UNIVERSIDAD NACIONAL AUTONOMA DE MEXICO

FACULTAD DE CIENCIAS

DESARROLLO DE UN SISTEMA BASICO DE ENTRADA SALIDA PARA EL MICROPROCESADOR 80186

TESIS

QUE PARA OBTENER EL TITULO DE:

MATEMATICO

PRESENTA:

RICARDO BARRON FERNANDEZ

MEXICO, D. F. 1985

UNAM – Dirección General de Bibliotecas Tesis Digitales Restricciones de uso

C O N T E N I D O

Agradecimientos

Quiero expresar mi agradecimiento al M.en I. Miguel Lindig, director de esta Tesis, por sus valiosos consejos, por su colaboración y atenciones recibidas para la elaboración de la misma.

A la maestra Guadalupe Ibargüengoitia, asesora de este trabajo, por su amabilidad y orientación.

A todas las personas de la sección de Graduados de U.P.I.I.C.S.A. por sus atenciones y colaboración desinteresada.

A Inteligencia Artificial, S.A., por permitir el uso de sus instalaciones durante la elaboración de este trabajo.

A los miembros del jurado examinador por su gentileza y amabilidad.

Y a todas las personas que directa o indirectamente contribuyeron a la realización de este trabajo.

I. INTRODUCCION.

Pensando en solucionar el problema de los servicios de cómputo para los estudiantes, en UPIICSA se estableció en 1982 un proyecto para desarrollar una microcomputadora en base al microprocesador 80186 de Intel.

Uno de los objetivos de diseño de la máquina es la de ser compatible a nivel Software con la IBM-PC.

En el presente trabajo se hará un análisis del Sistema Básico de Entrada/ Salida (BIOS) de la microcomputadora en cuestión y se presentará una versión operativa del mismo.

Se describirá el aspecto funcional de las distintas partes de la microcomputadora pero sin entrar en detalles eléctricos. Esto permitirá tener una visión de lo que ocurre en el interior de la microcomputadora que es el ambiente en el que se desarrolla el BIOS.

Para que una microcomputadora pueda operar, no basta que tenga todas sus componentes físicas ensambladas correctamente; es necesario que existan programas que les den animación a estos dispositivos, que los sincronicen y que los hagan funcionar como un todo, permitiéndole al usuario aprovechar las capacidades de la microcomputadora mediante comandos de uso sencillo. Este conjunto de programas lo conforman el BIOS y el Sistema Operativo.

El BIOS está integrado fundamentalmente por programas que controlan el funcionamiento de los dispositivos periféricos, con sus direcciones físicas asociadas, realizando su inicialización, verificación y ejecución de comandos de alto nivel provenientes del Sistema Operativo.

Así, el BIOS proporciona al Sistema Operativo una interfase que le permite dirigir los dispositivos, la cual es independiente de direcciones físicas y de señales de sincronización.

El principal objetivo del Sistema Operativo es el de proporcionar un sistema de archivos y un medio ambiente de ejecución para los programas de aplicación. Esto lo consigue basándose en los servicios que le proporciona el BIOS.

Como puede observarse, la programación del BIOS está estrechamente ligada con las características del Hardware, por lo que es de esperarse que BIOS de microcomputadoras distintas sean diferentes.

Esto es cierto en cuanto a la configuración de los programas, pero por razones de compatibilidad, al menos es nuestro caso, de lo que se trata es de emular las funciones del BIOS de la IBM-PC; ésto significa que los Sistemas Operativos que corren en la IBM-PC correrán en nuestra microcomputadora sin ningún problema, obteniendo resultados similares.

Otro detalle importante en cuanto a programación del BIOS es que, de hecho, es el único módulo de Software que se entiende directamente con el Hardware, todos los demás, Sistema Operativo y Módulos de Aplicación, ven a los dispositivos más como entes lógicos que físicos.

Podemos decir que el correcto funcionamiento de la microcomputadora depende del BIOS, de la adecuada interacción de éste con el Hardware y con el Sistema Operativo.

En su momento se hará una descripción de la estructura del BIOS y de algunas de sus rutinas, tratando de construir un panorama que ilustre, en base a las operaciones básicas de una microcomputadora su comportamiento exterior.

II. CARACTERISTICAS DE UNA MICROCOMPUTADORA.

Por microcomputadora se entiende, para fines de esta tesis, una computadora cuyo CPU está implementado en un sólo dispositivo.

Desde su aparición en 1963 con la PDP-5, las microcomputadoras han ido desarrollando características que las sitúan cada vez más en un plano de igual a igual con las minicomputadoras y computadoras de tamaño mediano, con bastantes posibilidades de superarlas (14.1)(2.1).

(Nota: los números entre paréntesis indican referencias bibliográficas).

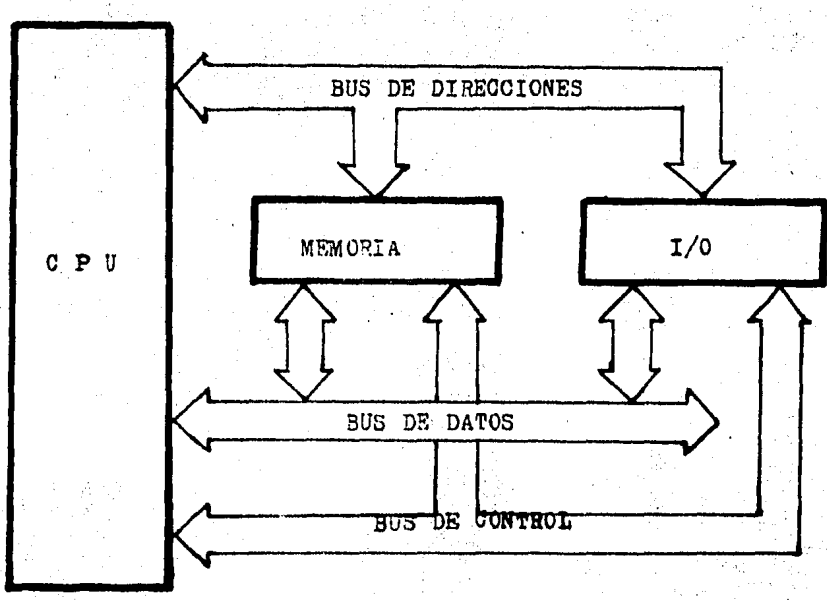

Fig. 1 Diagrama de bloques simplificado de una microcomputadora.

El fundamento tecnológico en el que se basan las microcomputadoras, son los circuitos integrados y en particular, en los de alta escala de integración (LSI). Estos permiten condensar en un pequeño espacio circuitos lógicos de un gran número de compuertas. De hecho, permiten la fabricación de toda una computadora en un solo dispositivo (sin contar con los dispositivos de Entrada/Salida).

Las microcomputadoras no sólo compiten con las computadoras medianas en el terreno propio de éstas, sino además abordan otros campos donde las computadoras grandes por su tamaño y costo no pueden entrar. Un ejemplo de ésto son las aplicaciones industriales y de control donde frecuentemente se usaban circuitos lógicos de propósito - específico y que ahora, con el bajo costo y disponibilidad de los microprocesadores, se pueden construir microcomputadoras que desempeñen el papel del Sistema digital de -- propósito específico, reduciendo costos y aumentando eficiencia y versatilidad.

III. EL MICROPROCESADOR 80186.

El microprocesador 80186 de Intel es un microprocesador de 16 bits altamente integrado; combina en un sólo dispositivo varias componentes que usualmente se hallan separadas. Esto permite que las microcomputadoras diseñadas a partir de él cuenten con un mínimo de componentes individuales. La comunicación del 80186 con los otros elementos, se lleva a cabo a través de sus "pins". Cada "pin" puede desempeñar una o más funciones de comunicación (3.3). Su CPU está integrado por la Unidad de Ejecución y la Interfase del BUS. En sus instrucciones de máquina, es totalmente compatible con el microprocesador 8086 y 8088 (2.2), ofreciendo diez instrucciones adicionales, algunas de alto nivel.

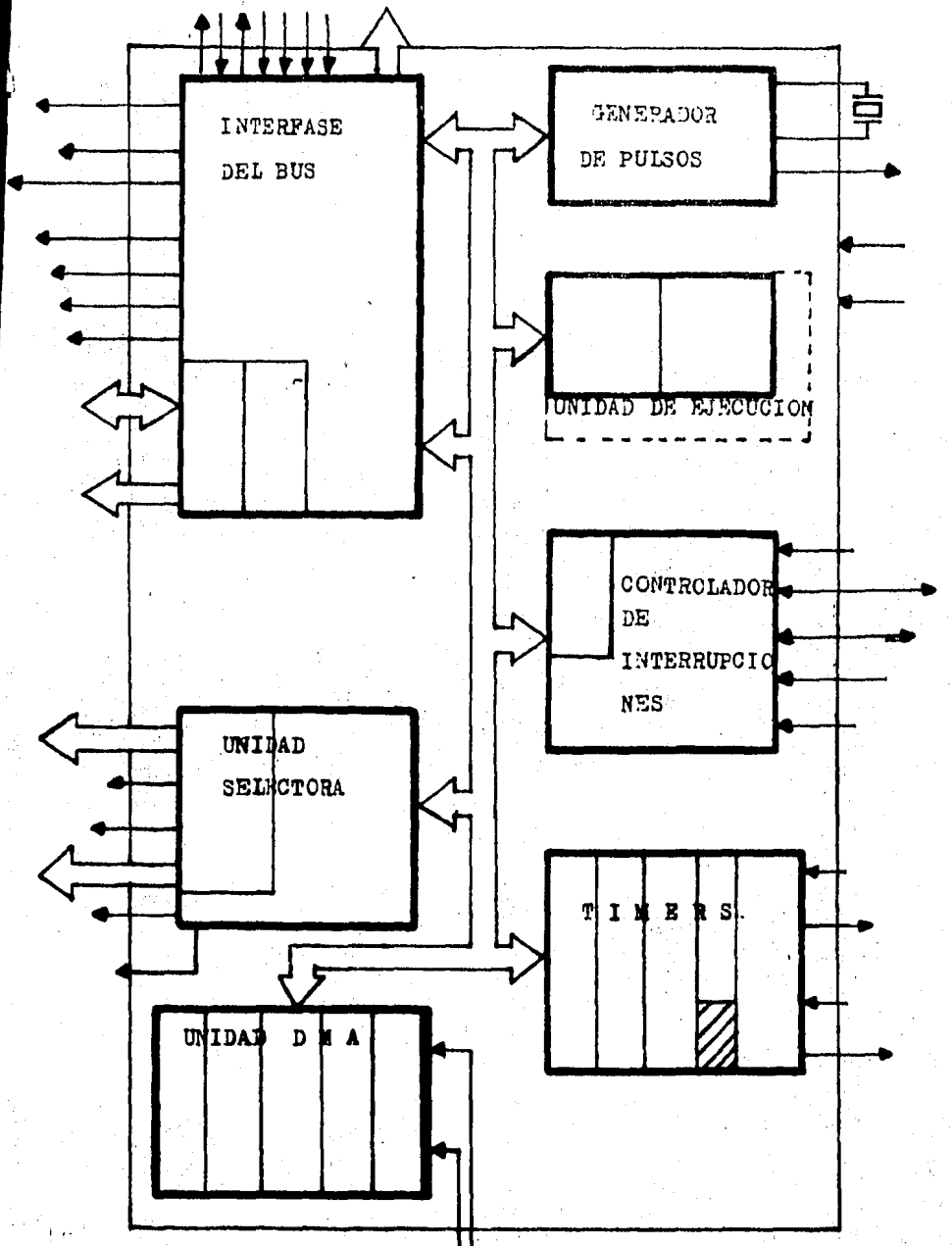

Fig. 2 El microprocesador 80186

GENERADOR DE PULSOS.

El generador de pulsos le proporciona al 80186 la señal de reloj fundamental (ciclos de máquina), en lo que se basa gran parte de su actividad.

El generador de pulsos deriva su frecuencia fundamental de una referencia externa (oscilador o cristal) del doble de la frecuencia de operación, es decir, para que el generador de pulsos oscile a una frecuencia de 6MHz, el oscilador externo debe oscilar a 12MHz (5).

Las frecuencias más usuales a las que oscila el 80186 son 8 y 6MHz. La ejecución de las instrucciones está sincronizada por los ciclos de máquina y puede medirse su duración de acuerdo al número de ciclos de máquina que ocupa; por ejemplo, una suma típica en el 80186 ocupa 3 ciclos y, si el micro oscila a 8MHz, se pueden efectuar aproximadamente 2.6 millones de sumas en un segundo.

El 80186 establece y controla la comunicación con los dispositivos de la microcomputadora externos a él, incluyendo los módulos de memoria, mediante la Unidad Selectora (Chip Select-Unit) y la Interfase del BUS (BUS Interfase Unit). Además, mediante la unidad DMA permite comunicación directa, sin pasar por el CPU, entre cualesquiera de los dispositivos externos, incluyendo los módulos de memoria.

UNIDAD SELECTORA.

Para poder Leer/Escribir a Memoria o a dispositivos periféricos, se les debe elegir antes mediante una señal de direccionamiento. La encargada de manejar ésta señal es la Unidad Selectora.

Seis líneas de salida son usadas para el direccionamiento de Memoria y siete para dispositivos. Las líneas selectoras de Memoria están divididas en tres grupos para direccionar de manera separada las áreas típicas de un sistema 80186; memoria superior para ROM, memoria baja para vectores de interrupción y memoria media para programas de aplicación. La dimensión de cada una de esas áreas, así como su dirección inicial, es programable.

Cada una de las siete líneas de selección de periféricos direcciona un área de 128 bytes contigues a partir de una dirección base programable, que puede estar localizada en Memoria o en espacio de Entrada/Salida.

INTERFASE DEL BUS.

La interfase de BUS se conecta directamente a - un BUS de datos externo, que se encarga de comunicar físi camente al CPU con todos los dispositivos de la microcomputadora para el intercambio de datos. En particular lo - conecta con los módulos de Memoria que es de donde se car gan las instrucciones y datos al CPU para ejecutarse. Nor malmente, la mayor parte del tiempo el BUS se encuentra - conduciendo de Memoria á o del CPU.

La interfase del BUS es la encargada de atender los requerimentos de uso de los dispositivos externos para enviar o recibir datos de otro dispositivo o de la Memoria; cuando se presenta un requerimento de éste tipo, la interfase del BUS, una vez que termina la transferencia, le cede el control del BUS al dispositivo que lo requirió Una vez que el dispositivo terminó de ocupar el BUS, su - control es devuelto al CPU.

El CPU también puede perder el control del BUS si la unidad DMA es requerida para hacer una transferencia. Después que el CPU termina de hacer su transferencia , si es que estaba haciendo alguna, la unidad DMA se apropia del BUS y pasa a ocupar el lugar del CPU, de modo que, si hay un requerimento externo del BUS, perderá el control sobre éste como lo hubiera perdido el CPU si lo tuviera.

En el 80186 la carga de la instrucción y su ejecución son realizadas por unidades separadas de funcionamiento independiente, la Interfase del BUS y la Unidad de Ejecución, respectivamente.

Cuando un programa se está ejecutando, el código de sus instrucciones es traído de Memoria y puesto en una cola de seis bytes por la Interfase del BUS. Cuando la ejecución requiere otra instrucción, la toma de la cola y no tiene que esperar a que sea traída desde Memoria. Esto incrementa la velocidad de ejecución, por el esquema de procesamiento paralelo en base a dos procesadores dedicados.

EL CPU.

El CPU tiene cuatro registros generales de 16 - bits (AX,BX,CX,DX), usados como operandos en operaciones aritméticas en las cuales se pueden usar sus ocho bits altos o bajos en forma separada. También contiene: cuatro - registros apuntadores de 16 bits (SI,DI,BP,SP), que pueden usarse en operaciones aritméticas o como apuntadores para accesar variables, cuatro registros de segmento (CS, DS,SS,ES), que permiten particionar la Memoria y construir programas modulares, un apuntador de 16 bits y un registro de estado también de 16 bits.

Para obtener una instrucción o un operando, el direccionamiento de Memoria es generado a partir de un registro de segmento y un desplazamiento.

El registro de segmento es desplazado cuatro - bits a la izquierda y sumado al desplazamiento, obteniéndose una dirección de 20 bits con la que se puede direccionar hasta un megabyte.

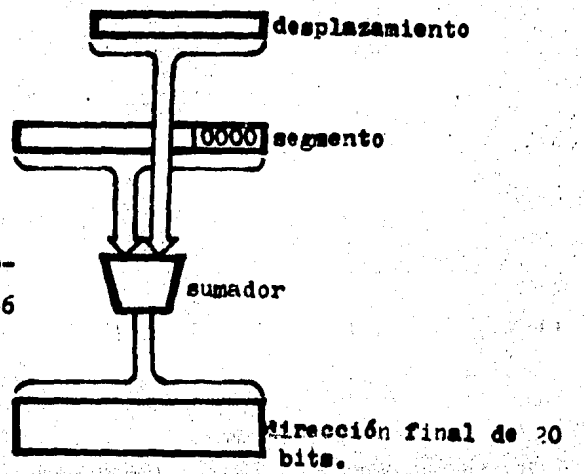

Fig. 3
Proceso de direccionamiento en el 80186

CONTADORES.

El 80186 incluye una Unidad Timer que contiene tres contadores de 16 bits. Dos de los cuales pueden ser usador para contar eventos, para proporcionar formas de onda derivadas del CPU o de un reloj externo, o para interrumpir al CPU después de un número específico de "eventos". El tercer contador cuenta sólo pulsos del CPU y puede ser usado para: interrumpir al CPU después de un número programable de pulsos, alimentar pulsos a cualquiera de los otros contadores o a ambos, hacer un requerimento de transferencia a la Unidad DMA después de ciertos pulsos de CPU.

CONTROLADOR DE INTERRUPCIONES.

Este controlador funciona como árbitro entre los requerimentos de atención tanto de interrupciones internas como externas. Permite, entre otras cosas, establecer prioridades entre las interrupciones e inhibirlas individualmente desde programa.

El juego de instrucciones del 80186 proporciona la facilidad de interrumpir al 80186 desde programa mediante la instrucción INT XX (donde XX es un número que sirve para localizar la rutina de servicio asociada). Esta instrucción se comporta esencialmente de la misma manera que las interrupciones por Hardware y permite inclusive desde programa accesar las interrupciones que normalmente se habilitan por Hardware.

IV. CONFIGURACION DE NUESTRA MICROCOMPUTADORA.

La configuración de nuestra microcomputadora está basada en dos microprocesadores, el 80186 y el 8031, - los dos de Intel. El primero es el microprocesador maestro del sistema y el segundo se encarga de atender exclusivamente el video y el teclado. Ambos microprocesadores se - comunican mediante un Puerto con un protocolo de comunicación asociado.

El microprocesador 8031 posee dos memorias, una para almacenar exclusivamente instrucciones y otra para - los datos con capacidad de hasta 64K bytes (16). En su Memoria de instrucciones residen los programas que controlan el video y al teclado, y en su memoria de datos (video-memoria) se lee o se escribe la información que mediante un manejo específico, se convierte en caracteres o figuras en la pantalla del video.

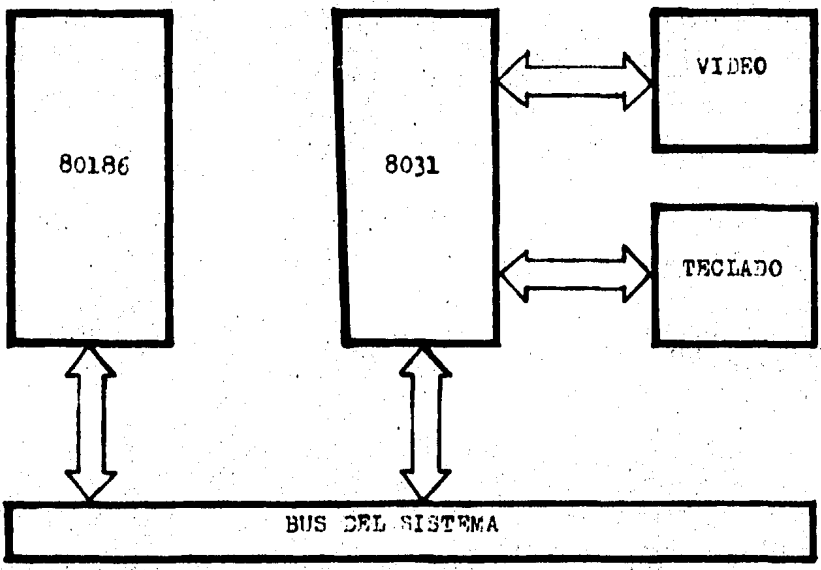

Fig. 4 Cooperación entre el 80186 y el 8031

Por la gran capacidad de integración del 80186, l número de dispositivos individuales de la microcomputa ora es reducido, puesto que, tanto el controlador de inte rupciones, los Timers y el controlador de DMA residen -- entro del 80186.

Se cuenta además con un controlador de diskette 272A con capacidad para manejar hasta cuatro unidades de iskette (Drivers) con dos cabezas cada uno.

V. CARACTERISTICAS DE UN SISTEMA BASICO DE ENTRADA/SALIDA (BIOS).

El BIOS es un conjunto de tablas y rutinas generalmente residentes en ROM que se encarga de proporcionar a los programas de aplicación y fundamentalmente, al Sistema Operativo, una interfase mediante la cual puedan dirigir el funcionamiento de los dispositivos tales como: pantallas, impresoras, graficadoras, etc., mediante comandos uniformemente estructurados.

La primer tarea del BIOS consiste en verificar e inicializar los dispositivos al encenderse la máquina, esto lo hace independientemente del Sistema Operativo con el que se vaya a trabajar.

El principal objetivo del BIOS consiste en establecer un marco independiente de direcciones físicas para el Software de aplicación. Esta función del BIOS permite desarrollar programas que se pueden ejecutar en toda una familia de microcomputadoras siempre y cuando cada miembro sea compatible en su juego de instrucciones de máquina y su BIOS soporte el mismo protocolo (10.9).

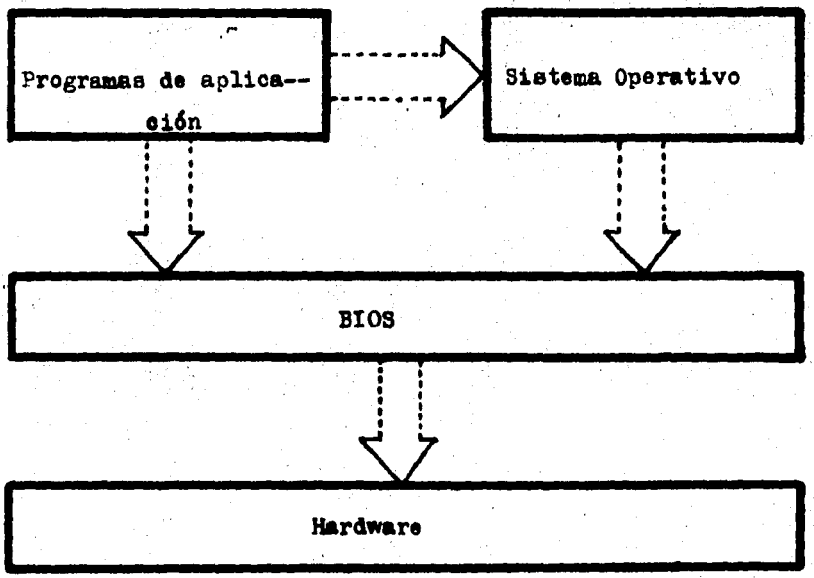

Fig. 5 El BIOS, interfase entre Software y Hardware

Es conveniente que el BIOS sea el único módulo de Software que interactúe directamente con el Hardware. Las razones de ésto son dos:

1. Portabilidad.

La comunicación con los dispositivos se realiza a través de Puertos. Generalmente, un dispositivo tiene asociado uno o más Puertos y a cada Puerto le corresponde una dirección fija, de manera que, si algún programa maneja directamente un dispositivo, por ejemplo la impresora, por fuerza en algún momento deberá hacer referencia a la dirección fija asociada con algún Puerto de la impresora. Lo que implica que, si queremos correr éste programa en otra microcomputadora, ésta debe manejar el mismo Puerto - en la misma dirección y ésto constituye una restricción bastante fuerte.

2. Eficiencia.

El algoritmo para manejar un dispositivo directamente puede ser grande y complejo, además, para implementarlo se debe conocer a cierto nivel el funcionamiento interno del dispositivo, por lo que es recomendable no gastar esfuerzo en una rutina que, por otra parte, seguramente se puede implementar usando el BIOS.

En conclusión, a partir del BIOS, el Sistema Operativo y los módulos de aplicación ven a los dispositivos como entes lógicos con características que no dependen de su implementación física.

ESTRUCTURA Y ACCESO AL BIOS.

EL BIOS está formado por:

- Rutinas de inicialización y prueba de los dispositivos.
- Area de datos ..... donde residen los parámetros de funcionamiento de los dispositivos.
- Vectores de Interrupción ..... que indican el comienzo de las rutinas de servicio.
- Rutinas de servicio ..... que dirigen el funcionamiento de cada acción específica del BIOS.

Las rutinas de inicialización y prueba tienen como primer objetivo, detectar alguna falla en los dispositivos que pueda repercutir posteriormente en el funcionamiento adecuado de la microcomputadora. Este tipo de pruebas se hacen alimentando a los dispositivos con datos elegidos de tal manera que permitan conocer de antemano en qué forma ha de responder un dispositivo normal. Si el comportamiento del dispositivo no es el esperado, entonces se activa una rutina de error.

Las rutinas de prueba pueden ser tan complicadas como se desee, ésto dependerá de lo que se considere importante checar y lo que nó y con qué grado de sofisticación.

Una vez que se ha verificado el buen funcionamiento de un dispositivo, si es necesario se procede a inicializarlo, es decir, a dejarlo en un estado a partir -

del cual pueda operar normalmente en lo sucesivo (10.9).

Finalmente, una vez que todos los dispositivos que lo requieran han sido probados e inicializados, se ejecuta un procedimiento (BOOT STRAP) mediante el cual se carga desde disco al Sistema Operativo en la memoria principal de la microcomputadora.

Las rutinas de inicialización y prueba pueden empezar a ejecutarse a partir del encendido de la máquina (RESET de encendido) o bien, si la máquina ya está encendida, presionando una combinación adecuada de teclas (RESET de teclado).

El acceso a los servicios del BIOS es a través de interrupciones "blandas" cuyos parámetros residen en los registros del CPU.

Por interrupción "blanda" debe entenderse una interrupción generada por Software. A nivel lenguaje ensamblador, éstas interrupciones toman la forma: INT XX -- donde XX es un número que sirve para obtener la dirección de inicio de la rutina de servicio asociada a la interrupción (2.3).

Para obtener la dirección de inicio de la rutina de servicio, se multiplica XX por cuatro, éste número se toma como la dirección de dos palabras de memoria consecu

tivas (a éste par de palabras se les conoce como Vector de Interrupción), de las que se toma al segmento y el desplazamiento de la dirección buscada (5).

Puesto que el número XX en la instrucción INT XX ocupa un byte, se pueden definir hasta 256 vectores de interrupción, lo que implica que el primer K byte de memoria esté potencialmente ocupado por vectores de interrupción.

Existen algunos vectores de interrupción que pueden ser accesados por Hardware. Esto se consigue asignándole a las líneas de interrupción usadas por los dispositivos para interrumpir al 80186, un número XX que desempeña el mismo papel que en la instrucción INT XX. Esto se hace con el fin de que en cuanto un dispositivo interrumpa, se tenga un rápido acceso a la rutina de servicio que atenderá dicha interrupción. La asignación de los vectores es fija y no puede ser modificada por Software.

El acceso de las rutinas de servicio mediante vectores de interrupción permite a los programas de aplicación definir sus propias rutinas de servicio, interceptar las que existen o sustituirlas. En el caso de la sustitución de una que ya existe, ésto se lleva a cabo sustituyendo un vector de interrupción por otro que apunte a la nueva rutina de servicio (10.5).

VI. INTERACCION DEL BIOS CON LOS DISPOSITIVOS PERIFERICOS.

En términos generales, un dispositivo periférico es para la microcomputadora el órgano de algún sentido. Es decir, el dispositivo periférico le permite actuar sobre el medio exterior o percibir alguna acción del mismo.

La comunicación del CPU con los periféricos se lleva a cabo a través de Puertos, que son direcciones que bien pueden corresponder a localidades de memoria o nó (14.6).

En el caso del 80186, existen instrucciones específicas para manejar los Puertos. Estas son IN dir (donde dir es la direcoión del Puerto de donde se Lee/Escribe datos), para leer un Puerto y OUT para escribir en él; generalmente se lee para enviarle un comando o los parámetros de inicialización. El control de los periféricos se efectúa mediante lecturas y escrituras a los Puertos correspondientes. La máquina puede controlar hasta 65535 - Puertos.

La interacción entre el BIOS y un dispositivo dado se establece a solicitud de un programa que requiere el uso del dispositivo mediante una interrupción blanda, o requerida por el dispositivo mismo mediante una interrupción Hardware con el objetivo de avisarle al 80186 que ha entrado en un estado que requiere atención.

En cada caso la atención a la interrupción, lo efectúa una rutina de servicio. Esta rutina de servicio forma parte del BIOS, y está direccionada por el vector de interrupción asociado al dispositivo. Todas las localidades de memoria usadas por las rutinas de servicio como variables propias, se encuentran en un área llamada BIOS DATA (400H a 482H).

La variedad de periféricos que se pueden conectar a un microprocesador depende de todo aquello que se pueda controlar mediante señales digitales, como por ejemplo: la operación de un rayo láser, el movimiento de un brazo mecánico, el volante de un carro, etc. Aunque en la actualidad y debido al tipo de aplicaciones dominantes los más usados son: diskette, video, teclado, impresora, etc. En lo que sigue se hará una descripción funcional de algunos de ellos y de su rutina de servicio asociada.

(Nota: la descripción de las rutinas de servicio no es un diagrama de bloques fiel de las rutinas del listado, su objetivo es dar una idea de su funcionamiento).

6.1 EL TECLADO.

El principal medio de comunicación entre el usuario y la microcomputadora lo constituye el teclado. Cada una de las teclas tiene asociado un valor relacionado con su posición en el teclado (Código de Rastreo) y otro valor que indica al Sistema Operativo una acción específica a efectuar (valor ASCII)(10.9), éste valor está representado nemotécnicamente por la figura en la parte superior de la tecla y es asignado por Software. Por ejemplo, la tecla ‹RETURN› en el teclado de la IBM-PC tiene un Código de Rastreo de 28 y un valor ASCII de 13 . Su valor ASCII le indica al Sistema Operativo que hay que hacer un retorno de carro sobre la pantalla, es decir, posicionar al cursor en el principio de la línea corriente.

Cada tecla tiene asociado un sólo Código de Rastreo y ninguno, uno o más valores ASCII, ésto depende si la tecla es presionada sola o simultáneamente con alguna otra tecla que altere su valor ASCII. Las teclas que pueden alterar el valor ASCII son: ‹CONTROL›, ‹SHIFT›, ‹ALTERNATE›, ‹UPER CASE›. En nuestro teclado, éstas teclas no generan Código de Rastreo, únicamente un bit que las identifica (13).

Cuando se presiona una tecla, los circuitos del teclado generan una señal (STROBE) que interrumpe al microprocesador que atiende al teclado, indicándole que hay

un dato listo para ser enviado; éste dato está compuesto del Código de Rastreo de la tecla que se presionó y de información extra que indica que una tecla modificadora fué presionada simultáneamente. Una vez que el microprocesador recibe la interrupción y lee el dato, genera una señal (Acknowledge) que avisa al teclado que el dato ha sido léído y que puede presentar otro dato a la salida.

Por razones de compatibilidad, nuestro teclado emula al teclado de la IBM-PC, ésto quiere decir que, una vez que se ha recibido el dato del teclado, se le debe modificar para generar el Código de Rastreo y el valor ASCII que en un teclado IBM-PC hubiera generado la tecla correspondiente (1).

El teclado tiene asociadas dos rutinas de servicio: CONSOLE_INT (INT 0DH), KEYBOARD_IO (INT 16H).

La primera se encarga de atender la interrupción Hardware del teclado, almacenando los caracteres que recibe en un buffer. La segunda usando el buffer que carga la primera, regresa caracteres al usuario o avisa que el buffer está vacío. (En realidad, en nuestro caso el que recibe la interrupción del teclado es el 8031 y después de convertir la información recibida a un formato IBM, interrumpe al 80186).

CONSOLE INT.

Se encarga de recibir el Código de Rastreo y el código ASCII de la tecla o combinación de teclas presionada, una vez recibidos los almacena en el buffer del teclado (organizado como una cola circular), que reside en BIOS DATA. Otra de sus funciones es checar si los valores recibidos tienen un significado especial, por ejemplo, RESET o BREAK; si éste es el caso, ejecuta alguna acción específica.

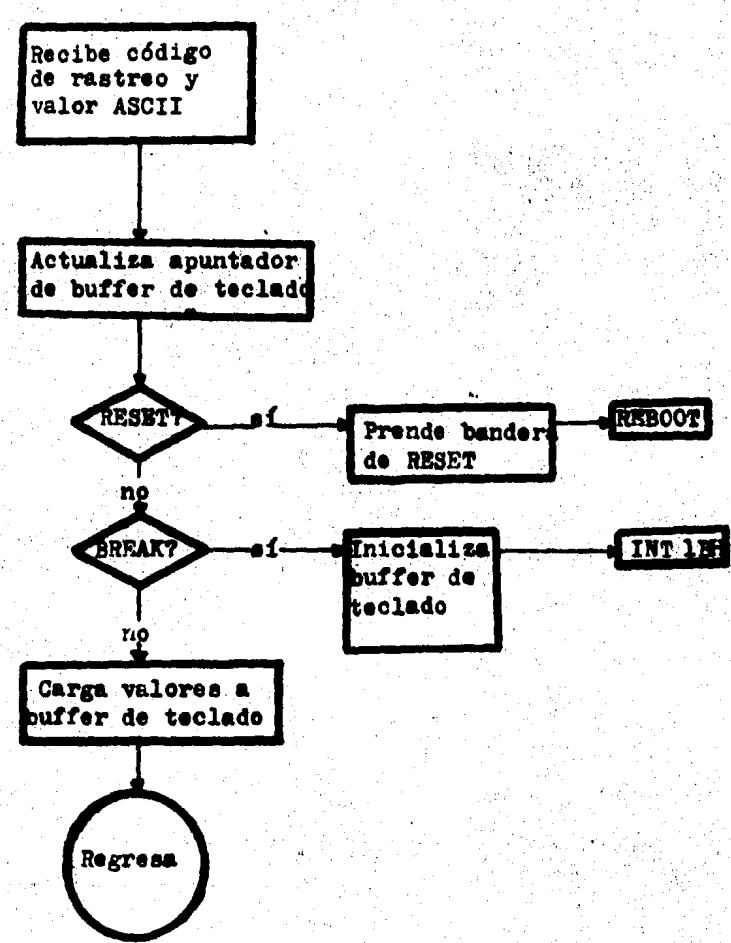

Fig. 6 CONSOLE INT. Diagrama

KEYBOARD_IO.

Esta rutina de servicio tiene tres modalidades, la opción se indica mediante el registro AH.

AH = 0 Espera al siguiente caracter del teclado y una vez que llega, regresa con código ASCII en AL, Código de Rastreo en AH.

AH = 1 Substrae del buffer de teclado el primer caracter de la cola y lo pone en AX, apaga la bandera ZF de la palabra de estado, si está vacío el buffer de teclado, prende ZF.

AH = 2 Carga el registro de las teclas de control presionadas en AL.

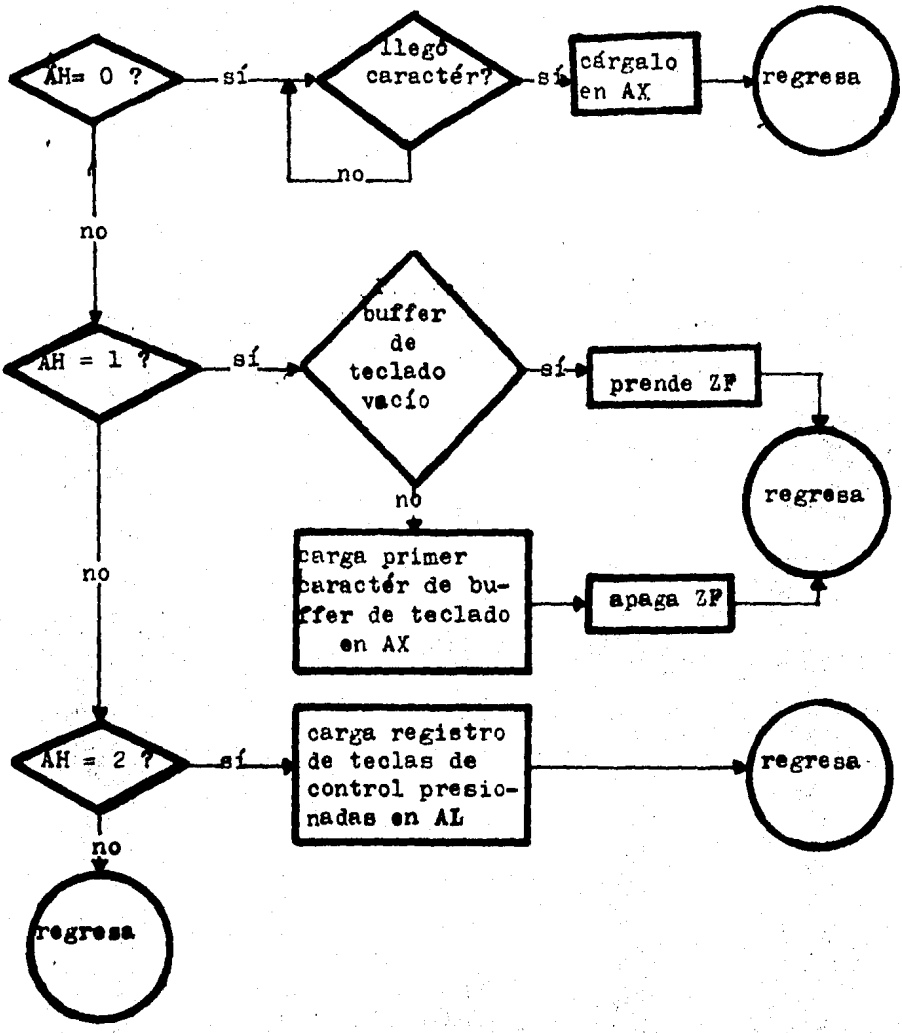

Fig. 7 KEYBOARD_IO. Diagrama

6.2 VIDEO.

Físicamente lo comstituye una pantalla similar a una televisión y un convertidor digital-analógico que convierte el patrón de bits de los caracteres en señales de video que generan en la pantalla la forma del caracter En nuestro caso, para representar un caractér se usan 16 bytes, lo que equivale a contar con una matriz de 16 X 8 bits para cada caractér implementado.

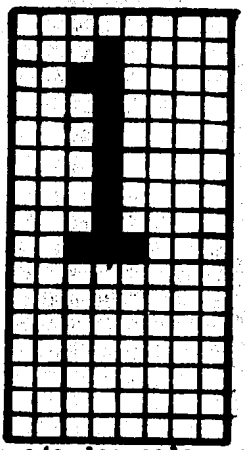

Fig. 8 Matriz del número uno

El video maneja dos modos: alfanumérico y gráfico. En el primero, para que un caractér aparezca en la pantalla, es necesario colocar su correspondiente valor ASCII en la primera parte de una memoria RAM llamada video-memoria; éste valor ASCII se utiliza para direccionar una memoria ROM llamada generador de caracteres, de donde se toma el patrón de bits correspondiente al caractér (10.8).

En la segunda parte de la video memoria, se deposita el atributo con que se desea aparezca el caractér en la pantalla, éste atributo puede ser: subrayado, video inverso, centelleo o combinaciones de ellos.

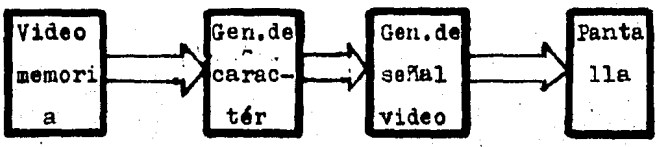

Fig. 9 Modo alfanumérico

Hay una correspondencia uno a uno entre las posiciones en la video memoria y la pantalla, de modo que para que aparezca un caractér o punto en la pantalla en el lugar que se desee, basta con conocer el lugar correspondiente en la video memoria (1).

En el modo gráfico no se utiliza el generador de caracteres, el patrón de bits de los caracteres o figuras a desplegar en la pantalla se toman directamente de la video memoria. Esto permite más versatilidad, ya que se pueden manejar caracteres y figuras de cualquier tipo juntos.

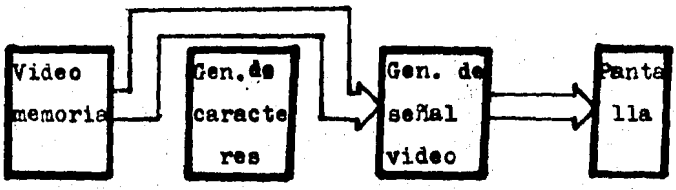

Fig. 10 Modo gráfico

Tanto en el modo gráfico como en el alfanumérico, la video memoria es leída ininterrumpidamente para obtener ya sea un código ASCII o patrones arbitrarios de bits; el único momento cuando se suspende ésta lectura es cuando se presenta una señal llamada RETRACE que indica que el haz de exploración de la pantalla terminó de imprimir una línea y se dirige a comenzar otra. Todas las lecturas y escrituras a la video memoria deben efectuarse durante el proceso indicado por RETRACE (1).

El video tiene una sola rutina de servicio VIDEO_IO (INT 10H).

Esta rutina de servicio soporta varias opciones que son especificadas a través del registro AH. La mayoría de las opciones son autoexplicativas (ver listado).

En nuestra configuración el microprocesador que tiene contacto directo con la video memoria es el 8031 - por lo que en algunas opciones libera de parte del trabajo al 80186. La comunicación entre el 80186 y el 8031 la llevan a cabo las subrutinas CONSOLE TX y CONSOLE RX.

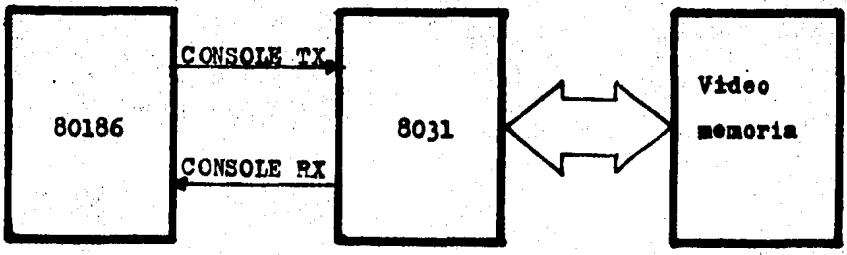

Fig. 11 Cooperación en el manejo del video

6.3 DISKETTE.

Nuestra microcomputadora cuenta con un controlador de diskette (FDC) 8272A capaz de manejar hasta cuatro unidades de diskette (Drivers); en el Drive se encuentran los circuitos que manejan la cabeza Lectora/Escritora, — que es la que en última instancia se encarga de leer o escribir sobre el diskette (11).

La comunicación entre el 80186 y el FDC se lleva a cabo a través de dos Puertos que corresponden a dos registros internos del FDC: el registro de estado y un registro de datos. En el registro de estado residen los bits que indican cuándo puede enviar o recibir datos el FDC y la dirección de transmisión de éstos datos, es decir, si van o vienen del 80186, etc.

En el registro de datos se reciben los comandos que ejecuta el FDC además de sus parámetros y, cuando finaliza la ejecución de un comando, se puede accesar a través del registro de datos la información que indica en — qué condiciones terminó la ejecución del comando.

La operación del FDC se desarrolla ejecutando los comandos que le indica el 80186. Este puede indicarle hasta 15 comandos distintos.

La ejecución de un comando está dividida en tres fases: fase de comando, fase de ejecución y fase de resultados.

Durante la fase de comando, el registro de estado indica que el FDC puede aceptar datos de comando. El comando ocupa un byte y puede requerir hasta 7 bytes como parámetros. El FDC una vez que ha recibido todos los parámetros que requiere el comando, inhibe la recepción de datos de comando e inicia la fase de ejecución.(4.6).

Durante la fase de ejecución, el FDC envía al Drive las señales adecuadas para que se lleven a cabo las acciones establecidas por el comando y sus parámetros.

Finalmente, la fase de resultados inicia una vez que ha terminado la ejecución del comando, en ese momento, el registro de estado indica que se pueden enviar datos del FDC al 80186; estos datos pueden ser hasta 7 bytes y es necesario leerlos si se quiere saber en qué condiciones terminó la ejecución del comando. Una vez que se han leído todos los datos de información correspondientes, el registro de estado indica que el FDC queda en disposición de aceptar otro comando. A continuación se muestran algunos de los comandos que puede ejecutar el 8272A.

BRASER (seek).

Mediante éste comando, la cabeza Lectora/Escritora se posiciona en la pista adecuada para, posteriormente, con un comando de lectura o escritura escribir o leer sobre él.

RECALIBRAR (recalibrate).

Dentro del FDC hay un registro que indica sobre qué pista se encuentra la cabeza Lectora/ Escritora; después que el FDC detecta un error de operación o después de un RESET, el contenido de éste registro y la cabeza Lectora/ Escritora no coinciden. Para que se pueda seguir operando normalmente, es necesario forzar ésta coincidencia y ésto es precisamente lo que hace éste comando, posiciona la cabeza encima de la pista cero y clarea el citado registro.

FORMATEA UNA PISTA (Format a Track).

Mediante éste comando se escribe información al diskette que posteriormente permitirá identificar cada sector de una pista en forma individual, ésta identificación es necesaria para leer o para escribir.

LEE IDENTIFICADOR (Read ID).

Permite obtener la posición actual de la cabeza Lectora/ Escritora.

LEE DATOS (Read data).

Lee uno o varios sectores dentro de la pista donde se halla la cabeza Lectora/ Escritora.

ESCRIBE DATOS (Write data).

Escribe uno o varios sectores dentro la pista donde se halla la cabeza Lectora/ Escritora.

El diskette tiene asociadas dos rutinas de servicio: DISKETTE_INT (INT OEH) y DISKETTE_IO (INT 13H).

La primera atiende a la interrupción generada por el controlador del diskette (FDC), por ejemplo, al concluir la ejecución de algún comando. La segunda permite al Sistema Operativo o programas de aplicación hacer uso del diskette.

DISKETTE_INT.

Unicamente prende la bandera de interrupción del SEEK_STATUS, localizado en el BIOS DATA.

DISKETTE_IO.

Esta rutina de servicio atiende los requerimentos de usar el diskette. Soporta varios comandos especificados en el registro AH. Su principal componente es la subrutina DISK_EXEC que es la que directamente se encarga de ejecutar los comandos.

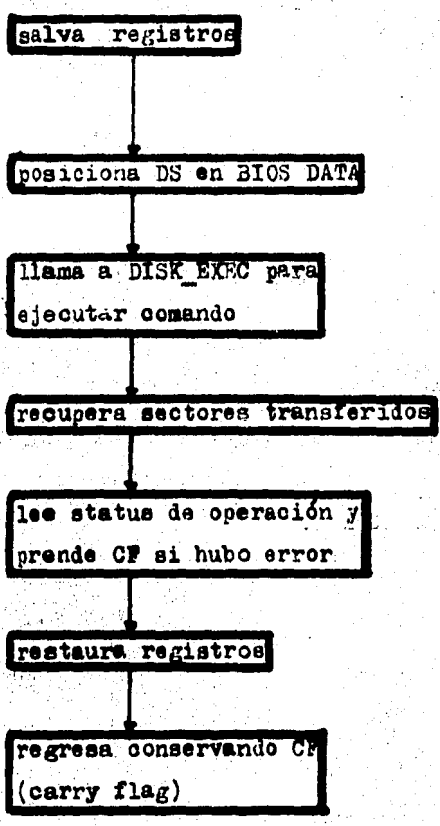

Fig. 12 DISKETTE_IO. Diagrama

VII. EL RELOJ DE TIEMPO REAL.

Nuestra microcomputadora tiene implementado un reloj de tiempo real basado en los Timers 0 y 2.

El Timer 2 funciona como un divisor de la frecuencia de oscilación del 80186 (en nuestro caso se está suponiendo que el 80186 está oscilando a 6MHZ) y el Timer 0 proporciona el cuantum de tiempo requerido al generar una interrupción cada determinado tiempo.

El funcionamiento de los Timers en el 80186 responde aproximadamente a lo siguiente:

Los Timers son contadores de pulsos, dichos pulsos los puede proporcionar el CPU, una fuente externa o el Timer 2. La cuenta se lleva en un registro especial dentro de cada Timer y se pone en ceros cuando empieza la operación o se alcanza una cuenta máxima que indica otro registro del Timer. La cuenta máxima es programable, como también lo es la capacidad de generar una interrupción una vez alcanzada la cuenta máxima.

La programación de los Timers se hace mediante su correspondiente palabra de control (5), en nuestro caso el momento preciso de hacer esto es durante la inicialización usando los parámetros que se hallan en TIMER TABLE.

El Timer 2 es programado para que genere un pulso cada 24 del CPU, esto da un total de 250 mil pulsos por segundo.

El Timer 0 es programado para que cuente los pulsos generados por el Timer 2 e interrumpa al 80186 cada 13733 de ellos, lo que implica que el Timer 0 interrumpe al 80186 a razón de 18.204 interrupciones/segundo.

Físicamente el reloj de tiempo real se halla representado por dos localidades de memoria en BIOS DATA, TIMER_HIGH y TIMER_LOW. La primera registra las horas y la segunda las interrupciones del Timer 0 (Nota: desde el momento en que TIMER_LOW vale cero hasta que alcanza su cuenta máxima $2^{16}-1$, transcurre una hora, $(2^{16}-1)/18.204 = 3600$).

El reloj de tiempo real tiene asociadas dos rutinas de servicio, TIMER0_INT (INT 1AH) y TIME_OF_DAY (INT 8). La primera atiende la interrupción generada por el Timer 0 y actualiza el reloj de tiempo real. La segunda la utiliza el Sistema Operativo o los programas de aplicación para leer o inicializar el reloj de tiempo real.

TIMERO_INT.

Es la rutina activada cada vez que el TIMERO interrumpe al 80186.

Actualiza al reloj de tiempo real, si TIMER_LOW alcanza su cuenta máxima, incrementa TIMER_HIGH.

Una vez terminada una operación del diskette, la cuenta del motor del FDC da un margen de tiempo antes de apagar el motor. De ésta manera, si dentro del margen se intenta otra operación, se encuentra al motor en marcha.

Finalmente, invoca una rutina de usuario a través de la interrupción ICH (Timer-Tick) , ésta rutina puede ser diseñada por el usuario para obtener un cuantum de tiempo más acorde a sus necesidades.

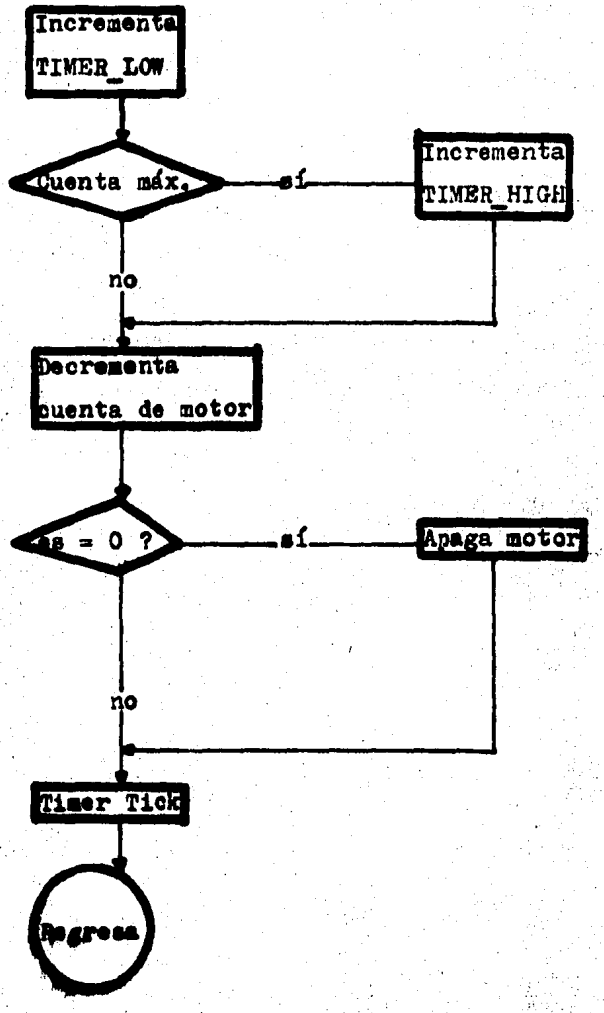

Fig. 13 TIMER0_INT. Diagrama

TIME_OF_DAY.

Usada por el Sistema Operativo para leer o inicializar el reloj de tiempo real.

Para inicializar se cargan los valores a cargar en TIMER_HIGH y TIMER_LOW en los registros CX y DX respectivamente.

Al leer se recibe el contenido de TIMER_HIGH, TIMER_LOW en CX y DX respectivamente.

VIII. INTERACCION DEL SISTEMA OPERATIVO CON EL BIOS.

Una vez que terminan las rutinas de inicialización y prueba sin novedad, se ejecuta el BOOT STRAP y a partir de ahí el Sistema Operativo toma el control de la microcomputadora (9.A).

Una de las primeras cosas que hace el Sistema Operativo es indagar con cuántos y de cuáles recursos de la microcomputadora puede disponer; por ejemplo: cuánta memoria de usuario hay disponible, con cuántos Drives de diskette se cuenta, etc. Toda ésta información se la proporciona el BIOS.

El trabajo del Sistema Operativo consiste fundamentalmente en proporcionar un sistema de archivos y un medio ambiente adecuado para los programas de aplicación. Para éste dispone de todos los servicios que puede obtener del BIOS (7).

El sistema de archivos permite manejar bloques de información llamados archivos (que bien pueden ser programas, datos, texto), mediante un sólo nombre designado por el usuario (nombre del archivo). Podemos mover un archivo de diskette a memoria o viceversa, cambiar de nombre a un archivo, crearlo o destruirlo. Todo ésto mediante un sólo comando (distinto en cada caso) y uno o varios parámetros.

Para hacer todo éste trabajo, el Sistema Operativo lo único que necesita hacer es leer o escribir en el diskette y gracias al BIOS, ésto lo puede hacer usando la interrupción 13H.

Para proporcionarles un medio ambiente de ejecución adecuado a los programas de aplicación, el Sistema Operativo define durante su autoinicialización varios vectores de interrupción y carga en Memoria, como parte del Sistema Operativo, las rutinas de servicio asociadas a éstos vectores de interrupción.

Los servicios relacionados con un ambiente de ejecución permiten, por ejemplo, a un programa en ejecución , ejecutar otro programa distinto, abrir o cerrar un archivo, escribir o leer sobre un archivo, etc. Externamente, desde teclado, un usuario puede abortar la ejecución de un programa; ésto es útil, por ejemplo, cuando se sospecha que el programa entró en un loop o está haciendo cosas totalmente indebidas. Además de todo ésto, el Sistema Operativo es el encargado de administrar la memoria -- donde se ejecutan los programas de aplicación (8).

La comunicación con el Sistema Operativo se puede establecer externamente desde teclado mediante comandos captados e interceptados por un módulo del Sistema Operativo específicamente encargado de ésto (CCP), o internamente durante la ejecución de un programa mediante interrupciones blandas definidas por el Sistema Operativo.

Las rutinas asociadas a los comandos pueden residir en Memoria o en diskette; generalmente, las rutinas de los comandos más usados están en memoria y los menos frecuentes, en diskette. De manera que, luego que el CCP interpreta un comando, busca la rutina asociada al comando, en Memoria; si no está, la busca en diskette y si no la encuentra , indica ERROR, pero si está, la carga en Memoria y la activa.

ESTRUCTURA DE MS-DOS.

Para ejemplificar un poco lo anterior, a continuación se muestra el esquema de uno de los Sistemas Operativos de microcomputadoras más populares, el MS-DOS.

Si se desea trabajar con MS-DOS, antes de dar el BOOT se le debe colocar en el Drive en donde actúa el BOOT. Una vez dadσ el BOOT, el MS-DOS es cargado en Memoria y activado; al área donde se carga el MS-DOS se le conoce como área del sistema.

MS-DOS consta de tres archivos:

- COMMAND.COM
- MSDOS.SYS
- IO.SYS

COMMAND.COM se encarga de interpretar los comandos emitidos por el usuario desde el teclado y MSDOS.SYS conjuntamente con IO.SYS controlan los recursos de la computadora entre los que destacan el espaδio en disco y la memoria central.

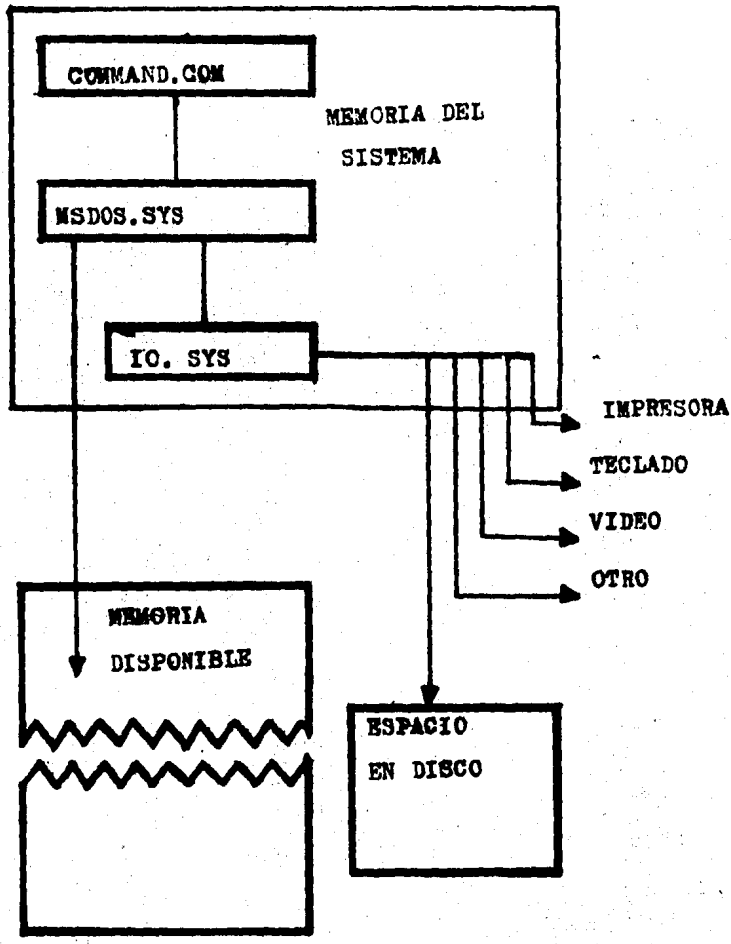

Fig. 14 Estructura del MS-DOS

APENDICE.

LISTADO DEL BIOS.

CONTENIDO.

```
title (bios para una microcomputadora basada en el 80186)
; las rutinas del bios son accesibles mediante interrupciones,
; lo que permite que se les pueda invocar sin conocer la direc-
; cion en que se encuentran.Es recomendable que no se intente--
; acsesarlas mediante las direccines que aparecen en el listado-
; porque estas pueden cambiar.
;-----------------------------------------------------------------


;-----------------------------------------------------------------
; direcciones de puertos y constantes globales
;-----------------------------------------------------------------
acs_reg     equ         0ffa4h
dc_main_status  equ     100h
ma_ctrl_word    equ     0ffcah
dc_cmd      equ         108h
art_ctrl    equ         182h
imer_ctrl   equ         0ff52h
nt_ctrl     equ         0ff32h
nt_poll     equ         0ff24h
nt1_ctrl    equ         0ff3ah
o:_reg      equ         0ff22h
ios_data    equ         40h
;-----------------------------------------------------------------
; area de los vectores de interrupcion                          :
;-----------------------------------------------------------------
bs0         segment     at 0
t_loc0      label       byte
            org         2*4
m:_ptr      label       word
            org         5*4
nt3_ptr     label       word
            org         8*4
nt_addr     label       word
nt_ptr      label       dword
            org         10h*4
:deo_int    label       word
            org         1dh*4
arm_ptr     label       dword
            org         18h*4
as:c_ptr    label       word
            org         1eh*4
:a_point    label       dword
            org         1fh*4
t_ptr       label       dword
            org         40h*4
o_rom_init  dw          ?
o_rom_seg   dw          ?
            org         400h
ata_area    label       byte
ata_word    label       word
            org         7c00h
_loc        label       far
            ends
;-----------------------------------------------------------------
; area de stack usada solo durante la inicializacion
;-----------------------------------------------------------------
tack        segment     at 30h
            dw          128 dup(?)
os          label       word
tack        ends
```

```
; area de datos del bios                                          :
;-----------------------------------------------------------------
data      segment  at 40h
rs232_base  dw     4 dup(?)
printer_base dw    4 dup(?)
equip_flag  dw     ?
mfg_tst     db     ?
memory_size dw     ?
io_ram_size dw     ?
;-----------------------------------------------------------------
; area de datos del teclado                                       :
;-----------------------------------------------------------------
kb_flag     db     ?
; valores posibles de kb_flag
ins_state   equ    80h
caps_state  equ    40h
num_state   equ    20h
scroll_state equ   10h
alt_shift   equ    08h
ctl_shift   equ    04h
left_shift  equ    02h
right_shift equ    01h
kb_flag_1   db     ?
; valores posibles del segundo byte del estado del teclado
ins_shift   equ    80h
caps_shift  equ    40h
num_shift   equ    20h
scroll_shift equ   10h
hold_state  equ    08h
alt_input   db     ?
buff_head   dw     ?
buff_tail   dw     ?
kbd_buff    dw     16 dup(?)
kb_buff_end label  word

;-----------------------------------------------------------------
; area de datos del diskette                                      :
;-----------------------------------------------------------------
seek_status db     ?
motor_status db    ?
motor_count db     ?
motor_wait  equ    37
diskette_stat db   ?
fdc_status  db     7 dup(?)
;-----------------------------------------------------------------
; area de datos del video                                         :
;-----------------------------------------------------------------
crt_mode    db     ?
crt_cols    dw     ?
crt_len     dw     ?
crt_start   dw     ?
cursor_pos  dw     8 dup(?)
cursor_mode dw     ?
active_page db     ?
addr_6845   db     2 dup(?)
crt_mode_set db    ?
crt_palete  db     ?
;-----------------------------------------------------------------
; area de datos de cassete                                        :
;-----------------------------------------------------------------
edge_cnt    dw     ?
crc_reg     dw     ?
last_val    db     ?
;-----------------------------------------------------------------
; area de datos del timer
```

```
;-------------------------------------
timer_low       dw   ?
timer_high      dw   ?
timer_ofl       db   ?
counts_sec      equ  18
counts_min      equ  1092
counts_hour     equ  65543
counts_day      equ  1573040=1800b0h
;-------------------------------------------------------------------------
; area de datos del sistema                                              :
;-------------------------------------------------------------------------
bios_break      db   ?
reset_flag      dw   ?
;-------------------------------------------------------------------------
; area de datos del disco fijo                                           :
;-------------------------------------------------------------------------
                dw   ?
                dw   ?
;-------------------------------------------------------------------------
; tiempos de salida de la impresora y el rs232                           :
;-------------------------------------------------------------------------
print_tim_out db    4 dup(?)
rs232_tim_out db    4 dup(?)
;-------------------------------------------------------------------------
; area del teclado extra                                                 :
;-------------------------------------------------------------------------
buff_start      dw   ?
buff_end        dw   ?
data        ends
;-------------------------------------------------------------------------
; area de datos extra                                                    :
;-------------------------------------------------------------------------
xxdata      segment  at 50h
status_byte     db   ?
xxdata      ends
;
;-------------------------------------------------------------------------
;----- fin del area de datos                                            :
;-------------------------------------------------------------------------
;-------------------------------------------------------------------------
code        segment
assume      cs:code, ds:data
;----- constantes -------
pacs_val                equ     3eh
mmcs_val                equ     1fah
mpcs_val                equ     0c0beh
keyboard_reset          equ     1234h
bank1                   equ     1000h
bit7                    equ     80h
bios_data_c             equ     0
stack_val               equ     30h
drive_0                 equ     19h
no_drive_0              equ     8
error_bit               equ     40h
equip_par               equ     6fh
mfg_test                equ     0
m_size                  equ     120
io_size                 equ     40h
not_dsr                 equ     7fh
cond_inic               equ     5
inti_en                 equ     1
set_mode_cmd            equ     2
set_cur_type            equ     100h
range                   equ     607h
outsw                   equ      6fh
outsb                   equ      6eh
```

```
org     100h
start:
        jmp       instal
retpoint0 dw      err0                  ; direccines de retorno
retpoint1 dw      err1                  ; para MEMTEST
inic_start label  far
        mov     dx,pacs_reg             ; PCS a dir. base 0, 2 edos.
        mov     ax,pacs_val             ; de espera sin espera ext.
        out     dx,ax
        inc     dx                      ; MMCS a dir base 0, 2 edos.
        inc     dx                      ; de espera con espera ext.
        mov     ax,mmcs_val
        out     dx,ax
        inc     dx                      ; MM block size 512k, 7
        inc     dx                      ; PCS lines con 2 estados de
        mov     ax,mpcs_val             ; de espera sin espera externa
        out     dx,ax
reboot label  far
        cli
        mov     ax,bios_data
        mov     ds,ax
        mov     bp,reset_flag                   ; Es este un RESET de
        cmp     bp,keyboard_reset               ; teclado ?
        je      warm_start
        mov     ax,cs                           ; base
        mov     ss,ax
        mov     sp,offset retpoint0
        jmp     memtest
err0:   jne     memerr
        mov     ax,bank1
        mov     ds,ax
        mov     es,ax
        mov     sp,offset retpoint1
        jmp     memtest
err1:   je      inic_cont
memerr: mov     dx,fdc_cmd
        mov     al,bit7
        out     dx,al
        hlt
warm_start:
        sub     ax,ax                   ; llena primer
        mov     es,ax                   ; k byte con ceros
        assume  es:abs0
        mov     di,ax                   ; (1200 bytes)
        mov     cx,bios_data_c
        cld
clean:  stosw
        loop    clean
        mov     reset_flag,bp
inic_cont: mov  ax,stack_val            ; inicializa stack
        mov     ss,ax
        mov     sp,offset tos
        cld                             ; copia tabla de
        mov     cx,32                   ; vectores de interrupcion
        sub     di,di                   ; a inicio de la ram
        mov     si,offset vector_table
        push    cs
        pop     ds
        sub     ax,ax
        mov     es,ax
copy_vector: lodsw
        stosw
        mov     ax,cs
        stosw
        loop    copy_vector
        ...     offset timer_table
```

```
timer_init: mov   ax,offset dummy_return
            mov   cx,6                          ; timers 0,1 . 2
            mov   dx,timer_ctrl
copy_timer:
 db         outsw
            add   dl,4
            loop  copy_timer
            cli                                 ; no mas interrupciones
            mov   dx,int_ctrl                   ; inicializa controlador
            mov   cx,7                          ; de interrupciones
copy_int:
 db         outsw
            inc   dx
            inc   dx
            loop  copy_int
            sti                                 ; habilita interrupciones
            mov   ax,bios_data                  ; apunta ds a bios_data
            mov   ds,ax
            assume ds:data
            cmp   bp,keyboard_reset                 ; es Reset de teclado ?
            je    next_inic                     ; si- no pruebes disco
disk_test: mov   ah,0                           ; ejecuta ESPECIFY CMD
           mov   dl,ah
           int   13h
           jc    motor_off                      ; si error, apaga motor
           mov   dx,fdc_cmd                     ; prende motor,drive 0
           mov   al,drive_0
           out   dx,al
           mov   bl,3                           ; espera medio segundo
           xor   cx,cx
i_wait:    loop  i_wait
           dec   bl
           jnz   i_wait
           xor   dx,dx                          ; busca track 1
           mov   cx,1
           mov   seek_status.dl
           call  seek
           jc    motor_off                      ; si error, apaga motor
           mov   ch,34                          ; busca track 34
           call  seek
motor_off: mov   dx,fdc_cmd                     ; apaga motor y prende
           mov   al,no_drive_0                  ; RESET
           jnc   no_error                       ; verifica si hubo error
           or    al,error_bit                   ; si hubo - prende LED
no_error:  out   dx,al
next_inic: mov   di,410h
           mov   ax,equip_par                   ; de equipo
           stosw
           mov   al,mfg_test
           stosb
           mov   ax,m_size
           stosw
           mov   ax,io_size
           stosw
           mov   reset_flag,1234h
           mov   dx,uart_ctrl                   ; inicializacion del
           mov   si,offset uart_table
           push  cs
           pop   ds
           cld
           mov   cx,6                           ; seis valores en la
           cli                                  ; tabla
i_out:
 db        outsb
           nop                                  ; haz algo de tiempo
           nop
           loop                                 ; repite hasta terminar
```

```
        in      al,dx                   ; lee UART STATUS
        and     al,not_dsr              ; apaga DSR bit
        cmp     al,cond_inic            ; Tx y Rx puestos ?
        jmp     final_inic
        mov     dx,fdc_cmd
        mov     al,0c0h
        out     dx,al
        hlt
final_inic: mov dx,int_poll             ; apaga in service bits
        in      ax,dx
        mov     dx,int1_ctrl            ; habilita interrupcion
        mov     ax,int1_en              ; de consola
        out     dx,ax
        mov     ax,bios_data
        mov     ds,ax
        assume  ds:data
        mov     si,offset kbd_buff      ; inicializa cola de
        mov     buff_head,si            ; teclado
        mov     buff_tail,si
        mov     buff_start,si
        add     si,32
        mov     buff_end,si
        sti                             ; habilita interruociones
        mov     ax,set_mode_cmd         ; inicializa VIDEO
        int     10h
        mov     ax,set_cur_type
        mov     cx,range
        int     10h
        int     19h                     ; efectua BOOTSTRAP !!!
;-----------> fin de la inicializacion <----------------------------------!
memtest proc    near
        assume  cs:code, ds:data
        sub     cx,cx                   ; inicializa cuenta
m_count:  cld                           ; DF = incremento
        mov     bx,cx                   ; salva cuenta en BX
        mov     ax,0aaaah               ; inicializa patrones
        mov     dx,55ffh
        sub     di,di                   ; DI = 0
m_store: stosb                          ; carga memoria con 1er patron
        loop    m_store
backpass: dec   di                      ; apunta a ultimo byte escrito
        std                             ; DF = decremento
m_inic: mov     si,di                   ; inicializa SI y restaura
        mov     cx,bx                   ; cuenta
m_next: lodsb                           ; lee memoria y compara con
        xor     al,ah                   ; patron
        jne     m_fin                   ; Distintos ? Si; error
        mov     al,dl                   ; No; actualiza patron y llena
        stosb                           ; memoria
        loop    m_next
        and     ah,ah                   ; patron de ceros ?
        jz      m_fin                   ; Si: ya acabamos
        mov     ah,al                   ; No: mete patron nuevo a DL
        xchg    dh,dl
        and     ah,ah                   ; Es ceros ?
        jnz     continue                ; No: continua
        mov     dl,dh                   ; Si: pon ceros para ultima
                                        ;     escritura
        jmp     backpass                ; DF = incremento
continue: cld                           ; ajusta DI a final
        inc     di                      ; carga ultimos patrones
        jz      m_inic                  ; lee/escribe hacia atras
        dec     di                      ; DF = incremento y regresa
        mov     dx,1
        jmp     backpass
m_fin:  cld
```

```
memtest  endp
bootload proc   near
;-------- constantes ----------
;
         sti                                    ; no mas interrupciones
         sub    ax,ax                           ; DS=0
         mov    ds,ax
         assume  ds:abs0
         mov    word ptr disk_point,offset disk_base
         mov    word ptr disk_point+2,cs
         mov    cx,4                            ; no. de intentos
repeat:  push   cx                              ; salvalo
         mov    ah,0                            ; RST CMD al disco
         int    13h
         jc     next_try                        ; error- intentalo de nuevo
         mov    ax,201h
         sub    dx,dx                           ; comando para lectura
         mov    es,dx
         mov    bx,offset boot_loc
         mov    cx,1
         int    13h                             ; lee disco
next_try: pop   cx                              ; recupera cuenta
         jnc    cpm_start                       ; no hubo error
         loop   repeat                          ; si hubo -repite
no_boot: jmp    b_error
cpm_start: jmp  boot_loc                        ; transfiere control al
b_error:   jmp    examina                       ; Sistema Operativo
bootload endp
;
         assume  cs:code, ds:data
d_eoi    proc   near
;---------- constantes ----------
eoi_cmd    equ   8000h
;
         push   ax                              ; salva DX y AX
         push   dx
         mov    dx,eoi_reg                      ; emite non-especific
         mov    ax,eoi_cmd                      ; EOI CMD
         out    dx,ax
         pop    dx                              ; restaura registros y
         pop    ax                              ; regresa
         iret
d_eoi    endp
;
; int dummy rtn
;
dummy_rtn   proc  near
         iret
         nop
dummy_rtn endp
;
; int timer tick (1ch)
;
timer_tick  proc near
         iret
         nop
timer_tick endp
;
; int equip deter 11h
;
; Salida: AX contiene equipo de acuerdob al siguiente codigo:
;
; bit no.      0: FDD (drives)
;              1: No usado
;            2,3: tamano de RAM base en incr. de 16k
;                 4,5: modo de video
```

```
          6,7: Cant. de FDD's
            8: No usado
      9,10,11: no. de puertos RS232C
           12: tarjeta de juego
           13: no usada
        14,15: no. de impresoras

ip_deter proc near
        sti
        push    ds
        mov     ax,bios_data
        mov     ds,ax
        assume ds:data
        mov     ax,equip_flag
        pop     ds
        iret
ip_deter endp

nt memory syze deter 12h

alida: AX contiene numero de blockes adyacentes de 1k byte
          en memoria

ory_size_det proc near
        sti
        push    ds
        mov     ax,bios_data
        mov     ds,ax
        mov     ax,memory_size
        pop     ds
        iret
mory_size_det endp

int time of day   8h
Esta rutina lee e inicializa el reloj de tiempo real

Entradas:   AH=0    lectura del reloj. Devuelve CX = horas
                                               DX = cuenta baja
                                               AL = 0 si < 24 hrs
                                                    1 si => 24 hrs.

            AH=1    Inicializa reloj           CX = horas
                                               DX = cuenta baja

me_of_day proc near
        sti
        push    ds
        push    ax
        mov     ax,bios_data
        mov     ds,ax
        assume  ds:data
        pop     ax
        or      ah,ah
        jz      read_time
        dec     ah
        jz      set_time
_return:  sti
        pop     ds
        iret
ead_time: cli
        mov     al,timer_ofl
        mov     timer_ofl,0
        mov     cx,timer_high
        mov     dx,timer_low
        jmp     t_return
```

```
et_time: cli
         mov     timer_low,dx
         mov     timer_high,cx
         mov     timer_ofl,0
         jmp     t_return
         nop
ime_of_day endp

 int time of day (timer 0 int) 1ah

 Esta interrupcion cumple con las siguientes funciones:

 a.) Cuenta las interrupciones generadas por el timer 0 desde el
     encendido (18.204/seg).
 b.) Actualiza cuenta de motor del FDC , apaga motor al expirar
     cuenta y limpia banderas asociadas.
 c.) Invoca una rutina de servicio del usuario a traves de la
     interrupcion 1CH (timer tick)

timer0_int  proc near
;---------- constantes ---------
eoi_cmd_tim0   equ   8
enc            equ     0f4h
;
         sti                                    ; habilita interrupciones
         push    ds                             ; salva DS,AX y DX
         push    ax
         push    dx
         mov     ax,bios_data                   ; inicializa DS a bios_data
         mov     ds,ax
         assume ds:data
         inc     timer_low                      ; actualiza reloj de
         jnz     test_day                       ; tiempo real
         inc     timer_high
test_day: cmp    timer_high,24
         jnz     disk_control
         sub     ax,ax                          ; reinicializa reloj de
         mov     timer_high,ax                  ; tiempo real
         mov     timer_low,ax
         mov     timer_ofl,1
disk_control:
         dec     motor_count                    ; actualiza MOTOR COUNT
         jnz     tim_tick                       ; si no-0 , continua
         and     motor_status,enc               ; apaga banderas
         mov     al,8                           ; apaga todo menos RESET
         mov     dx,fdc_cmd
         out     dx,al
tim_tick: int 1ch                               ; llama interrupcion de
         mov     dx,eoi_reg                     ; usuario y termina la
         mov     ax,eoi_cmd_tim0                ; rutina de servicio
         out     dx,ax
         pop     dx                             ; restaura registros
         pop     ax                             ; utilizados y regres
         pop     ds
         iret
         nop
         nop
timer0_int endp
;
; int 1  console int  0dh
;
; Esta es la interrupcion por hardware  de la consola. Recibe dos
; bytes del teclado , o transmite un byte (AL: a la terminal.
; funcion especifica depende del modo de control programado
;
; Nota: el programa supone que el primer byte recibido es e
        '0800'  y el segundo el "scan code"  los bytes recib:
```

```
ole_int proc near
--- ------ constantes ----------
OY          equ     2
et          equ     0
_cmd_c      equ     0dh
_tx_cmd     equ     2

        push    ax                      ; salva registros utilizados
        push    dx
        push    bx
        push    si
        push    ds
        push    ax                      ; salva byte a transmitir
        mov     ax,bios_data            ; establece bios data area
        mov     ds,ax
        assume  ds:data
        mov     dx,uart_ctrl            ; lee UART STATUS
        in      al,dx
        test    al,RxRDY                ; int. causada por recepcion?
        pop     ax                      ; recupera byte a transmitir
        jz      transmit                ; No- transmite byte
        sti                             ; Si- no mas interrupciones
        dec     dx                      ; lee primer byte (ASCII)
        dec     dx
        in      al,dx
        push    ax                      ; salvalo
        mov     bx,buff_tail            ; actualiza buffer de teclado
        mov     si,bx                   ; bufftail original a SI
        call    act_pointer
        mov     buff_tail,bx            ; bufftail actualizado
        inc     dx                      ; 2- byte ya llego?
        inc     dx
_wait:    in      al,dx
        test    al,RxRDY                ; No- espera
        jz      c_wait                  ; Si- apunta a UART DATA
        dec     dx
        dec     dx
        pop     ax                      ; recupera primer byte
        xchg    ah,al                   ; salvalo en AH
        in      al,dx                   ; segundo byte a AL
        xchg    ah,al                   ; invierte posicion
        cmp     ax,reset                ; es RESET ?
        jne     c_cont                  ; no- continua
        mov     reset_flag,1234h
        jmp     reboot
_cont:  mov     [si],ax
_fin:   mov     dx,eoi_reg              ; emite eoi cmd
        mov     ax,eoi_cmd_c
        out     dx,ax
        pop     ds                      ; recupera registros
        pop     si                      ; originales
        pop     bx
        pop     dx
        pop     ax
        iret                            ; regresa de la interrupcion

transmit: dec   dx
        dec     dx
        out     dx,al
        inc     dx
        inc     dx
        mov     al,not_tx_cmd
        out     dx,al
        jmp     c_fin
```

```
_____t  endp
;
; keyboard i/o  int 16h                                          A1)
; interrupcion de servicio al teclado
; Entradas:  AH=0  espera al sig. caracter del teclado. Regresa con
;                  codigo ASCII en AL, "scan code" en AH.
;            AH=1  controla ZF:
;                  ZF=0 si hay codigo; esta en AX
;                  ZF=1 no hay codigo.
;            AH=2 transfiere "shift status" a AL
;
keyboard_io  proc far
;-------- constantes ---------
shift_stat_cmd    equ    8dh
        sti                                 ; hab. interrupciones
        push    ds                          ; salva registros utilizados.
        push    bx
        push    dx
        push    ax                          ; salva comando
        mov     ax,bios_data                ; DS apunta a BIOS DATA
        assume  ds:data
        mov     ds,ax
        pop     ax                          ; re cupera comando
        or      ah,ah                       ; cmd=0 ?
        jz      ascii_read                  ; si- espera caracter
        dec     ah                          ; cmd= 1 ?
        jz      ascii_stat                  ; si- controla ZF
        dec     ah                          ; cmd=2;?
        jz      shift_stat                  ; si- obten shift status
k_fin:    pop     dx                        ; Ninguno de los anteriores
        pop     bx                          ; recupera registros y regresa
        pop     ds
        iret
ascii_read: sti                             ; deja que ocurra una
        nop                                 ; interrupcion
        cli
        mov     bx,buff_head                ; compara cabeza con cola
        cmp     bx,buff_tail
        jz      ascii_read                  ; iguales?- no hubo int.
        mov     ax,[bx]                     ; lee caracter
        call    act_pointer                 ; actualiza apuntador
        mov     buff_head,bx
        jmp     k_fin                       ; terminamos
ascii_stat: cli                             ; no mas interrupciones
        mov     bx,buff_head                ; compara cabeza con cola
        cmp     bx,buff_tail
        mov     ax,[bx]                     ; lee valor
        sti                                 ; hab. interrupciones
        pop     dx                          ; recupera registros
        pop     bx
        pop     ds
        ret     +2                          ; regresa bandera de consola
shift_stat: mov  al,shift_stat_cmd
        call    console_tx
        call    console_rx
        mov     kbd_flag,al
        jmp     k_fin
keyboard_io  endp
;
; subrutina act pointer
;
act_pointer  proc  near
        inc     bx
        inc     bx
        cmp     bx,buff_end
        jne     rt
```

```
        mov
rt:     ret
act_pointer endp
;
; subrutina console tx
;
; Esta subrutina transmite el contenido de AL a la consola, haciendo
; "polling" sobre terminal READY. No altera registros.
;
console_tx proc  near
;---------- constantes ----------
txrdy       equ     1
tx_cmd      equ     7
rx_cmd      equ     6
dsr_bit     equ     80h
        push    dx                      ; Salva DX,CX y AX
        push    cx
        push    ax
        mov     dx,uart_ctrl            ; apunta a UART CTRL
tx_wait:        in      al,dx           ; lee status
        cmp     al,85h                  ; UART Listo ?
        jnz     tx_wait                 ; No- espera
        pop     ax
        dec     dx
        dec     dx
        out     dx,al
        inc     dx
        inc     dx
        mov     cx,40h
tx_wait1: in    al,dx
        cmp     al,0
        jz      tx_cont
        loop    tx_wait1
tx_cont: pop    cx
        pop     dx
        sti
        ret
console_tx endp
;
; subrutina console rx ; recibe un byte del teclado y lo deja en AL.
; La subrutina deshabilita CONSOLE INT,esto es ,inhibe recepcion del
; teclado, solamente AL es alterado.
;
console_rx proc near
;---------- constantes ----------
no_int1     equ     0fh
;
        push    dx                      ; salva registros involucrados
        push    cx
        mov     ch,ah                   ; salva AH en CH
        mov     dx,int1_ctrl            ; apunta a INT1 CTRL
        push    dx                      ; salva offset
        mov     ax,no_int1              ; no mas interrupciones
        out     dx,ax                   ; por esa linea
        mov     dx,uart_ctrl            ; UART listo para ser leido?
rx_wait:        in      al,dx
        test    al,rxrdy
        jz      rx_wait                 ; No- espera
        dec     dx                      ; si- lee byte recibido
        dec     dx
        in      al,dx
        mov     cl,al                   ; salva byte en CL
        mov     dx,int_poll             ; limpia "service request" bit
        in      ax,dx
        pop     dx                      ; recupera INT1 CTRL
        mov     ax,int1_en              ; habilita interrupcion 1
```

```
        mov     dx,ax
        mov     ax,cx                   ; recupera byte leido
        pop     cx                      ; recupera registros
        pop     dx
        ret                             ; regresa
console_rx endp
;
; int 10h  video i/o
;
; Esta rutina atiende los requerimientos de uso del VIDEO.
; Las funciones soportadas son las siguientes:
;
;     AH=0      SET MODE  (AL contiene el codigo del modo)
;
;     AH=1      SET CURSOR TYPE
;
;     AH=2      SET CURSOR POSITION
;
;     AH=5      SELECT ACTIVE PAGE
;
;     AH=6      SCROLL ACTIVE PAGE UP
;
;     AH=7      SCROLL ACTIVE PAGE DOWN
;
;     AH=8      READ ATRIBUTE/CARACTER AT CURRENT CURSOR POSITION
;
;     AH=9      WRITE ATRIBUTE/CARACTER AT CURRENT CURSOR POSITION
;
;     AH=10     WRITE CHARACTER ONLY AT CURRENT CURSOR POSITION
;
;     AH=14     WRITE TELETYPE TO ACTIVE ACTIVE PAGE
;
;     AH=15     CURRENT VIDEO STATE
;
;     El manejo de los parametros de cada funcion se describe el la
;     rutina correspondiente.
;
video_io  proc near
;------------- constantes -------------
v_rango       equ       20h
;
        push    es                      ; salva segmentos
        push    ds
        push    dx                      ; salva registros
        push    cx
        push    bx
        push    si
        push    di
        push    ax                      ; salva comando
        mov     al,ah                   ; conviertelo en apuntador
        xor     ah,ah
        sal     ax,1
        mov     si,ax                   ; salva apuntador en SI
        cmp     ax,v_rango              ; comando valido ?
        jb      exec
        pop     ax                      ; no- restaura todo y regresa
video_rtn: pop  di
        pop     si
        pop     bx
part_rtn: pop   cx
        pop     dx
        pop     ds
        pop     es
        iret
exec:   sti
        mov     ax,bios_data            ; si-,DS apunta a BIOS DATA
```

```
        assume  ds:data
        pop     ax
        mov     ah,crt_mode                     ; lee CRT MODE ACTUAL
        jmp     cs:[si+offset tabla]            ; brinca a rutina correspon.
la      label   word
        dw      offset  set_mode        ;ah=0
        dw      offset  set_cursor_type ;ah=1
        dw      offset  set_cursor_pos  ;ah=2
        dw      offset  read_cursor_pos ;ah=3
        dw
        dw      offset  selec_act_page  ;ah=5
        dw      offset  scroll_up       ;ah=6
        dw      offset  scroll_down     ;ah=7
        dw      offset  read_ac_curr    ;ah=8
        dw      offset  write_ac_curr   ;ah=9
        dw      offset  write_c_curr    ;ah=a
        dw                              ;ah=b
        dw                              ;ah=c
        dw                              ;ah=d
        dw      offset  write_tty       ;ah=e
        dw      offset  curr_video      ;ah=f
------> fin de la tabla

int 10h set mode (ah=0)

Esta rutina inicializa la terminal y VIDEO DATA en el BIOS AREA
la rutina soporta unicamente el modo 2 ( BW , 80 columnas )

Entrada:  AL= mode

et_mode label word
        push    ax                              ; salva modo
        mov     crt_mode,al
        push    ds
        pop     es                              ; ES a bios data
        cld                                     ; incremento
        mov     di,4ah
        mov     ax,50h                          ; 80 columnas a CRT COLS
        stosw
        mov      cx,10                          ; ceros a CRT LEN,CRT START
        mov      al,0                           ; y CURSOR POS 0-7
tore:   stosw
        loop    store
        mov     ax,607h                         ; CURSOR MODE
        stosw
        xor     al,al                           ; ACTIVE PAGE
        stosb
        mov     al,0ff20h                       ; ADDR 6845 (space , no attr.)
        stosw
        mov     al,29h                          ; CRT MODE SET
        stosb
        xor     al,al                           ; CRT PALETTE
        stosb
        mov     al,80h                          ; comando al stack
        push    ax
        mov     cx,2                            ; dos bytes a transmit:
        jmp     transmit_string

nt 10h set cursor type (ah=1)

Esta rutina determina la linea inicial y final entre las cuales
el cursor esta activo. Actualiza CURSOR MODE.

Entradas:  CH= start line, CL= stop line
```

```
        mov     cursor_mode,cx          ; actualiza CURSOR MODE
        xchg    ch,cl                   ; start line al stack
        push    cx
        xchg    ch,cl                   ; stop line al stack
        push    cx
        mov     al,81h                  ; cmd  al stack
        push    ax
        mov     cx,3                    ; tres bytes a transmitir
transmit_string: pop ax                 ; byte a trans. en AL
        call    console_tx              ; transmitelo
        loop    transmit_string         ; repite hasta terminar
        jmp     video_rtn
;
; int 10h set cursor position  (AH=2)
;
; Esta rutina define la posicion del cursor. DH=fila,DL=columna,
; BH=pagina. Actualiza CURSOR POS de acuerdo con la pagina.
; Envia  comando solo si BH contiene pagina actualmente desplegada.
;
set_cursor_pos:
        mov     cl,bh                   ; calcula offset a video
        xor     ch,ch                   ; param. tabla
        sal     cx,1
        mov     si,cx                   ; offset en SI
        mov     offset cursor_pos[si],dx ; actualiza CURSOR POS
        cmp     active_page,bh          ; pagina act. desplegada
        jz      cont1                   ; si= continua
        jmp     video_rtn               ; no= ya terminamos
cont1:  xchg    bh,bl                   ; pagina al stack
        push    bx
        xchg    dh,dl                   ; fila al stack
        push    dx
        xchg    dh,dl                   ; columna al stack
        push    dx
        mov     al,82h                  ; comando al stack
        push    ax
        mov     cx,4                    ; cuatro bytes a trans.
        jmp     transmit_string         ; transmitelos.
;
; int 10h  read cursor position  (ah=3)
;
; Esta rutina lee CURSOR POSITION y CURSOR MODE a DX y CX,
; respectivamente.
;
read_cursor_pos  label  word
        mov     al,83h
        call    console_tx
        mov     al,bh
        call    console_tx
        call    console_rx
        mov     dl,al
        call    console_rx
        mov     dh,al
        mov     cx,cursor_mode
        pop     di
        pop     si
        pop     bx
        pop     ax
        pop     ax
        pop     ds
        pop     es
        iret
;
; int 10h select active page  (ah=5)
;
; AL contiene el valor de la pagina deseada.
```

```
ACTIVE PAGE e. actualizado.

selec_act_page  label  word
        mov     active_page,al          ; actualiza ACTIVE PAGE
        mov     cx,crt_len              ; lee tamano de RAM
        cbw                             ; convierte AL a palabra
        push    ax                      ; sal a AL
        mul     c                       ; calcula dir. inicial
        mov     cr. start,a             ; / salvala
        pop     bx                      ; pag. deseada a BL
        sal     bx,1                    ; x 2 para offset
        mov     si,bx                   ; SI apunta a CURSOR POS
        mov     al,85h                  ; cmdo set active page
        call    console_tx              ; transmitelo
        mov     al,bl                   ; recupera num. de pagina
        shr     al,1                    ; deseada
        call    console_tx              ; transmite num. de pagina
        mov     [si+offset cursor_pos],dx ; lee cursor position
        mov     bh,al                   ; pag. deseada a BH
        jmp     conti                   ; ejecuta SET CURSOR POSITION

;
; int 10h  scroll active page (up:ah=6 , down:ah=7)
;
; Esta rutina hace "scrolling" de acuerdo a los parametros:
;
; AL - numero de lineas. El valor cero implica limpiar toda la pantall.
; CH - fila inicial  ; CL - columna inicial (esq. superior izquierda)
; DH - fila final    ; DL - columna final   (esq. inferior derecha)
; BH - atributo del las lineas de entrada
;
scroll_up  label  word
        mov     bl,86h                  ; cmdo a BL
        jmp     cont2                   ; brica a cont.
scroll_down: mov bl,87h                 ; cmd a BL
cont2:  xchg    bl,bh
        push    bx
        xchg    bl,b
        xchg    dl,d
        push    dx
        xchg    dl,dh
        push    dx
        xchg    cl,ch
        push    cx
        xchg    ch,cl
        push    cx
        push    ax
        push    bx
        mov     cx,7
        jmp     transmit_string
;
; int 10h  read ac current  (ah=8)
;
; Esta rutina pide a la terminal el caracter y atributos de la pos.
; actual del cursor.
;
; Entrada:   BH = numero de pagina
; Salida :   AL = caracter
;            AH = atributos (formato IBM)
read_ac_curr label word
        mov     al,88h                  ; envia comando
        call    console_tx
        mov     al,bh                   ; envia numero de pagina
        call    console_tx
        call    console_rx              ; recibe atributo
        mov     ah,al
```

```
        mov     ah,al
        call    console_rx
        jmp     video_rtn
;
; int 10h  writte ac current   (ah=9)
;
; Esta rutina escribe n bytes a partir de la posicion actual del cursor.
; salva el caracter en ADDR 6845 low, y el atributo modificado en
; ADDR 6845 high. Estos valores son usados por la rutina WRITE TTY.
;
; Entradas:  BH = numero de pagina, BL = atributo de caracter
;            CX = no. de bytes, AL = caracter.
;
write_ac_curr  label  word
        push    ax                              ; convierte atributos
        mov     al,bl
        call    attr_conv
        mov     bl,al
        mov     addr_6845[1],al                 ; salva atributos en video area
        pop     ax
        mov     dx,689h                         ; cmdo y cuenta a DX
        xchg    bh,bl                           ; pagina al stack
        push    bx
write_remain: mov  addr_6845,al                 ; salva caracter en video area
        xchg    bh,bl                           ; atributo al stack
        push    bx
        xchg    ch,cl                           ; cuenta alta al stack
        push    cx
        xchg    ch,cl                           ; cuenta baja al stack
        push    cx
        push    ax                              ; caracter al stack
        mov     al,dl                           ; comando al stack
        push    ax
        mov     cl,dh                           ; no. de bytes a transmitir
        xor     ch,ch
        jmp     transmit_string
        nop
write_c_curr: mov dx,58ah                       ; cmdo y cuenta a DX
        jmp     write_remain
        nop
;
; int 10h write tty  (ah=e)
;
; Esta rutina simula un teletipo. CURSOR POSITION es actualizado.
; El caracter escrito es almacenado en ADDR 6845 low.
;
write_tty  label word
;---------- constantes ----------
tty_cmd     equ       8eh
;
        mov     ah,tty_cmd
        xchg    ah,al
        call    console_tx
        xchg    ah,al
        call    console_tx
        jmp     video_rtn
;
; int 10h  current video state (ah=f)
;
; Esta rutina devuelve MODE en AL , no. de columnas en AH y
; pagina activa en BH.
;
curr_video  label  word
        mov     ah,byte ptr crt_cols
        mov     al,crt_mode
```

```
        pop     di
        pop     si
        pop     cx
        jmp     part_rtn
video_io  endp
;
; diskette int (0eh)
;
; Esta rutina maneja la interupcion generada por el FDC (int 0Eh).
; la rutina pone la bandera de interrupcion el SEEK STATUS.
; El controlador de interrupciones debera inicializarse como sigue:
; INT 2 CTRL (offset 3ch) = 6  (prioridad 6 , unmasked, edge-triggered)
; EOI CMD    (offset 22h) = 14
;
disk_int proc near
;---------- constantes --------------
int_flag        equ     80h
eoi_cmd_e       equ     0eh
;
        sti
        push    ds
        push    ax
        mov     ax,bios_data
        mov     ds,ax
        or      seek_status,int_flag
        mov     al,eoi_cmd_e
        push    dx
        mov     dx,eoi_reg
        out     dx,ax
        pop     dx
        pop     ax
        pop     ds
        iret
disk_int endp
;
; int diskette i/o 13h
;
; Este es el vector de servicio del disco flexible.
; El software soporta dos drives.
; El controlador esta conectado a la int. 2 del 80186
;
; La interrupcion soporta los siguientes comandos:
;                         AH=0    RESET DISK SYSTEM
;                            1    READ STATUS
;                            2    READ SECTORS
;                            3    WRITE SECTORS
;                            4    VERIFY SECTORS
;                            5    FORMAT TRACK
;
; Los parametros para 2,3 y 4 son:
;                          AL:  numero de sectores (cantidad)
;                          CL:  no. de sector , 1 a 8
;                          CH:  no. de track  , 0 a 39
;                          DL:  Drive , DH: Head
;
diskette_io  proc  far
        sti                                     ; habilita interrupciones
        push    bx                              ; salva registros
        push    cx
        push    ds
        push    si
        push    di
        push    bp
        push    dx
        mov     bp,sp                           ; inicializa BP = SP
                                                ; inicializa DS a BIOS DATA
```

```
        push    cx
        mov     ax,bios_data
        mov     ds,ax
        assume  ds:data
        pop     ax
        call    disk_exec                       ; procede con el comando
        push    ax                              ; salva sectores transferidos
        mov     bx,4                            ; llama retardo de apagado
        call    get_param
        mov     motor_count,ah
        pop     ax                              ; recupera sectores transferidos
        mov     ah,diskette_stat                ; lee status de la operacion y
        cmp     ah,1                            ; prende CF si hubo error
        cmc
        pop     dx                              ; restaura registros utilizados
        pop     bp
        pop     di
        pop     si
        pop     ds
        pop     cx
        pop     bx
        ret     +2                              ; regresa conservando CF
diskette_io endp
;
; subrutina disk exec
;
; Esta subrutina ejecuta el comando invocado por la int. 13H
; Entradas:  el comando con parametros
;            ES:BX   =  direccion inicial de RAM
;
; Salidas:        AL   =  System status (AH=1) o numero de sectores leidos.
;                 AH   =  Status de la operacion (codigo de error, en su caso)
;
; Nota:           En relacion al BIOS de la IBM-PC, se introducen los
;                 siguientes cambios:
;                 a.) Soporta unicamente dos drives
;                 b.) bits 6 y 7 de MOTOR STATUS no se usan.
;                     Todos los comandos excepto Reset y System status
;                     generan espera de motor.
;                 c.) TC - terminal count - se maneja por software
;
        assume  cs:code, ds:data
disk_exec  proc  near
;------------ constantes -------------
bit5            equ         20h
reset_bit       equ         8
cmd_bits        equ         0c0h
fdc_read_cmd    equ         0e6h
tc_bit          equ         4
fdc_format_cmd  equ         4dh
motor_bits      equ         3
dma_read_cmd    equ          0a246h
dma_write_cmd   equ          1686h
fdc_write_cmd   equ          0c5h
dma_verify_cmd  equ         8246h
dma_format_cmd  equ         1486h
rec_not_found   equ         4
bad_crc         equ         10h
bad_dma         equ         8
write_protec    equ         3
bad_addr_mark   equ         2
bad_cmd         equ         1
bad_fdc         equ         20h
boundary        equ         9
no_tc_bit       equ         0fbh
;
        mov     dh,al
```

```
        or      ah,ah                   ; AH=0?
        jz      disk_res
        dec     ah                      ; AH=1?
        jz      disk_status
        mov     diskette_stat,0         ; inic. diskette status
        cmp     dl,2                    ; rango de drives bien?
        jae     cmd_error               ; no- mal comando
        dec     ah                      ; AH=2 ?
        jz      disk_read
        dec     ah                      ; AH=3 ?
        jz      disk_write
        dec     ah                      ; AH=4 ?
        jz      disk_verify
        dec     ah                      ; AH=5 ?
        jz      disk_format
cmd_error: mov  diskette_stat,bad_cmd   ; ninguno de los anteriores
        ret                             ; - regresa
disk_res: mov dx,fdc_cmd                ; apunta a FDC CMD REG
        cli                             ; no mas interrupciones
        mov     al,motor_status         ; algun motor prendido ?
        mov     cl,4                    ; ponlo en nible alto
        sal     al,cl                   ; y habilita drive
        test    al,bit5
        jz      drive0
        inc     al
drive0: inc     al
        out     dx,al
        mov     seek_status,0           ; RESET al FDC
        mov     diskette_stat,0         ; inicializa SEEK y
        or      al,reset_bit            ; DISKETTE status
        out     dx,al                   ; apaga RESET del FDC
        sti                             ; habilita interrupciones
        call    sense_int_status        ; staus despues de reset
        mov     al,fdc_status           ; lee status reg. 0
        cmp     al,cmd_bits             ; FDC listo para recibir com.?
        jz      especify
        or      diskette_stat,bad_fdc   ; no- FDC malo
        ret
especify: mov   ah,3                    ; SPECIFY cmd al FDC
        call    fdc_output
        mov     ah,6fh
        call    fdc_output
        mov     ah,8
        call    fdc_output
        ret
        add     bx,ax
disk_status: mov al,diskette_stat
        ret
disk_read:      mov     ax,dma_read_cmd
dma_start: call inic_dma
        mov     ah,fdc_read_cmd
        jmp     r_w_v
disk_write:     mov     ax,dma_write_cmd
        call inic_dma
        mov     ah,fdc_write_cmd
        jmp     r_w_v
disk_verify: mov        ax,dma_verify_cmd
        jmp     dma_start               ; haz como lectura
        nop
disk_format: mov        ax,dma_format_cmd
        call    inic_dma
        mov     ah,fdc_format_cmd
r_w_v:  jnc     cont3                   ; error en DMA SET UP ?
        mov     diskette_stat;bound_er  ; si- prende boundary error
        mov     al,0                    ; apaga TC bit y regresa
```

```
        ret
cont3:  push    ax                      ; salva cmdo.track y sector
        push    cx
        mov     cl,dl                   ; utiliza drive deseado y
        mov     al,1                    ; establece motor en. en AL
        sal     al,cl
        mov     cl,4                    ; al nible alto
        sal     al,cl
        test    al,bit5                 ; prende bit correspondiente
        jz      disk_next               ; en nible bajo
        inc     al
disk_next:      inc     al
        mov     ah,motor_status         ; lee motor status
        test    ah,tc_bit               ; establece TC enable en AL
        jz      no_tc
        or      al,tc_bit
no_tc:  mov     motor_status,al         ; actualiza motor stat.
        and     al,no_tc_bit            ; apaga TC bit en AL
        or      al,reset_bit            ; y prende RESETn bit
        cli                             ; no mas interrupciones
        mov     motor_count,0ffh        ; cuenta larga
        push    dx                      ; manda drive en. , motor en.
        mov     dx,fdc_cmd              ; y reset al FDC
        out     dx,al
        pop     dx
        sti                             ; habilita interrupcines
        and     ah,motor_bits           ; motor deseado ya estaba
        test    ah,al                   ; andando ?
        jnz     no_wait                 ; si- no esperes
        mov     bx,14h                  ; no- espera que disco
        call    get_param               ; gire a velocidad estable
        or      al,ah
wait1:  jz      no_wait
        sub     cx,cx
wait2:  loop    wait2
        dec     ah
        jmp     wait1
no_wait: pop    cx                      ; recupera track/sector
seek1:  call    seek                    ; ejecuta SEEK TRACK
        pop     ax                      ; recupera comando
        mov     bh,ah                   ; salva comando en BH
        mov     dh,0
        jc      error1                  ; error en SEEK
        mov     si,offset error1        ; retorno si error en
        push    si                      ; FDC WRITE
        call    fdc_output              ; comando al FDC
        mov     ah,[bp+1]
        sal     ah,1
        sal     ah,1
        and     ah,4
        or      ah,dl
        call    fdc_output              ; primer param. al FDC
        cmp     bh,fdc_format_cmd       ; es formateo ?
        jne     next_word               ; si- brinca a otros param.
        mov     bx,7
        call    get_param               ; no- seg. param. al FDC
        mov     bx,9
        call    get_param               ; tercer param. al FDC
        mov     bx,15
        call    get_param               ; cuarto param. al FDC
        mov     bx,17                   ; offset para quinto param.
        jmp     last_word
next_word: mov  ah,ch
        call    fdc_output              ; segundo param. al FDC
        mov     ah,[bp+1]
```

```
        mov     ah,cl
        call    fdc_output              ; cuarto param. al FDC
        mov     bx,7
        call    get_param               ; quinto param. al FDC
        mov     bx,9
        call    get_param               ; sexto param. al FDC
        mov     bx,11
        call    get_param               ; septimo param. al FDC
        mov     bx,13
last_word: call get_param               ; ultimo param. al FDC
        pop     si                      ; no hubo error en cmdo.
        call    wait_int                ; que suceda la operacion
error1: jc      error2                  ; error en espera
        call    fdc_results             ; llama resultados
        jc      return                  ; error en lectura de resul.
        cld                             ; DF -> incremento
        mov     si,offset fdc_status    ; apunta a fdc status 0
        lods    fdc_status              ; lee valor
        and     al,0c0h                 ; bits 6 y 7 =0 ?
        jz      op_ok                   ; operacin sin error
        cmp     al,40h                  ; bit 7=0 , 6=1 ?
        jnz     no_ok                   ; no- FDC debe estar malo
        lods    fdc_status              ; lee staus 1
        sal     al,1                    ; bit 7 puesto ?
        mov     ah,rec_not_found
        jc      rw_fail
        sal     al,1                    ; bit 5 puesto ?
        sal     al,1
        mov     ah,bad_crc
        jc      rw_fail
        sal     al,1                    ; bit 4 puesto ?
        mov     ah,bad_dma
        jc      rw_fail
        sal     al,1                    ; bit 2 puesto ?
        sal     al,1
        mov     ah,rec_not_found
        jc      rw_fail
        sal     al,1                    ; bit 1 puesto ?
        mov     ah,write_protec
        jc      rw_fail
        sal     al,1                    ; bit 0 puesto ?
        mov     ah,bad_addr_mark
        jc      rw_fail
no_ok:  mov     ah,bad_fdc
rw_fail: or     diskette_stat,ah        ; error a DISKETTE_STATUS
        call    num_trans               ; cuantos sectores transferidos ?
        jmp     return
error2: call    fdc_results
        jmp     return
op_ok:  call    num_trans
        xor     ah,ah
return: pushf
        and     motor_status,motor_bits
        popf
        ret
disk_exec endp
;
; subrutina fdc results
; Esta subrutina lee los registros de estado del FDC al concluir
; este un comando.
; Salida:   cy = 1  si se exede el periodo de espera
;
fdc_results  proc  near
;---------- constantes ----------
time_out        equ     80h
                equ     10h
```

```
        cld                                     ; dir = incremento
        mov     di,offset fdc_status            ; DI  = inicio de tabla
        push    cx
        push    dx
        push    bx
        mov     bl,7                            ; max. num. de status bytes
next1:  xor     cx,cx
        mov     dx,fdc_main_status              ; lee main status
wait3:  in      al,dx
        test    al,80h                          ; req. for master
        jnz     direcc                          ; si- ve a dir
        loop    wait3                           ; no- espera
        or      diskette_stat,time_out          ; acabo el tiempo
error:  stc                                     ; error
fin3:   pop     bx                              ; regresa
        pop     dx
        pop     cx
        ret
direcc: in      al,dx                           ; req. for read ?
        test    al,40h
        jnz     read1                           ; si- lee al FDC
bad_oper: or    diskette_stat,bad_fdc           ; no- FDC malo
        jmp     error
read1:  inc     dx                              ; lee status y dep. en
        inc     dx                              ; BIOS DATA area
        in      al,dx
        mov     [di],al
        inc     di
        mov     cx,10                           ; espera para sig. byte
wait4:  loop    wait4
        dec     dx
        dec     dx
        in      al,dx                           ; FDC aun ocupado ?
        test    al,fdc_busy_bit
        jz      fin3                            ; no- ya terminamos
        dec     bl
        jnz     next1                           ; no leimos 7 ?
        jmp     bad_oper                        ; si- FDC malo
fdc_results endp
;
; subrutina fdc output
;
; Esta rutina envia un byte al, FDC, verificando previamente al
; MAIN STATUS REG la rutina espera un tiempo para que el byte sea
; aceptado. Si no es aceptado en ese tiempo, prende al bit 7 de
; DISKETTE STATUS y la bandera de acarreo.
;
; Entrada:   AH = byte a escribir
; Salida :   CF = 1 si el byte no fue aceptado. En ese caso, el regreso
;                 se efectua un nivel mas alto al de la llamada de la
;                 subrutina - esto es, regresa al vector 13h o al
;                 punto de llamada de GET PARAM.
;
fdc_output  proc  near
        push    dx
        push    cx
        mov     dx,fdc_main_status              ; lee main status
        xor     cx,cx
fdc_wait1:      in      al,dx
        test    al,40h                          ; req. for write ?
        jz      output                          ; si- ve a salida
        loop    fdc_wait1                       ; no- espera
error_out: or   diskette_stat,80h               ; espera terminada
        pop     cx                              ; regresa con error
        pop     dx                              ; un nivel mas alto
```

```
        pop     ax
        stc
        ret
tput:   xor     cx,cx
_wait2:  in     al,dx                   ; lee main status
        test    al,80h                  ; req. for master ?
        jnz     write                   ; si- escribe
        loop    fdc_wait2               ; no- espera
        jmp     error_out
te:     mov     al,ah                   ; escribe byte al FDC
        mov     dl,2                    ; y regresa
        out     dx,al
        pop     cx
        pop     dx
        ret
c_output endp

subrutina get param

Esta subrutina lee un byte de la tabla Disk Base. El apuntador a
la tabla es la variable Disk Pointer (int. 1eh). El offset a la
tabla (x2) esta en BX . Si el valor de BX es impar, el byte leido,
es transferido al FDC.

Entrada:        BX = indice de la tabla
Salida :        AH = byte leido

AX es alterado.

et_param  proc  near
        assume  ds:abs0
        push    ds
        sub     ax,ax
        mov     ds,ax
        lds     si,ds:disk_point
        shr     bx,1
        mov     ah,[si+bx]
        pop     ds
        jc      fdc_output
        ret
et_param endp
        assume  ds:data

subrutina num trans

Esta rutina calcula el numero de sectores que fueron transferidos
de o hacia el disco.

Entrada:  CH = no. de track
          CL = sector inicial de operacion

Salida :  Al = cantidad de sectores transferidos

um_trans  proc  near
        mov     al,fdc_status+3         ; AL = ultimo track
        cmp     al,ch                   ; igual al inicial ?
        mov     al,fdc_status+5         ; AL = sector final
        jz      cont0                   ; mismo track: no hay correc.
        mov     bx,8                    ; no- lee ultimo sector
        call    get_param
        mov     al,ah                   ; AL = ultimo sector leido
        inc     al                      ; AL = AL+1
cont0:  sub     al,cl                   ; ultimo menos primero
        ret
um_trans endp
```

```
;subrutina seek
;
; Esta subrutina busca el track deseado. Si el drive accesado desde
; RESET, el comando RECAL (localizacion del track 0) es efectuado.
; Nota: el comando RECAL se intenta 2 veces antes de reportar error.
;
; Entradas: CH = no. de track  DL = no. de drive
; Salidas : CF = 1 error (track no localizado) , modifica DISKETTE STAT.
;
; AX,BX son destruidos
;
seek  proc  near
;--------- constantes ----------
recal_cmd       equ     7
seek_cmd        equ     0fh
;
        mov     al,1                    ; establece mascara para
        push    cx                      ; determinar si requiere REC.
        mov     cl,dl
        rol     al,cl
        pop     cx
        test    al,seek_status          ; requiere RECAL ?
        jnz     move_track              ; no- busca track
        or      seek_status,al          ; si- prede staus bit
        mov     bl,2                    ; establece no. de intentos
recal:  mov     ah,recal_cmd            ; envia RECAl y drive no.
        call    fdc_output              ; al FDC
        mov     ah,dl
        call    fdc_output
        call    sense_int_status        ; track 0 localizado ?
        jnc     move_track              ; si- busca track
        dec     bl                      ; no- intentalo otra vez
        jnz     recal
        stc                             ; aun no : error
        ret
move_track: mov ah,seek_cmd             ; envia SEEK cmdo
        call    fdc_output
        mov     ah,dl                   ; envia drive no.
        call    fdc_output
        mov     ah,ch                   ; envia track no.
        call    fdc_output
        call    sense_int_status        ; averigua que paso
        pushf
        mov     bx,18                   ; llama head settle param. y
        call    get_param               ; espera
        push    cx                      ; salva datos
more_wait: mov  cx,4b0h                 ; cuenta para 1 mseg.
        or      ah,ah                   ; espera terminada ?
        jz      fin2                    ; si- ve a FIN
s_wait: loop    s_wait                  ; no- sigue esperando
        dec     ah
        jmp     more_wait
fin2:   pop     cx
        popf
        ret
seek    endp
;
; subrutina sense int status
;
; Esta rutina da servicio a la interrupcion generada por el FDC
; al concluir un comando RECAL, SEEK o RESET. La rutina espera
; la interrupcion, lee el status correspondiente y regresa el
; resultado.
;
; Salida:  CF = 1  error en DISKETTE STATUS
```

```
; AX es destruido

sense_int_status  proc  near
;---------- constantes ------------
bad_seek          equ           40h
sense_int_cmd     equ           8
;
        call    wait_int                ; espera la interrupcion
        jc      sen_return              ; regresa sobre error
        mov     ah,sense_int_cmd        ; ejecuta SENSE INT. cmd
        call    fdc_output
        call    fdc_results             ; lee resultados
        jc      sen_return              ; regresa si error
        mov     al,fdc_status           ; examina status 1
        and     al,60h                  ; aisla bits 5 y 6
        cmp     al,60h                  ; ambos puestos ?
        jz      sen_error               ; si- error
        clc                             ; no- todo en orden
sen_return: ret                         ; regresa
sen_error:  or  diskette_stat,bad_seek  ; mal SEEK, prende CF y
        stc                             ; regresa
        ret
sense_int_status  endp

; int 10h  subrutina atribute conv

; Esta subrutina convierte atributos del formato IBM al formato
; de la terminal.

; Entrada:   AL = atributos
; Salida :   AL = atributos

; No altera registros excepto AL
attr_conv  proc  near
;---------- constantes --------------
blinking        equ         80h
intens          equ         8
non_dspl        equ         0efh
no_int          equ         0f7h
inv_mode        equ         7fh
;
        push    cx                      ; salva CX
        or      cl,0ffh                 ; ningun atributo
        test    al,blinking             ; blink puesto ?
        jz      inten                   ; no- exam. intensidad
        and     cl,0bfh                 ; blink bit = 0
inten:  test    al,intens               ; intensity puesto ?
        jz      mode                    ; no- examina modo
        and     cl,0ffh                 ; intensity bit = 0
mode:   and     al,77h                  ; apaga blink e intensidad
        jnz     next2                   ; modo es non display ?
        and     cl,non_dspl             ; display bit = 0
con_fin:        mov     al,cl           ; atributo a AL
        pop     cx                      ; recupera CX y regresa
        ret
next2:  cmp     al,1                    ; modo es subrayado ?
        jne     normal                  ; no- examina modo
        and     cl,no_int               ; si- underline bit = 0
        jmp     con_fin
normal: cmp     al,7                    ; modo invertido ?
        jnz     con_fin                 ; no- ya acabamos
        and     cl,inv_mode             ; mode bit = 0
        jmp     con_fin
attr_conv   endp
;
```

```
; Esta subrutina inicializa el canal 0 del controlador de DMA
; para lectura/escritura o verificacion del disco. La verificacion
; simula una escritura a RAM a la direccion FFFFF .
; El bit 2 de MOTOR STATUS habilita TC para todos los comandos
; exepto FORMAT.
;
; Entradas:    AX = comando al DMA
;              DH = numero de sectores
;         ES:BX = direccion inicial destino/fuente de RAM
;
; Salidas :    CF = 1 se exede una frontera de 64k
;
; AX,BX   destruidos
;
inic_dma  proc  near
;---------- constantes -----------
source_pointer      equ     0ffc0h
fdc_dma_addr        equ     106h
dma_format_high     equ     14h
tc_en               equ     4
destination_point   equ     0ffc4h
dummy_addr          equ     0ffffh
transfer_count      equ     0ffc8h
;
        push    cx                          ; salva: track y sector
        push    dx                          ; cant. de sectores
        push    ax                          ; comando
        push    dx                          ; cant. de sectores
        push    ax                          ; comando
        cli                                 ; no mas interrupciones
        mov     dx,source_pointer           ; llena source y dest.
        mov     cx,2                        ; pointers con la dir. DMA
out:    mov     ax,fdc_dma_addr             ; del FDC
        out     dx,ax
        inc     dx
        inc     dx
        xor     ax,ax
        out     dx,ax
        inc     dx
        inc     dx
        loop    out
        mov     ax,es                       ; calcula direccion inicial
        mov     cl,4                        ; lineal a partir de ES
        rol     ax,cl                       ; y BX
        mov     ch,al
        and     al,0f0h
        add     ax,bx
        jnc     dma_cont
        inc     ch
dma_cont:       mov     bx,ax
        mov     al,ch
        and     ax,000fh
        mov     cx,ax
        pop     ax                          ; recupera comando
        push    bx                          ; salva dir. inicial
        cmp     ah,dma_format_high          ; Es comando de formato ?
        jz      next3                       ; si- no habilites TC
        or      motor_status,tc_en
next3:  rol     ax,1                        ; bit 15 de comando puesto ?
        jc      read2                       ; si- es lectura o verif.
        mov     dx,source_pointer
inic:   mov     ax,bx                       ; inicializa dir. inicial
        out     dx,ax
        inc     dx
        inc     dx
```

```
        mov     ax,cx
        out     dx,ax
        jmp     count
read2:  mov     dx,destination_point
        rol     ax,1                    ; bit 13 del comando puesto ?
        rol     ax,1
        jnc     verify                  ; no= 3 es verificacion
        jmp     inic
verify: mov     bx,dummy_addr           ; establece seudo direcc.
        mov     cl,00fh
        jmp     inic
count:  pop     bx                      ; recupera dir. inicial
        pop     dx                      ; recupera no. de sectores (DH)
        push    bx                      ; salva dir. inicial
        mov     ah,dh
        sub     al,al
        shr     ax,1                    ; AX = no. de sect. x 128
        push    ax                      ; salva ese valor
        mov     bx,6                    ; llama bytes/sector
        call    get_param
        mov     cl,ah
        pop     ax                      ; recupera no. de sect. x 128
        shl     ax,cl                   ; desplaza por bytes/sector
        pop     bx                      ; recupera dir. inicial
        mov     cx,ax                   ; CX = no. total de bytes
        mov     dx,transfer_count       ; inicializa DMA count
        out     dx,ax
        pop     ax                      ; recupera comando e
        inc     dx                      ; inicializa control
        inc     dx
        out     dx,ax
        mov     ax,bx                   ; suma cuenta de bytes
        add     ax,cx                   ; a dir inicial
        pop     dx                      ; recupera no. de sectores
        pop     cx                      ; y track/sector inicial
        sti                             ; habilita interrupciones y
        ret                             ; regresa
inic_dma  endp
;
; subrutina wait int
;
; Esta subrutina genera un tiempo de espera , durante el cual
; verifica el start/stop bit del DMA CTRL WORD. Si la funcion
; terminal count esta habilitada (bit 2 de motor ctrl.) emite
; TC al FDC CMD reg.
; Si el tiempo de espera concluye sin que se produsca una
; interrupcion , CF = 1 y el bit 7 de DISKETTE STATUS es puesto.
;
wait_int  proc  near
;--------- constantes ----------
start_stop_bit      equ     2
not_tc_reset        equ     0f3h
not_int_flag        equ     7fh
;
        sti                             ; hab. interrupciones
        push    ax                      ; salva registros
        push    cx
        push    dx
        push    bx
        mov     bl,2                    ; inicializa contador
        xor     cx,cx
        mov     dx,dma_ctrl_word        ; apunta a DMA CTRL
wait8:  test    motor_status,tc_bit     ; TC habilitado ?
        jz      extra_wait              ; no= compensa tiempo
        in      al,dx                   ; si= lee DMA CTRL
        test    al,start_stop_bit       ; start stop = 0 ?
```

```
        jz      terminal_count          ; si- maneja TC
        nop                             ; mas tiempo
        nop
cont:   test    seek_status,int_flag    ; ocurrio interrupcion ?
        jnz     fin1                    ; si- ya terminamos
        loop    wait8                   ; no- espera
        dec     bl                      ; continua esperando
        jnz     wait8
        or      diskette_stat,time_out  ; fin de tiempo
        stc                             ; prende CF
fin1:   pushf
        and     seek_status,not_int_flag ; limpia bandera
        popf
        pop     bx                      ; recupera todo y
        pop     dx                      ; regresa
        pop     cx
        pop     ax
        ret
extra_wait: jmp cont
terminal_count: push  dx                ; prende bit TC
        mov     dx,fdc_cmd              ; del FDC CMD reg
        mov     al,motor_status         ; aprox. 3.7 useg
        or      al,reset_bit
        out     dx,al                   ; apaga TC enable
        and     al,not_tc_reset         ; en MOTOR STATUS
        mov     motor_status,al
        or      al,reset_bit
        out     dx,al
        pop     dx
        jmp     cont
wait_int  endp
;
; subrutina format
;
; Esta subrutina formatea un track
; la tabla de parametros empieza en la dir 80h de BIOS DATA.
; Como se indica ,formatea track 0, cabeza 0, sector 0-7.
;
format  proc  near
        mov     ax,bios_data
        mov     es,ax
        mov     di,0080h
        mov     cx,8
        mov     dx,200h
store_f: mov    ax,0
        stosw
        mov     ax,dx
        stosw
        inc     dx
        loop    store_f
        mov     ax,500h
        mov     bx,80h
        mov     cx,1
        mov     dx,0
        int     13h
        jmp     examina
format  endp
;
; subrutina examina
;
; Esta subrutina transfiere al simulador de ROM el BIOS DATA AREA
; y todos los registros
;
examina  proc near
        nop
        nop
```

```
        sti
        hlt
        ret
examina  endp

; Tabla de vectores de Interrupcion.
; Contiene los offset respecto el Code Segment
; de las rutinas de servicio.
;
vector_table  label word
        dw      offset  d_eoi
        dw      offset  d_eoi
        dw      offset  dummy_rtn
        dw      offset  d_eoi
        dw      offset  d_eoi
        dw      offset  dummy_rtn
        dw      offset  d_eoi
        dw      offset  d_eoi
        dw      offset  timer0_int
        dw      offset  dummy_rtn
        dw      offset  d_eoi
        dw      offset  d_eoi
        dw      offset  d_eoi
        dw      offset  console_int
        dw      offset  disk_int
        dw      offset  d_eoi
        dw      offset  video_io
        dw      offset  equip_deter
        dw      offset  memory_size_det
        dw      offset  diskette_io
        dw      offset  dummy_rtn
        dw      offset  dummy_rtn
        dw      offset  keyboard_io
        dw      offset  dummy_rtn
        dw      offset  dummy_rtn
        dw      offset  bootload
        dw      offset  time_of_day
        dw      offset  dummy_rtn
        dw      offset  timer_tick
        dw      offset  dummy_rtn
        dw      offset  disk_base
        dw      offset  dummy_rtn
;
; tabla disk base
;
; Contiene los parametros de operacion del Diskette.
;
disk_base  label  byte
        db      6fh
        db      8h
        db      25h
        db      2
        db      9h
        db      2ah
        db      0ffh
        db      50h
        db      0f6h
        db      8h
        db      6
        db      90h
;
; tabla timer param
;
; Contiene los valores de inicializacion de los timers.
```

```
        dw      35a5h
        dw      0e009h
        dw      48h
        dw      0c001h
        dw      6h
        dw      0c001h
;
; parametros del controlador de interrupciones
;
int_contr_table  label word
        dw      0
        dw      0fh
        dw      0fh
        dw      0fh
        dw      0fh
        dw      6h
        dw      0fh
        dw      9090h
;
; tabla del uart
;
; valores de inicializacion del UART
;
uart_table label byte
        db      0
        db      0
        db      0
        db      40h
        db      6dh
        db      16h
;
ultimo:
; mensaje de instalacion del bios huesped
;
promt db '              Bios huesped version 1.0',13,10
      db '              -------------------------',13,10,'$'
;
; rutina de instalacion del bios huesped
;
; Esta subrutina es la encargada de instalar mediante la
; interrupcion 27h del Sistema Operativo al Bios Huesped.
;
instal  proc near
        assume  ds:code
;---- posicionate al principio de la pantalla
;
        mov     ah,2
        sub     dx,dx
        mov     bh,0
        int     10h
;
;---- limpia la pantalla
;
        mov     cx,1840
cls:    mov     ah,14
        mov     bl,0
        mov     al,20h
        int     10h
        loop    cls

        mov     ah,2
        sub     dx,dx
        mov     bh,0
        int     10h
```

```
ch:  mov   ah,14
     mov   bl,0
     mov   al,cs:[si]
     inc   si
     cmp   al,'$'
     je    cnt
     int   10h
     jmp   load_char
carga los vectores de interrupcion del bios huesped <--
     cld
     push  cs
     pop   ds
     sub   ax,ax
     mov   es,ax
     mov   cx,0
vec: mov   bx,cx
     mov   bx,offset int_table[bx]
     mov   si,offset vector_table
     add   si,bx
     add   bx,bx
     mov   di,bx
     cli
     lodsw
     stosw
     mov   ax,cs
     stosw
     sti
     inc   cx
     inc   cx
     cmp   cx,20
     jnz   load_vec                    ; apunta a ultimo byte
     mov   dx,offset ultimo            ; a salvar y regresa control
     int   27h                         ; al Sistema Operativo
table  label  word
     dw  10h        ; timer0_int
     dw  1ah        ; console_int
     dw  1ch        ; disk_int
     dw  20h        ; video_io
     dw  22h        ; equip_deter
     dw  24h        ; memory_size_det
     dw  26h        ; diskette_io
     dw  2ch        ; keyboard_io
     dw  32h        ; bootload
     dw  34h        ; time_of_day
     dw  38h        ; timer_tick
cal  endp
     ends
     end     start
------------>>>>>  FIN  <<<<<-------------------------
```

B I B L I O G R A F I A

1.- Manual de BIOS
generado en UPIICSA — Autores del Proyecto.
1984.

2.- APX86/88 User's Manual
Hardare Reference — Intel (USA).
mayo, 1983.

3.- Microsystems Components
Handbook Volume 1 — Intel (USA).
1984.

4.- Microsystems Components
Handbook Volume II — Intel (USA).
1984.

5.- Application Note Ap-186
"Introduction to the 80186 — Intel (USA).
Microprocessor"
marzo, 1983.

6.- Peripheral Design
Handbook — Intel (USA).
agosto, 1981.

7.- MS-DOS Manual — Micro Soft Corporation
1982.

8.- CP/M-86 Manual — Operating Systems
1983. — Digital Research (USA).

9.- Personal Compater Hardware Reference Library IBM abril, 1983. IBM (USA).

10.- Assembly Language Programing for the IBM Personal Computer David J. Bradley. Prentice Hall, Inc. Englewood Cliffs, New Jersey, USA.

11.- SAH50 Double-Sided Minifloppy Diskette Storage Drive OEM Manual. Hamilton Aunet Austin Texas, U.S.A.

12.- Revista BYTE V. 8 noviembre, 1983.

13.- Specification Control Drawing Keyboard, Mono Encoded Texas Instrument. Dallas, U.S.A.

14.- Computer Architectura and Organization. John P.Hayes 1978. McGraw-Hill, U.S.A.

15.- Teoría de Conmutación y Diseño Lógico. Frederick J. Hill y Gerald R. 1980. Limusa, México.