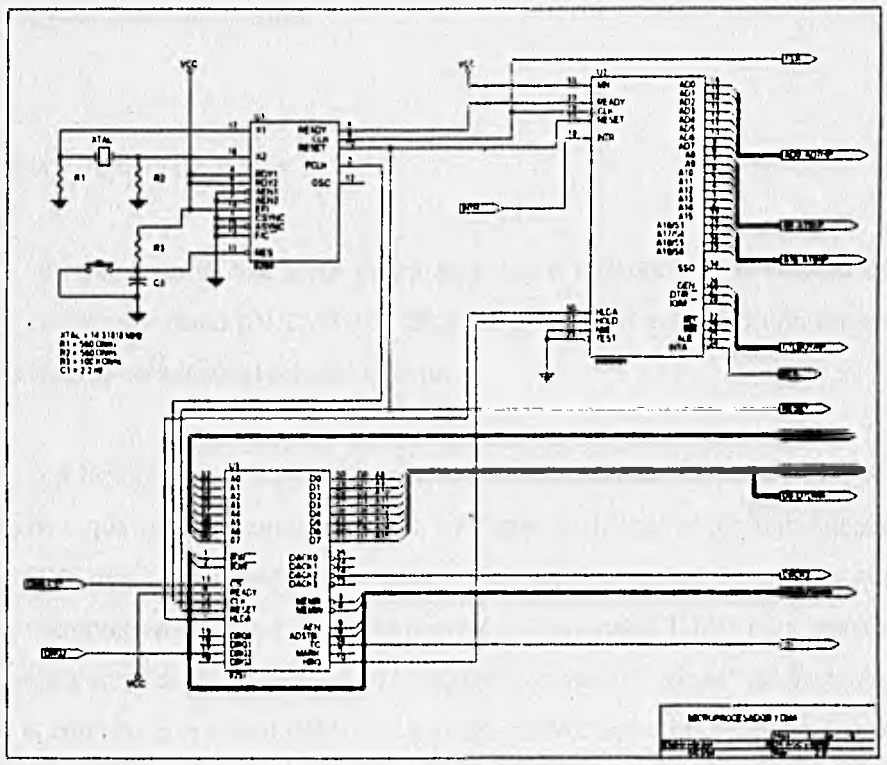

**Figura III.2**

El 8284 genera la señal de RESET para el sistema a través de un circuito RC conectado a un "Schmitt Trigger" interno del 8284 en el pin RES^ .

Esta señal junto con la de PCLK son utilizadas por el DMA 8257 (Direct Memory Access).

El DMA al recibir el DRQ2 proveniente del controlador de disco (figura III.5), solicita al procesador el control del bus del sistema. Una vez concedido, el DMA le avisa al controlador del "driver" con DACK2 que puede iniciar la transferencia de información.

### III.3.2) Control de Buses

Este circuito se diseñó para garantizar que el multiplexado de señales tanto del procesador como del DMA sea eficiente y asegurar que sólo uno de estos dispositivos controlará el bus del sistema.

La lógica externa permite separar las direcciones de los datos. En ella se observa que para el control del bus de datos se utiliza un circuito integrado 74LS245 que permite manejar los datos en forma bidireccional. Se utiliza la señal del microprocesador que controla la dirección de los datos DT/R^ para identificar si es entrada o salida y DEN^ como pulso para activar la "captura" del dato válido. En el caso de que sea el DMA el que tenga control sobre el bus no se requiere circuitería especial para manejar los datos.

Se utilizaron tres circuitos integrados 74LS373 y la combinación de las señales de control ALE del procesador y AEN y ADSTB del DMA para habilitar adecuadamente a estos circuitos e inhibir la posible contención de buses.

En el caso de que sea el microprocesador el que genera la dirección, los bits menos significativos (A0-A7) son capturados en el primer "latch" utilizando la

señal del procesador ALE y AEN del DMA. Los siguientes ocho bits (A8-A15) como no se multiplexan no se tienen que almacenar, y por último, la información de los bits más significativos (A16-A19) vuelven a capturarse usando las mismas señales que para el primer 74LS373.

Si es el DMA el que tiene control sobre el bus de direcciones, los bits menos significativos (A0-A7) que no se multiplexan, no se "latchean". Los siguientes ocho bits (A8-A15) son capturados en el segundo "latch" utilizando las señales del DMA AEN y ADSTB. El DMA no genera señales para accesar más de 64 kbytes, por lo que no requiere del tercer "latch".

El último 74LS373 es utilizado como "page register" para permitir que el DMA pueda accesar el mismo mapa de memoria que accesa el procesador. Como el DMA sólo tiene 16 bits de direcciones, es necesario proveer de manera externa los 4 bits más significativos. El programador debe asegurar que los datos del "page register" correspondan a la sección de memoria que se desea utilizar.

Para generar el pulso que guarde la información de los 4 bits en el "page register", se utiliza una operación lógica NOR de las señales "chip select" del "page register" (P.R.CS^) con WR^ y DEN^. que vienen directamente del control de buses del procesador.

En la figura III.3 se muestra el diagrama de conexiones del control de buses.

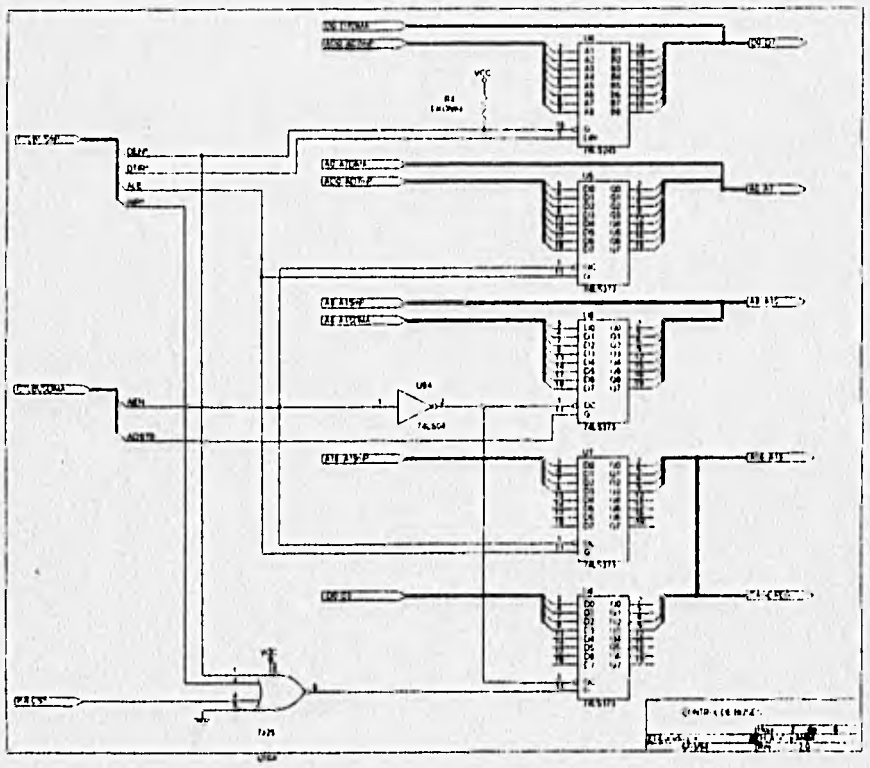

**Figura III.3**

### III.3.2) Líneas de Control y Decodificación de Memorias y Puertos.

Como ya se mencionó, las señales del bus de control son generadas por el procesador o por el DMA. Las señales aunque equivalentes no son iguales, el procesador genera las señales de IO/M^, RD^ y WR^ mientras que el DMA genera I/OR^, I/OW^, MEMR^ y MEMW^.

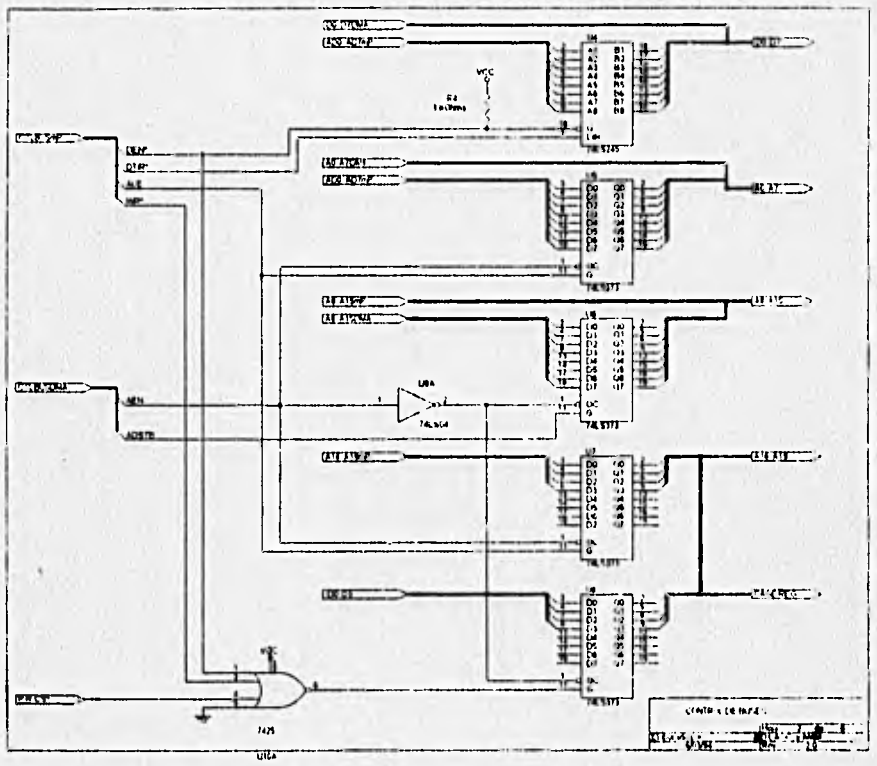

**Figura III.3**

## III.3.2) Líneas de Control y Decodificación de Memorias y Puertos.

Como ya se mencionó, las señales del bus de control son generadas por el procesador o por el DMA. Las señales aunque equivalentes no son iguales, el procesador genera las señales de IO/M^, RD^ y WR^ mientras que el DMA genera I/OR^, I/OW^, MEMR^ y MEMW^.

Para facilitar el control del sistema, las tres señales que genera el procesador son reacondicionadas utilizando el integrado 74LS257 para ser iguales a las del DMA. En la figura III.4 se muestra el diagrama de la conexión de este integrado.

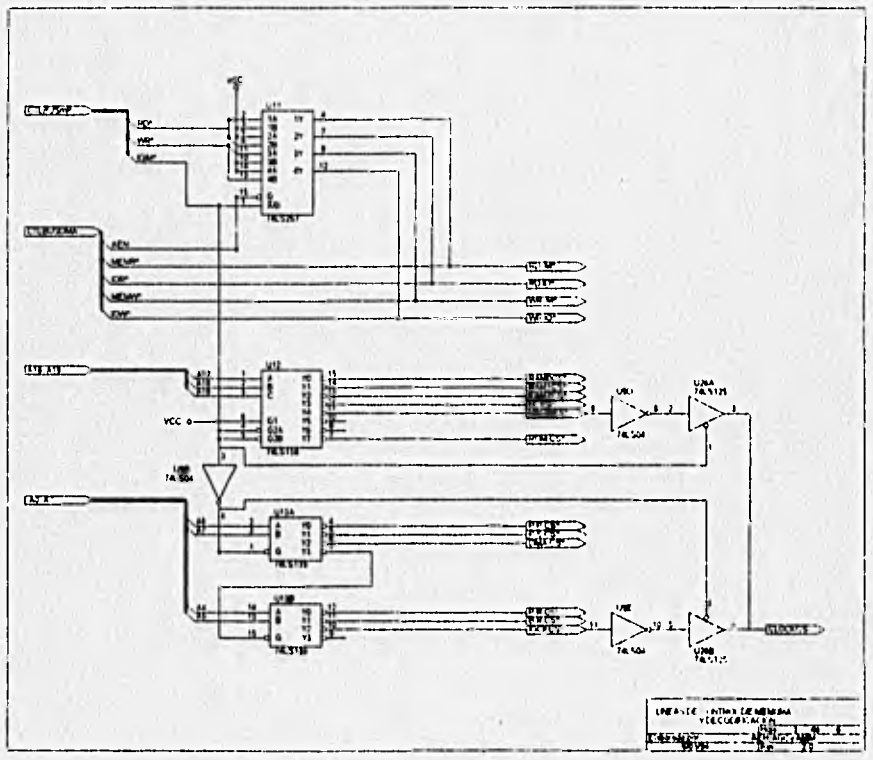

**Figura III.4**

En la misma figura se observa el diagrama de decodificadores para las memorias así como para los dispositivos de entrada y salida. El objetivo de esta decodificación es asegurar que no se activen dos dispositivos simultáneamente.

La decodificación de las memorias está determinada por una característica del procesador, que al encender el sistema o después de un RESET éste apunta a la dirección FFFF0H. En esta dirección debe encontrarse siempre un salto a otra dirección donde realmente se encuentre el programa de inicialización y control de la tarjeta. Este salto y el programa de inicialización y control se encuentran pregrabados en la EEPROM 27512.

Partiendo de esta condición, el mapa de memoria se definió como sigue:

| DIRECCIONES | | DISPOSITIVOS | SEÑAL |
|---|---|---|---|
| [illegible] | [illegible] | EPROM 27512 de 64K x 8 | ROM CS^ |
| [illegible] | [illegible] | | Libre |
| [illegible] | [illegible] | | Libre |
| [illegible] | [illegible] | Reloj de Tiempo Real | MECKCS^ |
| [illegible] | [illegible] | Controlador de Interrupciones | I. C. CS^ |
| [illegible] | [illegible] | RAM2 HM62256 de 32K x 8 | RAM2CS^ |
| [illegible] | [illegible] | RAM1 HM62256 de 32K x 8 | RAM1CS^ |
| [illegible] | [illegible] | RAM0 2016 de 8K x 8 | RAM0CS^ |

Se utiliza el circuito integrado 74LS138 que decodifica las señales A17, A18, A19 e IO/M^ emitidas por el procesador para habilitar a uno de estos dispositivos.

El mapa de los dispositivos de entrada y salida es controlado por el circuito integrado 74LS139 de la siguiente manera:

| DIRECCIONES | | DISPOSITIVOS | SEÑAL |
|---|---|---|---|
| 00H | 3FH | Puerto Paralelo 8255 | P. P. CS^ |
| 40H | 7FH | Puerto Serie 8251 | P. S. CS^ |
| 80H | BFH | DMA 8257 | DMACS^ |
| C0H | CFH | Page Register | P. R. CS^ |
| D0H | DFH | Generador de Baud Rate | B. R. CS^ |
| E0H | EFH | Reloj de Tiempo Real | IOCK CS^ |
| F0H | FFH | Libre | |

La selección de las alternativas se realiza con las señales del bus de direcciones A4-A7 y con la señal negada de IO/M^ pues el integrado se habilita en bajo.

Para controlar el reloj de tiempo real, se utiliza las señales MECKCS^ del decodificador de memorias así como IOCKCS^ del decodificador de entrada y salida y lógica externa. Lo anterior es necesario pues el reloj para su programación y uso requiere que se le habilite como memoria para leer o escribir un dato, y como dispositivo de entrada y salida para identificar el dato a utilizar. Con las dos señales se genera la de CLOCKCS.

### III.3.4) Memorias, controlador de interrupciones e interfaz para el controlador del "driver".

El diagrama de conexión de las memorias se muestra en la figura III.5. Como se observa, las señales ROMCS^, RAM0CS^, RAM1CS^ y RAM2CS^ generadas por el integrado 74LS138 habilitan a las memorias. Las señales del bus de control que se utilizan al operar con las memorias RAM0, RAM1 y RAM2 son: WRM^ y RDM^.

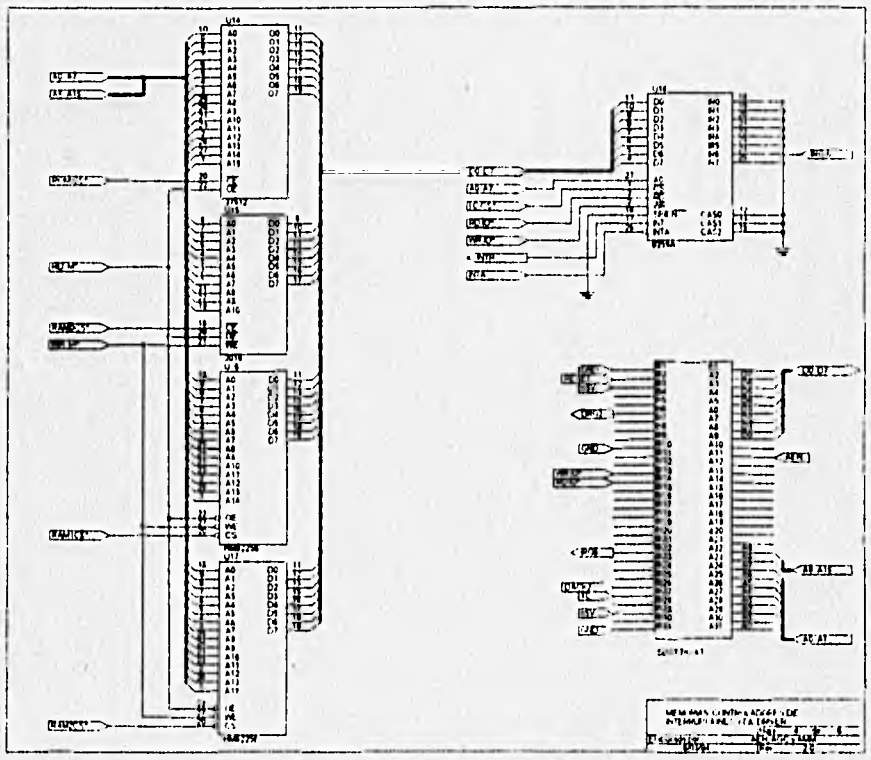

**Figura III.5**

La EEPROM 27512 sólo utiliza RDM^. En ella reside el programa de inicialización y de control de todo el sistema

El controlador de interrupciones 8259 considerado como un dispositivo de entrada/salida para el procesador, recibe la solicitud de interrupción del controlador del "driver" a través de la línea IRQ6 y del puerto serie por la IRQ7.

La señal "Slave Program Enable Buffer", está conectada a tierra ya que no hay otro controlador de interrupciones en la tarjeta.

También se muestra en la figura III.5 las señales de interfaz con el controlador del "driver".

### III.3.6) Puerto Serie 8251

La señal de CLK es usada para generar el tiempo interno de operación del puerto serie. Esta señal es tomada del PCLK del generador de pulsos 8284.

La programación del 8251 se realiza a través de la señal "Control/Data" (C/D^) en conjunto con la señal de WR IO^ y el P.S. CS^. Si C/D^ es igual a 1, el puerto serie espera ser programado, si es 0 el puerto serie está en modo operación.

El control de la señal C/D^ se realiza a través del bit menos significativo del bus de direcciones.

El 8251 a través de sus señales TxC y RxC, recibe la frecuencia de transmisión y recepción de datos, ambas señales deben ser de la misma frecuencia y estar en sincronía.

La señal de transmisión TxD se conecta con el circuito integrado 1488 que eleva el voltaje de la señal de 0 y +5 volts a una de +/- 12 volts para cumplir con el estándar RS232C.

Por otra parte, el circuito integrado 1489 se utiliza para la recepción de datos, transforma las señales de +/- 12 volts en voltajes TTL (0 y 5volts), que son enviados al puerto serie a través del pin RxD.

Para generar las frecuencias de transmisión y recepción, se utiliza un generador de Baud Rate que proporciona 16 frecuencias por rango. Para este diseño se optó por tomar el primer rango de frecuencias por lo que se utiliza un factor de 1 a través de conectar a tierra las líneas RSA y RSB.

Dentro de las frecuencias generadas en este rango, se seleccionaron 9600 bps (F1), 4800 bps (F3), 2400 bps (F5) y 1200 bps (F7). Para poder seleccionar por programa una de estas cuatro frecuencias utilizamos el 7475 y el 74153.

El 7475, hace las veces de un Flip Flop seguidor que guarda la información de los bits D0 y D1 con el pulso generado por la señal de B.R. CS^ y WR I/O^. Este valor almacenado lo recibe el 74153 por lo que deja pasar sólo el canal de frecuencia seleccionada.

En la siguiente figura se muestra el diagrama de conexión del puerto serie.

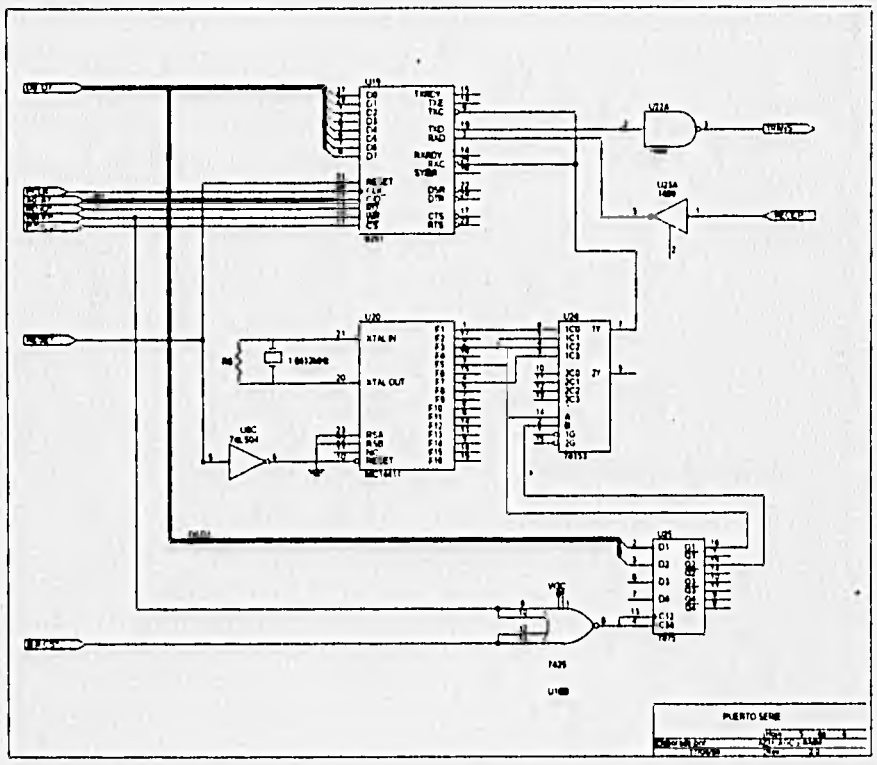


**Figura III.6**

### III.3.7) Puerto Paralelo y Reloj de Tiempo Real

El puerto paralelo en la tarjeta realiza básicamente dos operaciones. La primera como elemento de enlace con la interfaz con el usuario donde lee y despliega la selección de la operación a realizar.

La segunda como elemento para controlar el sensor o estación de muestreo así como recibir la información enviada por los mismos

Adicionalmente, este dispositivo puede ser utilizado para detectar y corregir problemas en la tarjeta controladora..

Por otra parte, el reloj de tiempo real es el dispositivo utilizado para generar la fecha y hora que utiliza MS-DOS al crear o actualizar un archivo.

El integrado se habilita con alto en las señales CS1 y CS2. En el diagrama de conexión se ve que CS1 está conectado a Vcc y que CS2 responde a la señal de CLOCKCS. Cabe aclarar que esta señal habilita al integrado para operar tanto como memoria como dispositivo de entrada y salida por lo que acepta los comandos de OUT y movE.

Las señal de TEST se conecta a tierra para evitar un mal funcionamiento debido a ruido y la de STOP a CLOCKCS para detener el funcionamiento del reloj durante las lecturas o escrituras.

Para la programación de este integrado se manda por el bus de datos el identificador de unidades de segundo como si fuera un dato a escribir en un puerto (comando OUT) y después se manda por ese mismo bus el dato con la instrucción de movE. De manera similar se programan el resto de los registros.

En la figura III.7 se observa el diagrama de conexiones tanto del puerto paralelo como del reloj de tiempo real.

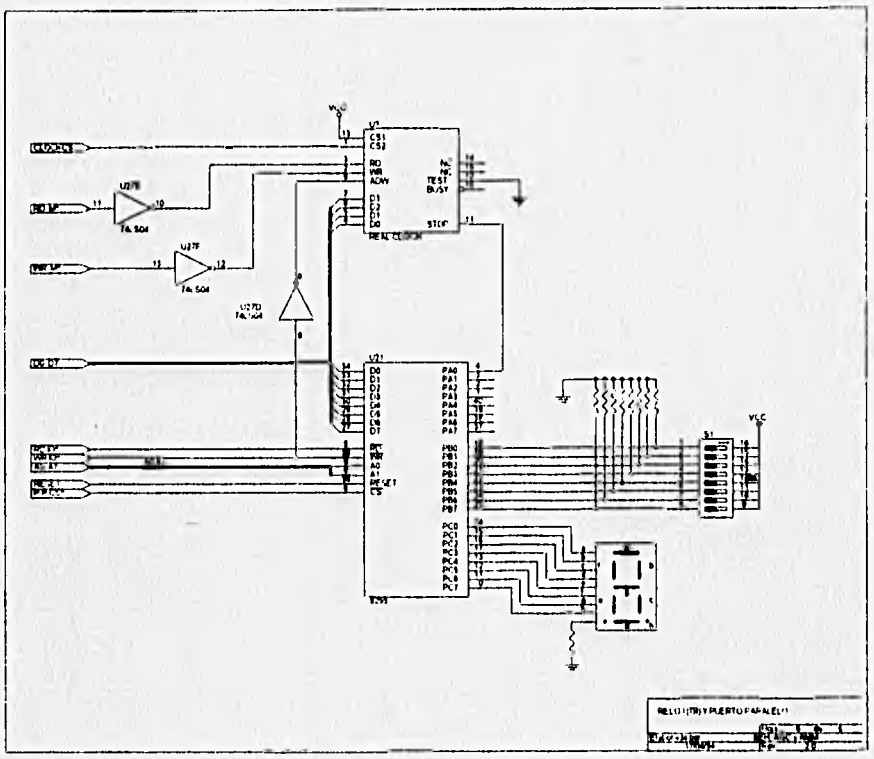

**Figura III.7**

# CAPITULO IV

# PROGRAMACION DEL CONTROLADOR

## IV.1) Diagrama General de Flujo

La programación se divide en cuatro módulos, el primero se encarga de inicializar los componentes de la tarjeta controladora, el segundo controla el manejador de disco, el tercero es la interfaz con el usuario final de la tarjeta y por último el módulo de programación para el sensor o estación de muestreo.

La secuencia en que se tienen que ejecutar estos módulos en la tarjeta se muestra en la figura IV.1. Los tres primeros son los programas base y se encuentran almacenados en la ROM de la tarjeta mientras que el último módulo es desarrollado por el programador para el manejo y control del sensor o estación de muestreo y no reside en la ROM de la tarjeta.

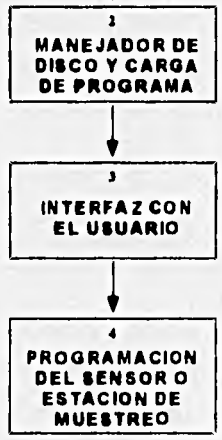


**Figura IV.1**

## IV.2) Rutina de Inicialización

Este módulo del programa se ejecuta cada vez que se enciende o se "resetea" la tarjeta controladora y su función principal es la de programar los elementos de la tarjeta y verificar su correcto funcionamiento. En la figura IV.2 se muestra la secuencia en la que cada uno de los dispositivos es programado y cómo se realiza su verificación.

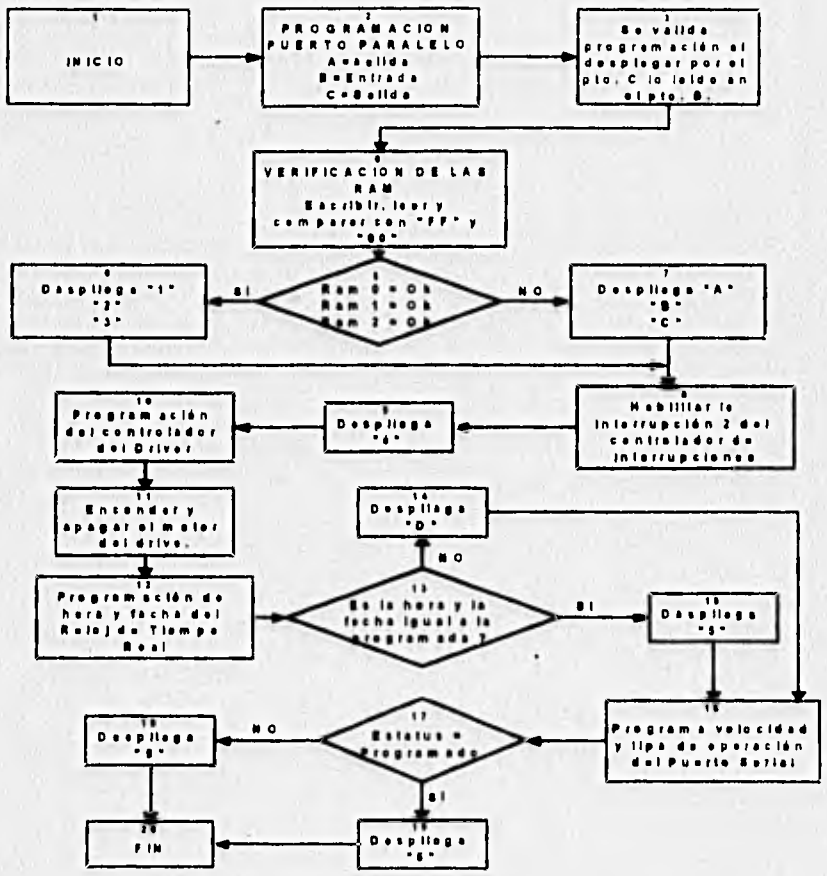


**Figura IV.2**

## IV.3) Manejador de Disco y Carga de Programa

Como ya se ha mencionado, las características principales de este diseño es poder accesar datos de disco en formato de MS-DOS, manejando las áreas de Fat y Dir de una manera equivalente a como lo hace MS-DOS. Esta situación y el diseño conceptual de la tarjeta permite al usuario final hacer un programa "Ad Hoc" para sensores y estaciones de muestreo, compilarlo y guardarlo en disco y posteriormente leerlo en la tarjeta controladora para ser ejecutado sin necesidad de escribir en la ROM de la tarjeta controladora.

Fundamentalmente, se elaboraron tres rutinas independientes para proveer al programador del sensor o estación de muestreo y al usuario final de comandos de software para poder realizar los accesos a disco y la ejecución de los programas de control y muestreo. Estos tres comandos o interrupciones son: Dir para desplegar el nombre de los archivos que se encuentran en el disco; Lee y ejecuta para leer un archivo del disco, colocarlo en memoria de la tarjeta y transferir el control de la tarjeta a dicho programa; y Escribe para escribir datos de memoria de la tarjeta a disco.

### IV.3.1) RUTINA DE DESPLIEGUE DE DIRECTORIO

Esta rutina consiste básicamente en leer la zona de directorio del disco y desplegar los datos de cada archivo localizados en el disco. Esto le da la oportunidad al usuario de medir la capacidad de almacenamiento que tiene, verificar que los archivos de datos se hayan grabado en su totalidad, etc.

A continuación, en la figura IV.3 se muestra el diagrama general de flujo de la rutina para desplegar el directorio del disco.

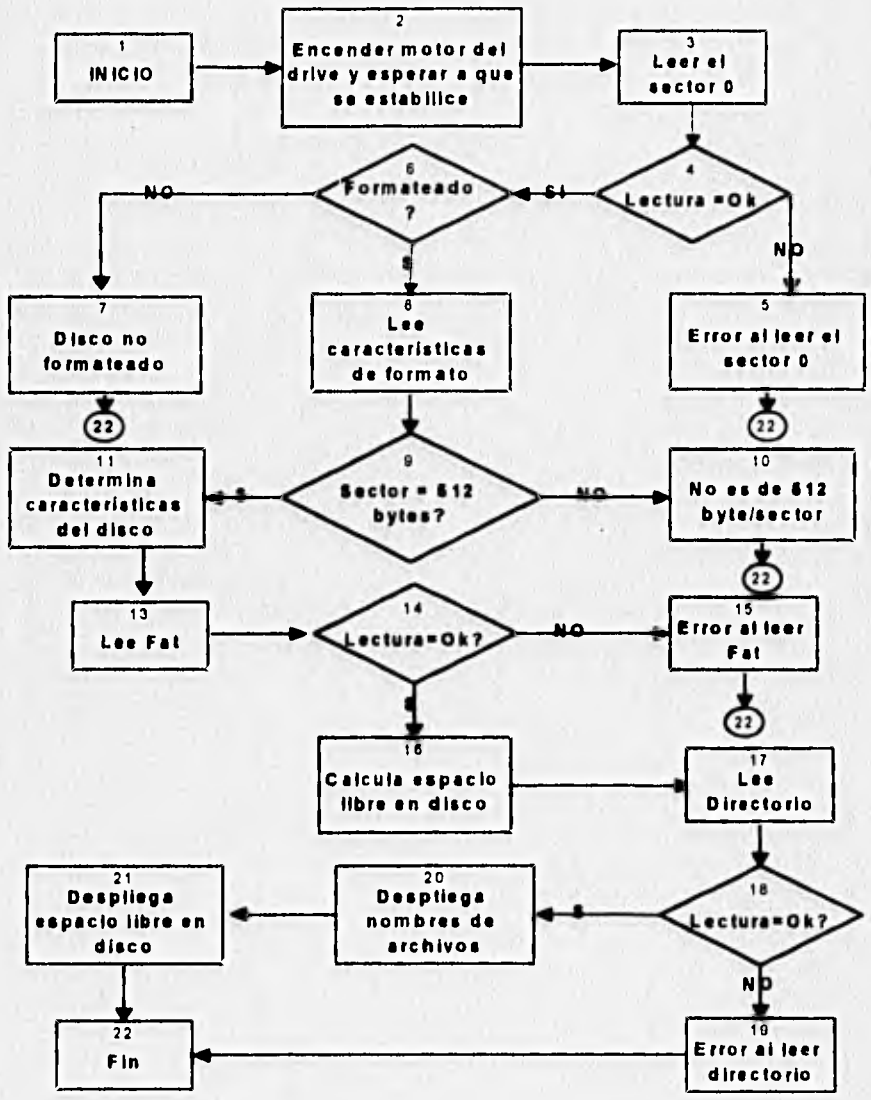


**Figura IV.3**

### IV.3.2) RUTINA PARA LEER Y EJECUTAR UN ARCHIVO

Esta rutina que es descrita en la figura IV.4, permite leer el archivo especificado del disco, cargarlo en ciertas localidades de memoria y transferirle el control de ejecución a dicho archivo. La función principal de esta rutina es proveer al usuario de la tarjeta un elemento para poder cargar el programa de manejo y control de sensores o estaciones de adquisición sin tener que grabar la ROM.

### IV.3.3) RUTINA PARA ESCRIBIR A DISCO

El propósito de esta rutina es permitir el almacenamiento de los datos recolectados por el sensor o estación de muestreo en un medio magnético de alta capacidad (disco flexible) en formato compatible con MS-DOS para permitir su procesamiento posterior en paquetes estándares de mercado. (Excel, Lotus, etc.).

Esto se muestra en la figura IV.5

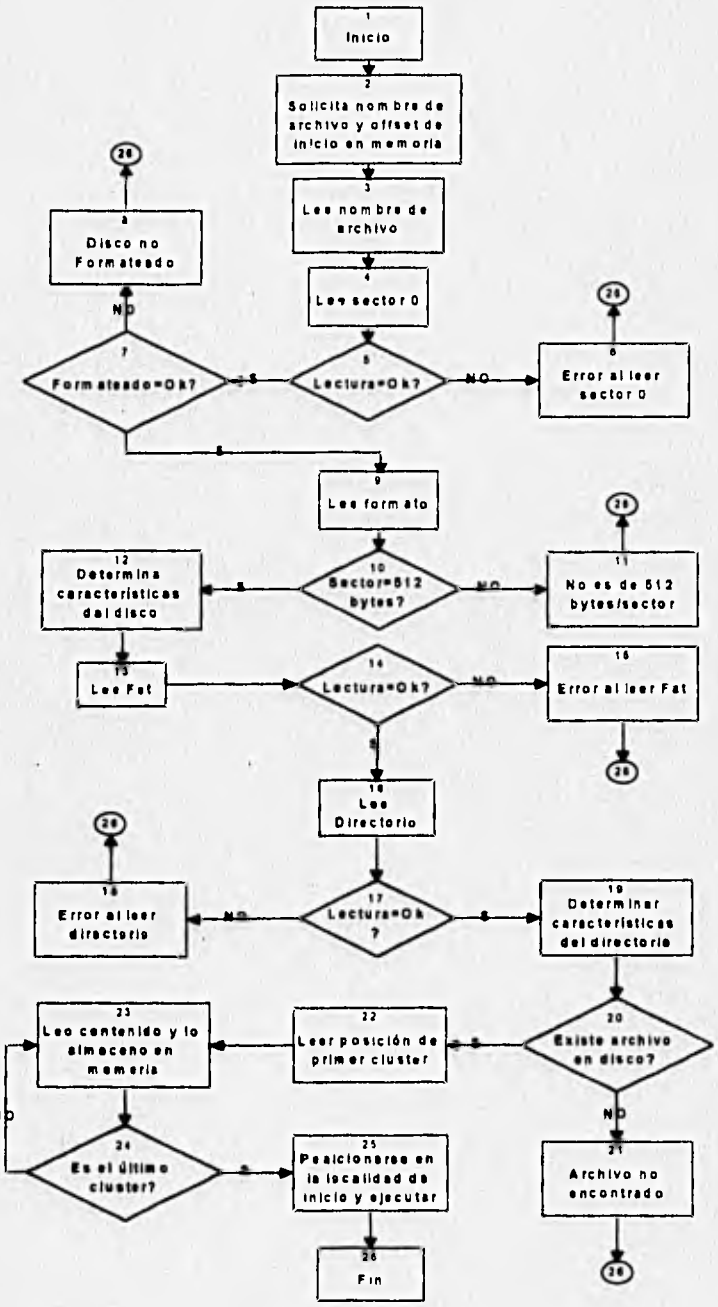
1
Inicio
2
Solicita nombre de archivo y offset de inicio en memoria
3
Lee nombre de archivo
4
Lee sector 0
5
Lectura=Ok?
SI
NO
6
Error al leer sector 0
7
Formateado=Ok?
NO
8
Disco no Formateado
9
Lee formato
10
Sector=512 bytes?
SI
NO
11
No es de 512 bytes/sector
12
Determina caracteristicas del disco
13
Lee Fat
14
Lectura=Ok?
NO
15
Error al leer Fat
16
Lee Directorio
17
Lectura=Ok ?
NO
18
Error al leer directorio
19
Determinar caracteristicas del directorio
20
Existe archivo en disco?
SI
NO
21
Archivo no encontrado
22
Leer posición de primer cluster
23
Leo contenido y lo almaceno en memoria
24
Es el último cluster?
NO
25
Posicionarse en la localidad de inicio y ejecutar
26
Fin

**Figura IV.4**

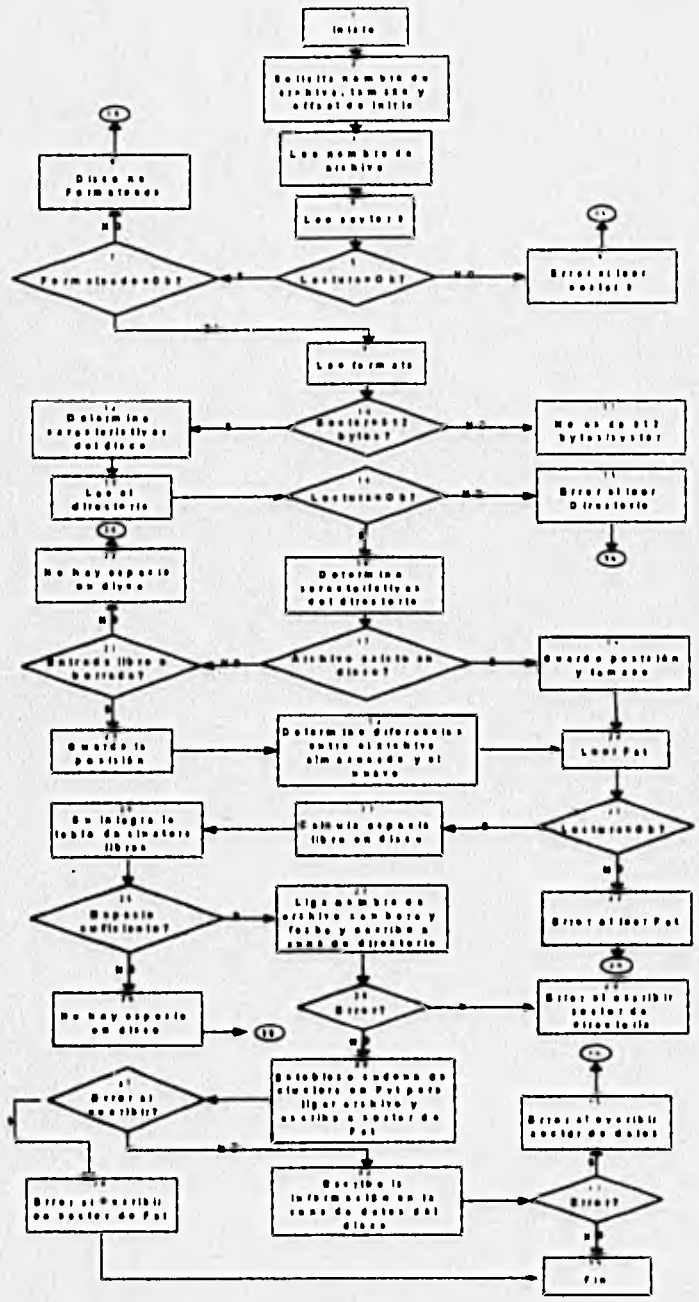

**Figura IV.5**

## IV.4) Interfaz con el Usuario

Se desarrollaron dos versiones de esta interfaz. Una en lenguaje ensamblador para residir en la memoria EEPROM para cuando se utilice el "display" y teclado de la tarjeta controladora y otra en BASIC, para ejecutarlo en la P.C. conectada a través del puerto serie que sustituye al "display" y teclado de la tarjeta.

La diferencia fundamental entre estas dos versiones reside en el puerto que se utiliza durante la interfaz. Si la tarjeta opera autónomamente, las instrucciones del usuario las recibe a través del puerto paralelo. Si la tarjeta utiliza el teclado y "display" de una PC la comunicación se realiza por medio del puerto serie.

Para no entrar en conflicto, sólo una de estas alternativas puede estar habilitada. La del puerto paralelo reside en memoria EEPROM como parte de la configuración inicial del sistema que puede ser deshabilitada al transmitírsele a través del puerto serie la secuencia de cambio.

Ambos programas permiten que el usuario seleccione una de las siguientes opciones: Dir, Escribe, Lee y Ejecuta y Programación del Reloj. En la figura IV.6 se muestra el diagrama de bloques de la interfaz paralelo y en la IV.7 la interfaz serial.

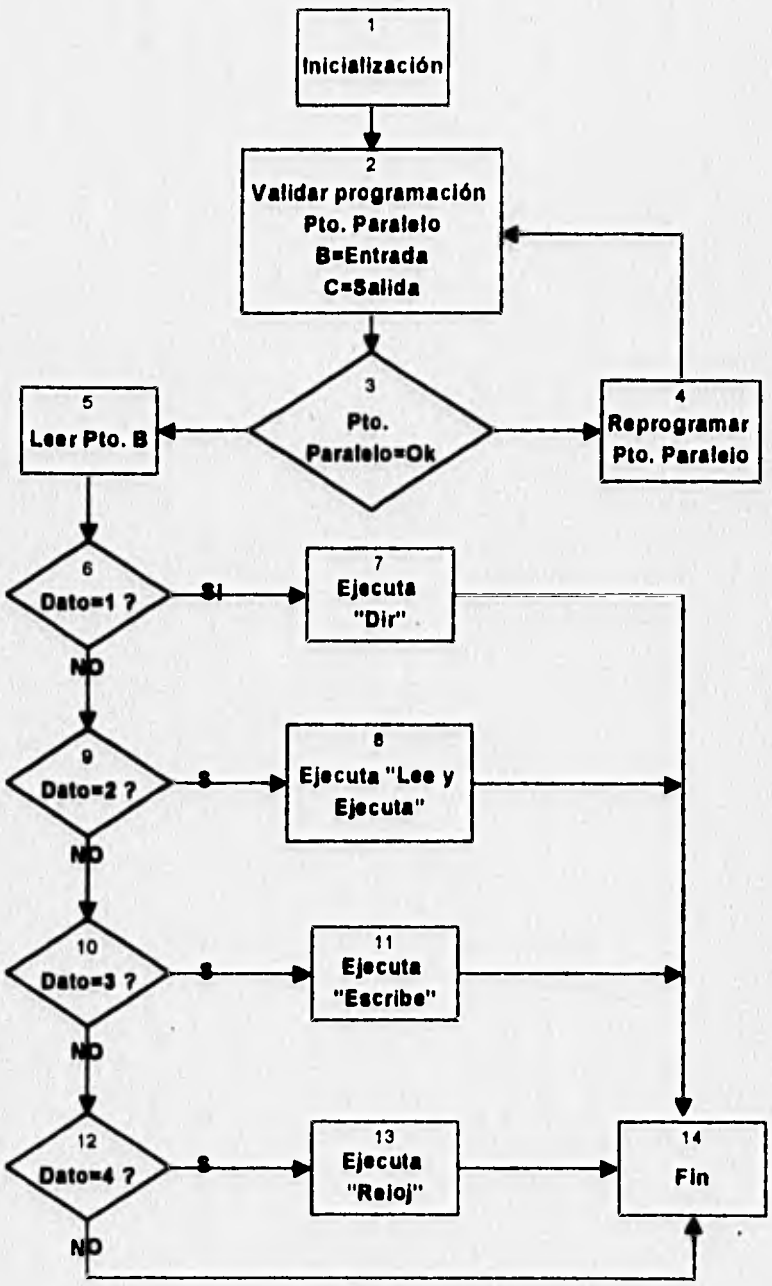
1
Inicialización
2
Validar programación
Pto. Paralelo
B=Entrada
C=Salida
3
Pto.
Paralelo=Ok
4
Reprogramar
Pto. Paralelo
5
Leer Pto. B
6
Dato=1 ?
SI
7
Ejecuta
"Dir"
NO
9
Dato=2 ?
S
8
Ejecuta "Lee y
Ejecuta"
NO
10
Dato=3 ?
S
11
Ejecuta
"Escribe"
NO
12
Dato=4 ?
S
13
Ejecuta
"Reloj"
14
Fin
NO

**Figura IV.6**

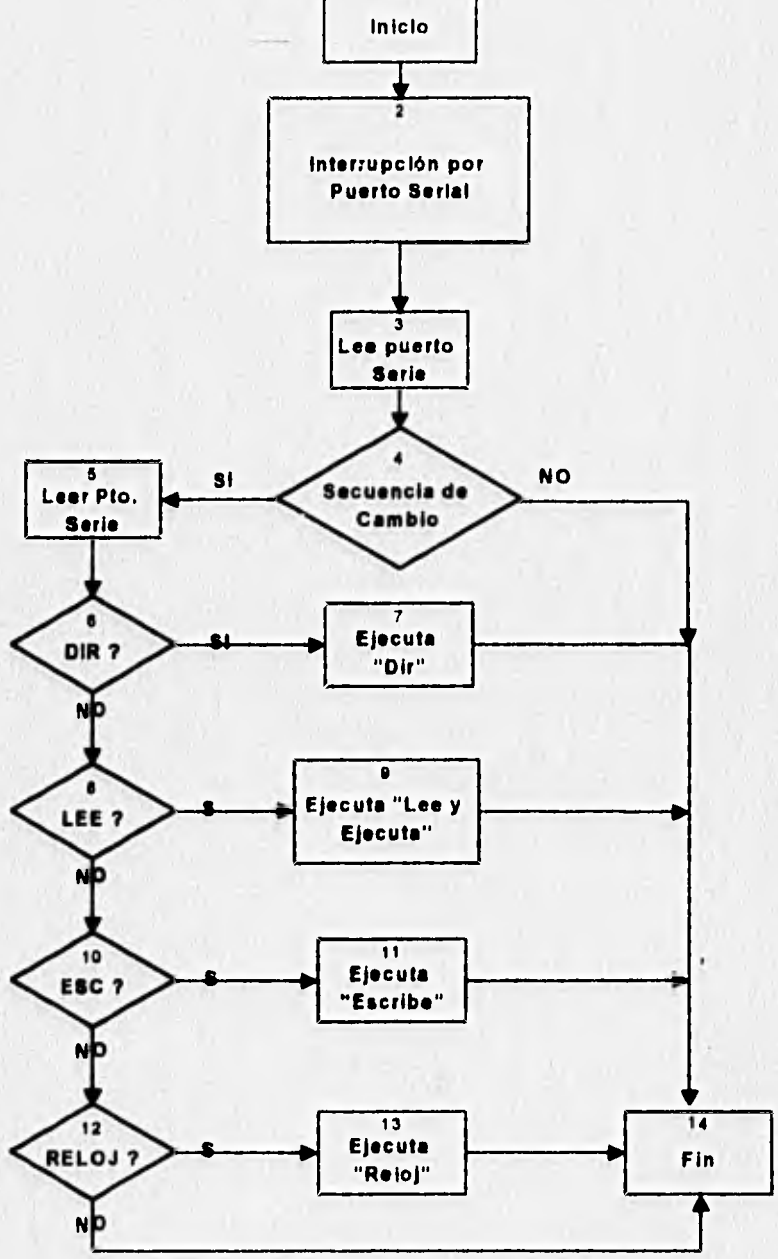
1
Inicio
2
Interrupción por Puerto Serial
3
Lee puerto Serie
4
Secuencia de Cambio
SI
NO
5
Leer Pto. Serie
6
DIR ?
SI
7
Ejecuta "Dir"
NO
8
LEE ?
S
9
Ejecuta "Lee y Ejecuta"
NO
10
ESC ?
S
11
Ejecuta "Escribe"
NO
12
RELOJ ?
S
13
Ejecuta "Reloj"
14
Fin
NO

**Figura IV.7**

## IV.5) Lineamientos para la programación del sensor

Este subíndice tiene por objeto el proveer de los elementos indispensables, a quien desarrolle el software de control del sensor(es) o estación(es) de muestreo, para manipular la tarjeta y el acceso a disco.

A continuación se muestran los mapas de memoria (Tabla IV.1), de interrupciones (Tabla IV.2) y el mapa de puertos (Tabla IV.3).

| MEMORIA | DIRECCION | COMENTARIO |
|---|---|---|
| RAM 0 | 00000h a 0002Fh | Reservado para 12 vectores de interrupción |
| | 00030h a 001FFh | Espacio reservado para el Stack |
| | 00200h a 01FFFh | Destinada al programa de los sensores. |
| RAM 1 | 20000h a 27FFFh | Zona libre para almacenamiento de datos |
| RAM 2 | 40000h a 47FFFh | Zona libre para almacenamiento de datos |
| 8259 | 60000h | 1a. palabra de control ICW1 |
| | 60001h | 2a. palabra de control ICW2 (vector y nivel) |
| | 60001h | 4a. palabra de control ICW4 |
| | 60001h | 1a. palabra de operación OCW1 (máscara) |
| 8255 | 80000 | Lectura o escritura de registros |
| ROM | F0000h a FFFFFh | Programa de inicialización, e interfaces de disco y usuario. |

**Tabla IV.1**

| INTERRUPCIONES | NUMERO | COMENTARIO |
|---|---|---|
| Software | | Son llamadas por el programador |
| Div | INT 01h | |
| Lee y Escribe | INT 02h | |
| Serie | INT 03h | |
| Hardware | | Son ejecutadas por el circuito |
| Acceso a Disco | 06h | |
| Puerto Serie | 07h | |

**Tabla IV.2**

| PUERTO | DIRECCION | COMENTARIO |
|---|---|---|
| Canal 0 | 00h | Puerto de Salida |
| Canal 1 | 01h | Puerto de Entrada |
| Canal 2 | 02h | Puerto de Salida |
| Modo | 03h | Programación. |
| Canal 0 | 40h | Lectura y/o escritura de datos. |
| Canal 1 | 41h | Programación. |
| Canal 2 | 84h | Canal 2 (dirección) |
| Canal 3 | 85h | Canal 2 (Terminal Count) |
| Page Register | C0 | Habilitador del Page Register |
| Canal 0 a 3 | D0 a D3 | Frecuencia 1 a 4 respectivamente |
| Canal 4 | E0 | Selección de registros |

**Tabla IV.3**

# CONCLUSIONES

Consideramos que los objetivos de flexibilidad y capacidad de almacenamiento del proyecto se cumplieron satisfactoriamente. Desarrollamos un controlador que se adecua a las necesidades de los proyectos de medición de perfiles de distribución de agua así como a estaciones de muestreo sísmicas y climatológicas sin tener la limitante de esclavizar una PC y con la ventaja de poder manejar la información obtenida en diskettes que pueden ser facilmente transportables.

Por otra parte se simplificó el uso de la tarjeta al permitir que el desarrollador del programa de control de los sensores accese a disco en forma similar a como se realiza en un ambiente de MS-DOS.

El acceso a las rutinas básicas de disco, video, teclado, etc. en MS-DOS se realizan invocando vía código máquina (INT xx) una interrupción para el propósito específico. Por lo tanto, las rutinas creadas por nosotros para el acceso a disco están grabadas en la EEPROM de la tarjeta para ser llamadas como interrupciones simulando el BIOS de cualquier PC.

Aunque inicialmente el alcance contemplado para este proyecto fue solamente el diseño conceptual de la tarjeta controladora, se desarrollaron programas que se probaron sustituyendo en una PC las rutinas de MS-DOS para el manejo de disco. Se pudo constatar que estas efectivamente simulan lo que el sistema operativo realiza, ya que los archivos grabados a disco pudieron ser leídos en otras PC's y exportados a algunas aplicaciones como Lotus o Excel.

Estos programas tuvieron que ser adecuados minímamente para poder ser operados en la tarjeta controladora, pero las rutinas principales para accesar el

"drive", leer el directorio del disco, leer un archivo en disco y bajarlo a memoria así como la rutina de escribir a disco conservan integramente su estructura. Las adecuaciones que se realizaron fueron el sustituir algunas rutinas especiales desarrolladas para la interfaz con el usuario.

Adicionalmente, se implementó el prototipo llegando hasta las fases de prueba de la rutina de inicialización.

Estamos conscientes de que el diseño de esta tarjeta controladora puede ser mejorado. En hardware, se le puede aumentar su capacidad de almacenamiento tanto en memoria como en disco.

En caso de que se desee crecer la memoria, en el diseño se deja la posibilidad de subsegmentar el mapa de memoria. En el caso de almacenamiento a disco, puede incluirse una tarjeta controladora de alta densidad para discos de 1.44 Mb de 3.5 in. o disco duro, debiéndo hacer los cambios necesarios en el programa del "Device Driver".

Se pueden incluir elementos adicionales de entrada/salida dependiendo de las necesidades de los sensores.

Debemos mencionar que se optó por utilizar un drive de baja densidad (360 kbytes), ya que la PC donde se efectuaron las pruebas de las rutinas contaba con este tipo de dispositivo, sin embargo, como fue tratado en el capítulo correspondiente, el utilizar un drive de mayor densidad no implica ni representa

diferencias sustanciales con el programa diseñado salvo por un punto en el cual los bytes por sector pueden ser de 1024 en lugar de 512.

Aunque actualmente existen en el mercado procesadores de la misma familia pero de mayor capacidad y rapidez consideramos que esto no es una limitante para el sistema, debido a que el 8088 cumple sobradamente con los requerimientos del diseño. Comparando este procesador con el Z80 que originalmente fue una alternativa, el 8088 tiene la capacidad de direccionar hasta 1Mb de memoria y existe toda una familia de dispositivos que facilitan el diseño como los son el controlador de interrupciones, el reloj y el DMA.

En el aspecto de software, en especial en la interfaz con el usuario, se pueden adicionar comandos para el formateo de discos, para el borrado de archivos y para el copiado de los mismos, así como rutinas especiales para el manejo de la información.

# ANEXOS

## PROGRAMA DE INICIALIZACION

```
; Programa principal
MAIN            Proc    FAR
                PUSH    DS
                mov     AX,0
                PUSH    AX
                mov     AX,DATA
                mov   DS,AX
                mov     ES,AX
; Inicio
                mov     AX,0000H
                mov   DS,AX
                mov     AX,07FFH
                mov     SS,AX
;               mov     AX,0F000H
;               mov     CS,AX
; Inicializacion del Puerto Paralelo, la palabra de control es 82
; Puertos A(00) y C(02) salidas, puerto B(01) es entrada
; Mandarlo por el 03 para quese guarde como palabra de control
                mov     AL,082h
                OUT     03,AL
                mov     AL,7Fh
INICIO:         mov     BX,0FFFFH
LOOP:           OUT     02,AL
                DEC     BX
                JZ      OTRO
                JMP     LOOP
OTRO:           CMP     AL,00H
                JE      FIN1
                SHR     AL,1
                JMP     INICIO
FIN1:           mov     AL,0FFH
                OUT     02,AL
                NOP
; Los cuatro primeros switches establecen un estado especial, el puerto C
; despliega el numero leido del puerto B(01)
                IN      AL,01
                CMP     AL,00
                JE      CERO
                CMP     AL,01
                JE      UNO
                CMP     AL,02
                JE      DOS
                CMP     AL,04
                JE      TRES
                CMP     AL,08
```

```
                    JE          CUATRO
                    mov   AL,086H
                    OUT         02,AL
                    JMP         FIN2
CERO:               mov         AL,0C0H
                    OUT         02,AL
                    JMP         FIN2
UNO:                mov         AL,0F9H
                    OUT         02,AL
                    JMP         FIN2
DOS:                mov         AL,0A4H
                    OUT         02,AL
                    JMP         FIN2
TRES:               mov         AL,0B0H
                    OUT         02,AL
                    JMP         FIN2
CUATRO:             mov         AL,099H
                    OUT         02,AL
FIN2:               NOP

; Verificacion de las tres RAM. Se graba "00" en toda la memoria validando
; su contenido, despues se graba "FF" revisando que todas las localidades
; de memoria tengan este dato. Se despliega mensaje numerico si la memoria
; opera correctamente y alfabetico en el caso contrario
; RAM0
                    mov    AX,0000H
;DS en 0000h para esta memoria
                    mov         DS,AX
                    mov         AL,00H
                    mov         BX,00H
SIGUE:              mov   [BX],AL
                    INC         BX
                    CMP         BX,0800H
                    JE          VALIDA
                    JMP         SIGUE
VALIDA:             mov   BX,00H
VALIDA1:            CMP         AL,[BX]
                    JNE         ERROR
                    INC         BX
                    CMP         BX,0800H
                    JNE         VALIDA1
                    mov   AL,0FFH
                    mov         BX,00H
SIGUE1:             mov         [BX],AL
                    INC         BX
                    CMP         BX,0800H
                    JE          VALIDA2
                    JMP         SIGUE1
VALIDA2:            mov         BX,00H
```

```
VALIDA3:            CMP     AL,[BX]
                    JNE     ERROR
                    INC     BX
                    CMP     BX,0800H
                    JNE     VALIDA3
                    mov     AL,0F9H
                    OUT     02,AL
;Mensaje "1" RAM0 OK!
                    JMP     RAM1
ERROR:              mov     AL,088H
;ERROR "A" problemas con la RAM0
                    OUT     02,AL
; RAM1
RAM1:               mov     AX,02000H
;DS en 2000h para la memoria RAM1
                    mov     DS,AX
                    mov     AL,00H
                    mov     BX,00H
SIGUE2:             mov     [BX],AL
                    INC     BX
                    CMP     BX,08000H
                    JE      VALIDA4
                    JMP     SIGUE2
VALIDA4:            mov     BX,00H
VALIDA5:            CMP     AL,[BX]
                    JNE     ERROR1
                    INC     BX
                    CMP     BX,08000H
                    JNE     VALIDA5
                    mov     AL,0FFH
                    mov     BX,00H
SIGUE3:             mov     [BX],AL
                    INC     BX
                    CMP     BX,08000H
                    JE      VALIDA6
                    JMP     SIGUE3
VALIDA6:            mov     BX,00H
VALIDA7:            CMP     AL,[BX]
                    JNE     ERROR1
                    INC     BX
                    CMP     BX,08000H
                    JNE     VALIDA7
                    mov     AL,0A4H
;Mensaje "2" RAM1 OK!
                    OUT     02,AL
                    JMP     RAM2
ERROR1:             mov     AL,083H
;ERROR "b" problemas con la RAM1
                    OUT     02,AL
```

```
; RAM2
RAM2:                   mov     AX,4000H
;DS en 4000H para la RAM2
                        mov             DS,AX
                        mov             AL,00H
                        mov             BX,00H
SIGUE4:                 mov             [BX],AL
                        INC             BX
                        CMP             BX,08000H
                        JE              VALIDA8
                        JMP             SIGUE4
VALIDA8:                mov             BX,00H
VALIDA9:                CMP             AL,[BX]
                        JNE             ERROR2
                        INC             BX
                        CMP             BX,08000H
                        JNE             VALIDA9
                        mov             AL,0FFH
                        mov             BX,00H
SIGUE5:                 mov             [BX],AL
                        INC             BX
                        CMP             BX,08000H
                        JE              VALIDA10
                        JMP             SIGUE5
VALIDA10:               mov             BX,00H
VALIDA11:               CMP             AL,[BX]
                        JNE             ERROR2
                        INC             BX
                        CMP             BX,08000H
                        JNE             VALIDA11
                        mov             AL,0B0H
;Mensaje "3" RAM2 OK!
                        OUT             02,AL
                        JMP             FINRAM
ERROR2:                 mov             AL,0C6H
;ERROR "C" problemas con la RAM2
                        OUT             02,AL
FINRAM:                 mov             CX,0FFFFH
SIGUE6:                 NOP
                        LOOP            SIGUE6

; Inicializacion de la palabra de control del DMA
; Para habilitar el DMA e indicarle que es la palabra de control
; la direccion es "88".
; La palabra de control es "66" habilitando solo el canal 2
; DMA
                        mov             AL,066H
                        OUT             88,AL
                        mov     AL,099H
```

```
;Mensaje "4" DMA OK!
                        OUT         02,AL
                        mov         CX,0FFFFH
SIGUE7:                 NOP
                        LOOP        SIGUE7

; Para habilitar el canal dos la salida debe mandarse a la "84"
; la direccion y a la "85" el tamaño indicando la operacion en
; b14 para escribir y en b15 para leer.

; Inicializacion del Controlador de Interrupciones
; Prueba de la tarjeta. Prender el Motor del Drive

                        mov         DX,3F2H
                        mov         AL,28
                        OUT         DX,AL


; Inicializacion del Reloj de Tiempo Real
                        mov         AX,8000H

;Habilitamos el clock por memoria
                        mov         DS,AX
                        mov         BX,00H
                        mov         AL,01H
;Mandar activar "STOP"

                        OUT         00,AL
                        mov         CX,00H
                        mov         AL,00H
                        OUT         0E0H,AL
;Selecciona UNIDADES de seg.
                        mov         CX,09H
                        mov         [BX],CX
;Cargamos con 09 las unidades de seg.
                        INC         AL
                        OUT         0E0H,AL
;Selecciona DECENAS de seg.
                        mov         CX,05H
                        mov         [BX],CX
;Cargamos con 05 las decenas de seg.
                        INC         AL
                        OUT         0E0H,AL
;Selecciona UNIDADES de min.
                        mov         CX,08H
                        mov         [BX],CX
;Cargamos con 08 las unidades de min.
                        INC         AL
                        OUT         0E0H,AL
```

```
;Selecciona DECENAS de min.
                        mov     CX,04H
                        mov     [BX],CX
;Cargamos con 04 las decenas de min.
                        INC     AL
                        OUT     0E0H,AL
;Selecciona UNIDADES de hora.
                        mov     CX,02H
                        mov     [BX],CX
;Cargamos con 02 las unidades de hora.
                        INC     AL
                        OUT     0E0H,AL
;Selecciona DECENAS de hora.
                        mov     CX,01H
                        mov     [BX],CX
;Cargamos con 01 las decenas de hora.
                        mov     AL,00H
;Desactivamos el "STOP"
                        OUT     00,AL
                        mov     AL,092H
;Mensaje "5" REAL CLOCK OK!
                        OUT     02,AL
                        mov     CX,0FFFFH
;SIGUE8:                 NOP
                        LOOP    SIGUE8
; Validar la correcta opecación del Reloj de Tiempo Real
                        mov     BX,00H
                        mov     AL,01H
;Activamos el "STOP"
                        OUT     00,AL
                        mov     DL,00h
;OTRO1:                  OUT     0E0H,AX
;Seleccionamos
                        mov     AL,[BX]
                        OUT     02,AL
                        mov     CX,0FFFFH
;SIGUE9:                 NOP
                        LOOP    SIGUE9
                        mov     CX,0FFFFH
;SIGUE10:                NOP
                        LOOP    SIGUE10
                        INC     AX
                        JMP     OTRO1
; Inicializacion del Puerto Serie
```

**PROGRAMA DE DIR**

```
data segment    para    public  'data'
;Cadenas a desp. (DEPURACION) con INT de soft. 21H func. 9 de MS-DOS.
string0 db     'error al leer sect. boot',13,10,'$'
string1 db     'disco no formateado',13,10,'$'
string2 db     'no tiene 512 bytes por sector',13,10,'$'
string3 db     '1. DIR, 2. LEE, 3. ESCRIBE : ?',13,10,'$'
string4 db     'Dame el archivo :',13,10,'$',
string5 db     'Archivo no encontrado',13,10,'$'
string6 db     'error al leer sect. dir.',13,10,'$'
string7 db     'error al leer sect. fats',13,10,'$'
string8 db     'error al leer SECTOR',13,10,'$'
string9 db     'No hay espacio en disco',13,10,'$'
string10 db    'Dame la long. del arch.:',13,10,'$'
string11 db    'Aqui voy...',13,10,'$'
string12 db    'error al esc SECTOR',13,10,'$'
string13 db    'error al esc sect. dir.',13,10,'$'
string14 db    'error al esc sect. fats',13,10,'$'

;Vars. globales en DS para almacenar inf. de disco o para referencia:
;area de Xferencia de disk. a mem. (INT 25H) por sec. log. 0 (pag 388 Adv.)
        bufzero db      512     dup     ('a')
;area de Xferencia de disk. a mem. (INT 25H) por 1 sec. log. (pag 388 Adv.)
        buffsec db      512     dup     ('a')
;area de Xferencia de disk. a mem. (INT 25H) por n sec. de dir.
        buffdir db      7168    dup     ('0')
;area de Xferencia de disk. a mem. (INT 25H) por n sec. de fats
        buffat  db      5120    dup     ('0')
;buffer para escritura de nombre de archivo.
        buffnom db      11      dup     ('20H')
;buffer para escritura de fecha hora start_clus, y tamaño de la zona de
directorio
        datosdir db     10      dup     ('20H')
;varible con datos a escribir a un archivo de disco
        archivo  db     512     dup     ('00112233445566778899,AEIOU')
;bandera de error entre rutinas
        banderr db      1       dup     ('1')
;bandera para determinar si se encontro un arch.
        arch_en db      1       dup     ('0')
;var. para el sec. de inic. de directorio
        secdir  dw      1       dup     ('0')
;var. para el INICIO de area de datos, es decir EL 1er SECTOR
        secdat  dw      1       dup     ('0')
;var. para num. de sect. por cluster
        seclus  dw      1       dup     ('0')
```

```
;var. para total de sectores logicos en disco
        totsec  dw      1       dup     ('0')
;var. para sector logico  pag. 170 adv. msdos
        sector  dw      1       dup     (0)
;var. para num. de sec. en dir.
        numsecdir dw    1       dup     (0)
;var. para num. de sec. de fats
        numsecfat  dw   1       dup     ('0')
;var. para num. de ent. en dir
        numentdir  dw   1       dup     ('0')
;var. para num. de sec. de inic. de fats
        secfat  dw      1       dup     ('0')
;var. para guardar cluster
        cluster dw      1       dup     ('0')
;variable para lectura de tecla
        opcion  db      1       dup     ('1')
;var. para guardar ultimo offset en fat
        last_clus  dw   1       dup     ('0')
;var. para guardar offset de entry libre
        entry   dw      1       dup     ('0')
;var. para guardar offset de arch enc.
        off_endir_enc   dw      1       dup     ('0')
;bandera arch borrado.
        arch_borr       db  1       dup     (0)
;bandera entry en directorio llenos
        entry_full      db  1       dup     (0)
;var. para guardar offset de arch borrado
        off_borr        dw  1       dup     ('0')
;var. para guardar lsw del tamano de arch.
        fsize_lsw       dw  1       dup     ('0')
;var. para guardar msw del tamano de arch.
        fsize_msw       dw  1       dup     ('0')
;variable para cta. de clusters
        ctaclus dw      1       dup     (0)
; var. para apuntar a la fat
        pointer dw      1       dup     (0)
;variable para denotar cluster par o impar
        par     db      1       dup     (0)
;tabla para guardar los clusters libres al recorrer fat
        clus_libre      dw      100     dup     (0)
; variable para guardar direcciones
        tabla   dw      1       dup     (0)
;variable
        clusterslibres dw       1       dup     (0)
;variable
        resul   dw      1       dup     (0)
;variable para saber si se trata del 1er cluster
        primcluster     db      1       dup     (0)
```

```
;variable para dimension de arch.
        necesito      dw     1      dup     (0)
;variable
        last_off_fat  dw     1      dup     (0)
;clusters de un arch. nvo.
        clus_a_esc    dw     1      dup     (0)
;clusters de un arch. viejo
        realsize      dw     1      dup     (0)
;variable para definir a la hora de escribir arch. si:
;0=off_borr, 1=off_endir_enc, 2=entry
        entryclas     dw     1      dup     (0)
;variable
        posicion      dw     1      dup     (0)
;tabla para guardar clusters de un arch. encontrado
        clus_de_arch  dw     100    dup     (0)

    offsetfat       dw     1      dup     (0)
    suficiente      db     1      dup     (0)
    data ends


code segment    para    public 'code'
            assume  cs:code,DS:DATA,ES:DATA

; Programa principal
MAIN            Proc    FAR
; inicio
                        PUSH        DS
                        mov         AX,0
                        PUSH        AX
                        mov         AX,DATA
                        mov         DS,AX
                        mov         ES,AX

; si no error, y segun la opcion seleccionada, es decir,
; obtener DIR, LEER archivo o ESCRIBIR archivo se hace:
                        mov         dx,offset string3 ;dame opcion
                        call        desp_mje
                        call        lee_tecla
                        mov         [opcion],al
; para ver que caracter se lee, se desplegara en pantalla
; despues del echo, usando chrout
                        cmp         [opcion],49   ;dir
                        je          directorio
                        cmp         [opcion],50   ;lec
                        je          leo
                        cmp         [opcion],51   ;ESC
                        je          escribo
```

```
directorio:             call        desp_dir
                        jmp         fin_main
leo:                    call        lee_arch
                        jmp         fin_main
escribo:                call        esc_arch
fin_main:               nop
                        ret
MAIN                    endp


; Esta rutina lee y guarda en 'bufzero' la inf. del sec. log. 0,
; da valor a las variables SECDIR, NUMSECDIR, SECDAT,
; SECLUS, NUMSECFAT, y SECFAT, si hay error, BANDERR
; se queda con 1 . Antes se llamaba basis
lee_seczero     proc
                        mov         [banderr],1
                        mov         al,1 ;drive b=1 drive a=0
                        mov         cx,1
                        mov         dx,0
                        mov         bx,offset bufzero
                        int         25h
                        jc          errbios
                        popf
; se despliega la inf. leida con desp
                        mov         cx,512
                        mov         bx,00h
                        mov         si,offset bufzero
;                       call        desp
                        jmp         cont
errbios:                mov         dx,offset string0 ;error
                        call        desp_mje
                        jmp         fin_zero
cont:                   mov         si,offset bufzero
                        mov         bx,0
                        mov         al,[si+bx]
                        cmp         al,0e9h
                        je          siformat
                        cmp         al,0ebh
                        je          siformat
                        mov         dx,offset string1 ;no format
                        call        desp_mje
                        jmp         fin_zero
siformat:               mov         bx,16h
                        mov         dx,[si+bx]  ;sectors/fat
                        mov         [numsecfat],dx
;num. sec. ocup. por 1 fat
                        mov         al,[si+10h] ;number of fats
                        cbw         ;se hace WORD
```

```
                        mul     dx
;en AX sect. ocup. por fats = sectors/fat*fats
                        mov     bx,[si+0eh] ;sectores reservados
                        mov     [secfat],bx ;inicio de fats
                        add     ax,bx
;en AX suma de sect. de fats y sect. res.
                        mov     [secdir],ax ;inicio de directorio
                        mov     ax,[si+11h] ;num. de ent. en dir.
                        mov     [numentdir],ax
                        mov     cx,32
                        mul     cx
;en AX hay 32 por num. ent. en dir.
                        mov     cl,[si+0ch] ; bytes/sector
                        cmp     cl,02h
                        jz      b512        ; si 512b/sec salta a b512
                        mov     dx,offset string2 ;no 512b/sec
                        call    desp_mje
                        jmp     fin_zero
b512:                   mov     cx,512
                        div     cx
                        mov     [numsecdir],ax
;num de sec en directorio
                        add     ax,[secdir]
                        mov     [secdat],ax ;inicio area datos
                        mov     al,[si+0dh]
                        cbw
                        mov     [seclus],ax ;sectores por cluster
                        mov     ax,[si+13h]
                        mov     [totsec],ax

; si hasta aqui no hubo error, entonces banderr se hace cero
                        mov     [banderr],0
fin_zero:               nop
                        ret
lee_seczero             endp


; Se determina la cap. libre del disco cuando se accesa desde
; la opcion de dir. o bien si se quiere esc. un archivo, se
; van determinando los clusters libres, y se va llenando tabla
; CLUS_LIBRE con CLUSTERS.
; Se llama desde lee_fat
buscacluster    proc
                        mov     [ctaclus],0
                        mov     si,offset buffat
                        mov     bx,0003h
denuevo:                cmp     bx,[last_off_fat]
                        Ja      salida
```

```
con_busq_libre:     mov     ax,[si+bx]
                    and     ax,0fffh
                    jnz     inc1
                    inc     [clusterslibres]
no_libre:           cmp     [opcion],49
                    je      inc1
                    mov     [par],1
                    call    esc_clus
                    cmp     [necesito],0
                    je      EOF
inc1:               inc     bx
                    cmp     bx,[last_off_fat]
                    Ja      salida
                    mov     ax,[si+bx]
                    and     ax,0fff0h
                    jnz     inc2
                    inc     [clusterslibres]
no_libre2:          cmp     [opcion],49
                    je      inc2
                    mov     [par],0
                    call    esc_clus
                    cmp     [necesito],0
                    je      EOF
inc2:               inc     bx
                    inc     bx
                    jmp     denuevo
EOF:                mov     di,[ctaclus]
                    mov     [clus_libre+di],0fffh  ;genera el EOF
salida:             nop
                    ret
buscacluster        endp


; Rutina para lectura de los NUMSECFAT sec. de las fats en 'buffat'
; a partir del sector logico SECFAT, con INT 25H
; Se determina ademas el espacio libre o el que se ocupara al esc. arch.
lee_fat     proc
                    mov     [banderr],1
                    mov     al,1 ;drive b=1 drive a=0
                    mov     cx,[numsecfat]
                    mov     dx,[secfat]
                    mov     bx,offset buffat
                    int     25h
                    jc      errorlecfat
                    popf
; se previene crecimiento incontrolado del stack
                    cmp     [opcion],50   ;lec
                    je      saltabusca
```

```
                        cmp         [suficiente],1
;se va a esc. un arch. que cabe
                        je          saltabusca
; se hace calculo de ESPACIO LIBRE en disco:
                        mov         ax,[totsec]   ;2880
                        sub         ax,[secdat]  ;ax-secdat  2847
                        mov         dx,0h
                        div         [seclus]
;en ax quedan clusters tot en datos 2847
                        add         ax,2        ;por el inicio de cluster con 2
                        sub         ax,1
;porque el offset empieza en 0
                        mov         [last_clus],ax ;2849
                        mov         cx,03
;se mult. por 1.5 para obt. bytes
                        mul         cx          ;8541
                        shr         ax,1        ;divide entre 2
                        jnc         sumpar
                        add         ax,1
sumpar:                 mov         [last_off_fat],ax  ;4272
                        call        buscacluster
; Segun opcion se determina espacio o clusters a usar para esc. arch.
saltabusca:             mov         [banderr],0
                        jmp         fin_secfat
errorlecfat:            mov         dx,offset string7 ;error al leer buff. fat
                        call        desp_mje
fin_secfat:             nop
                        ret
lee_fat                 endp


; Rutina para escritura de los NUMSECFAT sec. de 'buffat' en la FAT
; del disco a partir del sector logico SECFAT, con INT 26H
esc_fat         proc
                        mov         [banderr],1
                        mov         al,1 ;drive b=1 drive a=0
                        mov         cx,[numsecfat]
                        mov         dx,[secfat]
                        mov         bx,offset buffat
                        int         26h
                        jc          errorescfat
; se previene crecimiento incontrolado del stack
                        popf
                        mov         [banderr],0
                        jmp         finesc_fat
errorescfat:            mov         dx,offset string14 ;error al esc. buff. fat
                        call        desp_mje
finesc_fat:             nop
```

```
                        ret
esc_fat                 endp

;RUTINA PARA GUARDAR EL NUMERO DE CLUSTER LIBRES PARA
ESCRIBIR ARCHIVO

ESC_CLUS        PROC
                        mov             ax,0000h ; incializacion de ax
                        mov             ax,bx  ; en bx se tiene offset
                        mov             cx,2
                        mul             cx          ; ax=ax*2
                        mov             cx,3
                        div             cx          ; ax=ax/3  1er. clust libre
                        cmp             [par],0
                        jnz             nosedonde
                        add             ax,1
nosedonde:              push            bx
                        mov             bx,[ctaclus]
                        mov             [clus_libre+bx],ax
                        ADD             [ctaclus],2       ;tiene que ir de 2 en 2!

                        pop             bx
                        dec             [necesito]
ESC_CLUS                ENDP

; Rutina para lectura de los NUMSECDIR sec. del directorio en 'buffdir'
; a partir del sector logico SECDIR, con INT 25H
lee_dir         proc
                        mov             [banderr],1
                        mov             al,1 ;drive b=1 drive a=0
                        mov             cx,[numsecdir]
                        mov             dx,[secdir]
                        mov             bx,offset buffdir
                        int             25h
                        jc              errorlecdir
; se previene crecimiento incontrolado del stack
                        popf
                        mov             [banderr],0
                        jmp             fin_secdir
errorlecdir:            mov             dx,offset string6 ;error al leer buff. dir
                        call            desp_mje
fin_secdir:             nop
                        ret
lee_dir                 endp


; Rutina para escritura de los NUMSECDIR sec. en 'buffdir' al directorio
; de disco a partir del sector logico SECDIR, con INT 26H
```

```
esc_dir     proc
                        mov         [banderr],1
                        mov         al,1 ;drive b=1 drive a=0
                        mov         cx,[numsecdir]
                        mov         dx,[secdir]
                        mov         bx,offset buffdir
                        int         26h
                        jc          errorescdir
; se previene crecimiento incontrolado del stack
                        popf
                         mov        [banderr],0
                        jmp         fin_escdir
errorescdir:            CALL        VIDEO_WORD
; SE REGRESA ERROR EN AX
                        mov         dx,offset string13 ;error al esc. buff. dir
                        call        desp_mje
fin_escdir:             nop
                        ret
esc_dir                 endp


; se lee el sec. log. cero o BOOT SECTOR
; se lee la zona de fat y la de directorio
inicializacion  proc
                        call        lee_seczero
                        call        lee_fat ; se llena buffat y espacio disp.
                        call        lee_dir
                        ret
inicializacion  endp



; Rutina que despliega el dir., usando la inf.
; depositada en buffdir por lee_dir.
desp_dir    proc
                        CALL        INICIALIZACION
;funciona a medias,
                        call        scan_dir
;hay riesgo de overflow
                        mov         ax,[clusterslibres]
                        mov         cx,512
                        mul         cx
                        mov         cx,[seclus]
                        mul         cx
                        mov         bh,al
                        mov         bl,ah
                        mov         ax,bx
                        mov         cx,2
```

```
                        mov     [resul],ax
                        mov     si,offset resul
                        call    desp
                        ret
desp_dir                endp


; Rutina para barrer la zona de dir en disco y tomar acciones
; segun la opcion que siga el usuario. Se llama desde todas las
; opciones (Desp. dir. leer  o  escribir arch.)
scan_dir        proc
                        mov     [arch_borr],0
                        mov     [entry_full],0
                        mov     si,offset buffdir
                        mov     bx,00h
                        mov     cx,[numentdir]
otroarch:               mov     al,[si+bx]
                        cmp     al,00h
                        je      entry_zero
                        cmp     al,0e5h
                        je      borrados
                        cmp     al,02eh
                        je      buscaotro  ; especial (ja)
                        cmp     [opcion],49    ;dir
                        jne     ve_si_esc  ;para no desp. archivo
                        call    nomarch ;se apunta con BX en SI

ve_si_esc:              cmp     [opcion],51    ;esc
                        je      comparanom
                        cmp     [opcion],50    ;lee
                        jne     buscaotro
comparanom:             call    compa_nom
                        cmp     [arch_en],1
                         jne    buscaotro
                        mov     [off_endir_enc],bx
; bx en var. p' luego ai' meterlo
; traer 4 bytes del filesize en los regs. DX Y AX y llenar var [fsize]
                        add     bx,1ch
                        mov     ax,[si+bx] ;LSW
                        add     bx,2
                        mov     dx,[si+bx] ;MSW
                        mov     [fsize_lsw],ax
                        mov     [fsize_msw],dx
                j       mp      fin_scan_dir
buscaotro:              loop    aviso
                        mov     [entry_full],1
                        jmp     fin_scan_dir
borrados:               cmp     [opcion],51  ;esc
```

```
                        jne         buscaotro
                        cmp         [arch_borr],1
                        je          buscaotro
                        mov         [arch_borr],1       ;falta declarar
                        mov         [off_borr],bx
                        jmp         buscaotro
aviso:                  mov         dx,32
                        add         bx,dx
                        jmp         otroarch
entry_zero:             cmp         [opcion],51
                        jne         fin_scan_dir
                        mov         [entry],bx
fin_scan_dir:           nop
                        ret
scan_dir                endp


; Rutina que obtiene un cluster, el cual se guarda en [cluster]
obten_clusters  proc
                        cmp         [primcluster],1
; si no es el 1er. cluster salta
                        jne         clu_sig
                        mov         si,offset buffdir
                        mov         bx,[off_endir_enc]
                        mov         cx,1ah
                        add         bx,cx
; se suma a lo que llevaba bx 1ah
                        mov         cx,[si+bx]
                        mov         [cluster],cx
                        mov         [primcluster],0
                        jmp         fin_obt
; se puede usar SI para manejar buffat
clu_sig:                mov         si,offset buffat
                        mov         ax,[cluster]
                        mov         cx,3        ; mult. por 1.5
                        mul         cx          ; ax=ax*3
                        clc                     ; clear carry flag
                        shr         ax,1        ; corr. a la der. (div/2)
                        mov         bx, ax
; usar parte entera como offset en FAT
                        mov         ax,[si+bx]
; guardar la palab. en ese offset en reg. AX
                        jnc         op_and
; si el prod. fue ent. (no carry) salt. a op_and
                        mov         cl,4
                        shr         ax,cl
                        jmp         limites
op_and:                 and         ax,0fffh
```

```
limites:                mov     [cluster],ax
;guardo 12 bits como CLUSTER
fin_obt:                nop
                        ret
obten_clusters          endp


; Esta rutina lee y guarda en 'buffsec' la inf. del SECTOR,
; se despliega y si hay error, B        ANDERR es dejada en 1
lee_sector      proc
                        PUSH    SI
                        push    ax
                        push    bx
                        push    cx
                        push    dx
                        mov     [banderr],1
                        mov     al,1 ;drive b=1 drive a=0
                        mov     cx,1 ; numero de sec. a leer
                        mov     dx,[sector] ; sector logico a leer
                        mov     bx,offset buffsec
                        int     25h
                        jc      err_bios
                        popf
; se despliega la inf. leida con desp
                        mov     cx,512
                        mov     bx,00h
                        mov     si,offset buffsec
                        call    desp
                        jmp     no_error
err_bios:               mov     dx,offset string8 ;error
                        call    desp_mje
                        jmp     fin_sector
; si hasta aqui no hubo error, entonces banderr se hace cero
no_error:               mov     [banderr],0
fin_sector:             nop
                        pop     dx
                        pop     cx
                        pop     bx
                        pop     ax
                        POP     SI
                        ret
lee_sector              endp


; Esta rutina escribe en SECTOR inf. mantenida en memoria,
; como puede ser cualquier buffer.
esc_sector      proc
                        push    ax
```

```
                        push    bx
                        push    cx
                        push    dx
                        mov     [banderr],1
                        mov     al,1 ;drive b=1 drive a=0
                        mov     cx,1 ; numero de sec. a escrbir
                        mov     dx,[sector] ; sector logico a leer
                        mov     bx,offset archivo
                        int     26h
                        jc      error_bios
                        popf
                        jmp     finescsector
error_bios:             call    video_word
                        mov     dx,offset string12 ;error
                        call    desp_mje
finescsector:           nop
                        pop     dx
                        pop     cx
                        pop     bx
                        pop     ax
                        ret
esc_sector              endp


; Rutina para lectura de un archivo a partir de sus CLUSTERS
; (es la inf. que maneja la FAT) y obtener SECTORES (es lo que
; reconoce la cabeza del drive). El archivo se lee con lee_nomarch.
lee_arch        proc
                        mov     dx,offset string4 ;dame arch.
                        call    desp_mje
                        call    lee_nomarch
                        CALL    INICIALIZACION
; se hace barrido de dir para buscar el arch.
                        call    scan_dir
                        cmp     [arch_en],0
                        je      fin_busqda
                        mov     [primcluster],1
; se entra a loop que sigue los clusters para DESPLEGAR el arch.
otro_cluster:           call    obten_clusters ; da valor a [cluster]
                        cmp     [cluster], 0ff8h
; p' no desplegar la eof mark
                        jae     fin_leearch
                        call    obt_sect
; con CLUSTER se obt. SECTOR
                        call    lee_sector
; se obt. y desp. el 1er sec./clu.
                        cmp     [seclus],1
                        je      otro_cluster
```

```
                        add     [sector],1
                        call    lee_sector
; se obt. y desp. el 2o. sec./clu.
                        jmp     otro_cluster
fin_busqda:             mov     dx,offset string5 ;arch. no enc.
                        call    desp_mje
fin_leearch:            nop
                        ret
lee_arch                endp

; rutina para escribir en la zona de datos del disco,
; dados los clusters -en clus_libre- y esc. en sectores fisicos
escribedatos    proc
                        mov     bx,0
otrolibre:              mov     ax,[clus_libre+bx]
                        mov     [cluster],ax
                        call    video_word
                        cmp     ax,0fffh
                        jz      finescdat
                        call    obt_sect
                        call    esc_sector
                        add     [sector],1
                        call    esc_sector
                        add     bx,2
                        jmp     otrolibre
finescdat :             nop
                        ret
escribedatos            endp


; Rutina para escritura de un archivo .-
esc_arch        proc
                        mov     [clus_a_esc],1
                        mov     dx,offset string4 ;dame arch.
                        call    desp_m                  je
                        call    lee_nomarch
                        CALL    lee_seczero
                        call    lee_dir
                        call    scan_dir
                        cmp     [entry_full],1 ; ya no hay entries ?
                        je      intenta_borr
                        cmp     [arch_en],1    ; se encontro arch ?
                        je      bypass1
                        cmp     [arch_borr],0
                        je      dos
                        jmp     uno
dos:                    mov     [entryclas],2
                        jmp     tres
```

```
intenta_borr:       cmp     [arch_borr],0
                    je      bypass
uno:                mov     [entryclas],0
tres:               mov     ax,[clus_a_esc]
                    mov     [necesito],ax
cuatro:             call    lee_fat
                    cmp     [necesito],0
                    jne     bypass
                    cmp     [entryclas],1
                    je      sihayarch
                    cmp     [entryclas],0
                    je      erased
                    mov     ax,[entry]
                    mov     [posicion],ax
                    jmp     escribe_dir
bypass1:            jmp     sigue_arch
erased:             mov     ax,[off_borr]
                    mov     [posicion],ax
escribe_dir:        call    escribedir
                    call    escribefat
                    call    escribedatos
                    jmp     fin_escarch
sihayarch:          call    lee_fat
                    mov     ax,[off_endir_enc]
                    mov     [posicion],ax
                    mov     [primcluster],1
                    mov     di,0
other_cluster:      call    obten_clusters ; da valor a [cluster]
                    cmp     [cluster], 0ff8h
                    jae     terminador
                    mov     ax,[cluster]
                    mov     [clus_de_arch+di],ax
                    add     di,2
;se ha de mover de 2 en 2
                    jmp     other_cluster
terminador:         cmp     [suficiente],1
                    jne     escribe_dir
                    mov     [clus_de_arch+di],0fffh
;EOF mark en clus_de_arch
                    jmp     escribe_dir
bypass:             jmp     noti_nohay
backpass:           jmp     cuatro
sigue_arch:         mov     ax,[fsize_lsw]
                    mov     dx,[fsize_msw]
; en AX se tiene el tamaño del arch. viejo
                    mov     cx,400h
                    div     cx
                    cmp     dx,0
```

```
                        je          unalinea
                        inc         ax
unalinea:               mov         [realsize],ax
;tamaño real en clusters del arch. viejo
                        sub         [clus_a_esc],ax
; clus_a_esc tiene el tamaño del nvo.
                        mov         ax,[clus_a_esc]
                        mov         [necesito],ax
                        call        video_word
                        mov         [entryclas],1
                        cmp         [necesito],0
                        ja          backpass
                        mov         [suficiente],1
; el arch. es menor o igual
                        jmp         sihayarch
; si no cabe el archivo se dep. mje.
noti_nohay:             mov         dx,offset string9 ;no cabe el arch.
                        call        desp_mje
fin_escarch:            nop
                        ret
esc_arch                endp


; Rutina para meter inf. del nvo. arch. en la zona de dir del disco
escribedir      proc
                        mov         si,offset buffdir
                        mov         bx,[posicion]
                        mov         si, offset buffnom
                        mov         di, offset buffdir
                        add         di,bx
                        mov         cx,11
                        rep         movsb

                ;       call        tiempo
;se simulan las 17:51:26 -> 1000,1  110,011  0,1101
                        mov         bx,0
                        mov         [datosdir+bx],6Dh
                        inc         bx
                        mov         [datosdir+bx],8Eh

                ;       call        fecha
;se simula el 20/11/92   -> 0001,100  1,011  1,0100
                        inc         bx
                        mov         [datosdir+bx],74h
                        inc         bx
                        mov         [datosdir+bx],19h
                        inc         bx
                        cmp         [entryclas],1
```

```
                        jne         jump_one
                        mov         ax,[clus_de_arch+0]
                        jmp         jump_two
jump_one:               mov         ax,[clus_libre+0]   ; cluster inic.
jump_two:               mov         [datosdir+bx],al
                        inc         bx
                        mov         [datosdir+bx],ah
                        inc         bx
                        mov         [datosdir+bx],00H
                        inc         bx
                        mov         [datosdir+bx],04h
                        inc         bx
                        mov         [datosdir+bx],00
                        inc         bx
                        mov         [datosdir+bx],00
; se simulan 400h bytes
                        mov         bx,[posicion]
                        mov         si, offset datosdir
                        mov         di, offset buffdir
                        add         di,bx
                        add         di,16h
                        mov         cx,10
                        rep         movsb

                        mov         si, offset buffdir
                        add         bx,0bh
                        mov         [buffdir+bx],32 ; para el atributo
                        call        esc_dir
                        ret
escribedir              endp


; Rutina para meter inf. del nvo. arch. -clus_libre y clus_de_arch-
; en la zona de fat del disco
escribefat      proc
                        mov         si,offset buffat
;inicializacion
                        mov         di,0
                        mov         [pointer],0
                        cmp         [arch_en],1
                        je          con_clus_arch
;se usan clusters libres:
fat_sig:                mov         ax,[clus_libre+di]
                        mov         cx,3        ; mult. por 1.5
                        mul         cx          ; ax=ax*3
                        clc                     ; clear carry flag
                        shr         ax,1        ; corr. a la der. (div/2)
                        mov         [offsetfat]
```

```
, ax; usar parte entera como offset en FAT
                        jnc             entero     ; (no carry) salt. a entero
                        add             [pointer],2
                        mov             di,[pointer]
                        mov             ax,[clus_libre+di]
                        mov             cl,4
                        shl             ax,cl
                        mov             bx,[offsetfat]
                        or              [si+bx],ax
                        cmp             ax,0fff0h
                        jne             fat_sig
                        jmp             finescfat ; interfin
entero:                 add             [pointer],2
                        mov             di,[pointer]
                        mov             ax,[clus_libre+di]
                        mov             bx,[offsetfat]
                        or              [si+bx],ax
                        cmp             ax,0fffh
                        jne             fat_sig
                        jmp             finescfat
; se tienen clusters de archivo:
con_clus_arch:          cmp             [suficiente],1
                        jne             liga
; si es suficiente, trabajo solo con clus_de_arch
                        mov             ax,[realsize]
                        cmp             [clus_a_esc],ax
                        jne             bigtrouble
fat_sig1:               mov             ax,[clus_de_arch+di]
                        mov             cx,3       ; mult. por 1.5
                        mul             cx         ; ax=ax*3
                        clc                        ; clear carry flag
                        shr             ax,1       ; corr. a la der. (div/2)
                        mov             [offsetfat], ax
; usar parte entera como offset en FAT
                        jnc             entero1    ; (no carry) salt. a entero
                        add             [pointer],2
                        mov             di,[pointer]
                        mov             ax,[clus_de_arch+di]
                        mov             cl,4
                        shl             ax,cl
                        mov             bx,[offsetfat]
                        or              [si+bx],ax
                        cmp             ax,0fff0h
                        jne             fat_sig1
                        jmp             finescfat
entero1:                add             [pointer],2
                        mov             di,[pointer]
                        mov             ax,[clus_de_arch+di]
```

```
                        mov     bx,[offsetfat]
                        or      [si+bx],ax
                        cmp     ax,0fffh
                        jne     fat_sig1
                        jmp     finescfat
; el arch. nvo. es mas pequeño que el original bigtrouble:
; el arch. nvo. es mas grande que el original y han de usarse las 2 estructuras
liga:
                        lea     ax,clus_libre
;                       mov     cx,[ax+di]
finescfat:              call    esc_fat
                        ret
escribefat              endp


; Rutina para desplegar cadenas con INT 21H func. 09H,
; la cadena es puesta antes del llamado en DS:DX
desp_mje                proc
                        push    ax
                        mov     ah,9
                        int     21h
                        pop     ax
                        ret
desp_mje                endp


; Rutina obtiene sector dada la var. CLUSTER y el valor
; obtenido se guarda en SECTOR
obt_sect        proc
; si se usa bx, guardarlo en stack!
                        push    [cluster]
                        dec     [cluster]
                        dec     [cluster] ;restar 2 al cluster
                        mov     ax,[cluster]
                        mul     [seclus]
;mult. por num.sec/cluster, res. en AX
                        add     ax,[secdat]
;sumar no de sec. antes. de arch.
                        mov     [sector],ax
                        pop     [cluster]
                        ret
obt_sect                endp


; Rutina para leer cararcteres de teclado con INT 21H func. 01H,
; la tecla regresa en AL, y ha de ser tomada por la inst. sig.
lee_tecla       proc
                        mov     ah,1
```

```
                        int         21h
                        ret
lee_tecla               endp


; Rutina para leer la long. de un archivo con lee_tecla, la long
; se guarda en clus_a_esc. y solo ha de ser de hasta 99
lee_longarch    proc
                        mov         [clus_a_esc],0
                        mov         ax,0
                        mov         bx,1; peso o posicion en la cifra
leeotro:                call        lee_tecla
                        cmp         al,13
; se compara con RET
                        je          fin_leelong
                        cmp         bx,1
                        je          nounit
; el valor en AL ha de estar entre 48(0) y 57(9) y se hace word
                        sub         ax,48
; se le resta 48 para obtener el VALOR
                        add         [clus_a_esc],ax
; se suma a [clus_a_esc]
                        jmp         fin_leelong
nounit:                 sub         ax,48
                        push        cx
                        mov         cx,0ah
                        mul         cx
                        pop         cx
                        add         [clus_a_esc],ax
; se suma a [clus_a_esc]
                        dec         bx
                        jmp         leeotro
fin_leelong:            nop
                        ret
lee_longarch            endp


; Rutina para leer el nombre de un archivo con lee_tecla, el nombre
; se guarda en buffnom, el cual es una variable que acepta 8 carac.
; para el nombre, 3 para la ext.
lee_nomarch     proc
                        mov         [banderr],1
                        mov         bx,0
                        mov         cx,11
limpiabx:               mov     [   buffnom+bx],20h
                        inc         bx
                        loop        limpiabx
                        mov         bx,0
```

```
leeotra:            call        lee_tecla
; el carac. es pto. y bx < 7 entonces dejar con 20H desde la pos. bx a la 7
                    cmp         al,2eH
                    jne         guarda2; si car <> pto.
                    mov         bx,8
                    jmp         leeotra
guarda2:            cmp         al,13
; se compara con RET
                    je          fin_leenom
                    mov         [buffnom+bx],al
                    inc         bx
                    jmp         ieeotra
fin_leenom:         nop
                    ret
lee_nomarch         endp


; Escribe, usando la rutina chrout, el nombre con ext. de un arch.
; el nombre se tiene con lo apuntado por SI+BX
nomarch     proc
                    push        bx
; se guarda, pues es apuntador
; se escribe nombre
                    mov         cx,8
otrocarnom:         mov         al,[si+bx]
                    call        chrout
                    inc         bx
                    loop        otrocarnom
; se escribe ext
                    mov         al,46
                    call        chrout
                    mov         cx,3
otrocarext:         mov         al,[si+bx]
                    call        chrout
                    inc         bx
                    loop        otrocarext
                    mov         al,13
                    call        chrout
                    mov         al,10
                    call        chrout
                    pop         bx
                    ret
nomarch             endp

; Compara el nombre del archivo actualmente leido (SI+BX) con el
; que se busca leer o escribir, y de ser igual, prende la bandera
; de arch_en.
```

```
compa_nom     proc
                        push        cx
                        push        bx
                        mov         di,0
                        mov         [arch_en],0
                        mov         cx,11 ;nombre+ext
otrocar:                mov         al,[si+bx]
                        cmp         al,[buffnom+di]
                        jne         diferente
                        inc         bx
                        inc         di
                        loop        otrocar
                        mov         [arch_en],1
diferente:              nop
                        pop         bx
                        pop         cx
                        ret
compa_nom               endp


;Usando el proc. VIDEO va desplegando, previa conversion a HEX,
;los CX bytes que llegan apuntados por SI
desp            proc
                        push        ax
                        push        bx
                        push        dx
                        mov         bx,0
otronum:                mov         al,[si+bx]
                        push        cx
                        push        ax
                        mov         dx,2
                        and         al,0f0h
                        dec         dx
                        mov         cl,4
                        ror         al,cl
compara:                cmp         al,0ah
                        jae         letra
                        add         al,30h
                        call        video
                        jmp         nibble
letra:                  add         al,37h
                        call        video
nibble:                 cmp         dx,0h
                        jz          ultimo
                        pop         ax
                        and         al,0fh
                        dec         dx
                        jmp         compara
```

```
ultimo:                 pop     cx
                        inc     bx
                        loop    ctronum
                        pop     dx
                        pop     bx
                        pop     ax
                        ret
desp                    endp


; Escribe caracter ASCII que ESTA en AL a video, usando INT 10H
; la usa el proc. DESP
video       proc
                        PUSH    AX
                        mov     ah,14
                        int     10h
                        POP     AX
                        ret
video                   endp


; Escribe UN carac. ubicado en  AL en video, usando INT 21H func. 02
chrout      proc
                        push    ax
                        mov     dl,al
                        mov     ah,02h
                        int     21h
                        pop     ax
                        ret
chrout                  endp

VIDEO_WORD      PROC
                        PUSH    AX
                        PUSH    BX
                        PUSH    CX
                        PUSH    DX
                        mov     BX,2
otronumv:               push    ax
                        push    ax
                        mov     dx,2
                        and     al,0f0h
; se analiza el byte bajo nibble x nibble
                        dec     dx
                        mov     cl,4
                        ror     al,cl
comparav:               cmp     al,0ah
                        jae     letrav
                        add     al,30h
```

```
                call    video
                jmp     nibblev
letrav:         add     al,37h
                call    video
nibblev:        cmp     dx,0h
                jz      ultimov
                pop     ax
                and     al,0fh
                dec     dx
                jmp     comparav
ultimov:        pop     ax
                mov     al,ah ;se analiza el byte alto usando al
                dec     bx
                jnz     otronumv
                POP     DX
                POP     CX
                POP     BX
                POP     AX
                ret
VIDEO_WORD      ENDP

code            ends
                end     MAIN
```

# BIBLIOGRAFIA

| | |
|---|---|
| **Advanced MS DOS** | Duncan, Ray<br>Microsoft Press<br>U.S.A. 1986 |
| **Assembly Language Book for the IBM PC** | Norton, Peter<br>Ed. Brady<br>U.S.A. 1986 |
| **Desinging with the 8088 Microprocesor** | Zarrella, John<br>Ed. Microcomputer Application<br>Suisun City California U.S.A. 1984 |
| **IBM PC Assambler Language** | Tabler, Donna<br>Ed. Willey Press Book<br>U.S.A. 1985 |
| **IBM Personal System / 1 Hardware Interface Technical Reference** | IBM<br>U.S.A. 1992 |
| **Inside de IBM PC** | Norton, Peter<br>Ed. Brady<br>U.S.A. 1986 |

**Memory Component Handbook**

Intel

U.S.A. 1988

**Microprocessor and Peripheral Handbook**

Intel Corporation, Vol. 1

U.S.A. 1988

**Microsoft Macro Assembler for the MS DOS Operating System User's Guide**

Ed. Microsoft Corporation

U.S.A. 1985

**Peripheral Desing Handbook**

Intel

U.S.A. 1981

**Programmer's Guide to the IBM PC**

Norton, Peter

Microsoft Press

U.S.A. 1985

**Programmer's Problem Solver for the IBM PC, XT & AT** — Jourdain, Robert. Ed. Brady. U.S.A. 1986

**The TTL Data Book for Desing Engineers** — Ed. Texas Instuments. U.S.A. 1976

**400 Ideas for Desing Volume IV** — Editado por Grossman, Morris. Hayden Book Company, Inc. U.S.A. 1988

**8088 Assembler Language Programming the IBM PC** — Ed. Howard W. Sams. U.S.A. 1983