CENTRO DE INVESTIGACION Y DE ESTUDIOS AVANZADOS

DEL

INSTITUTO POLITECNICO NACIONAL

DEPARTAMENTO DE INGENIERIA ELECTRICA

SECCION DE COMPUTACION

"IMPLEMENTACION DE PASCAL CONCURRENTE PARA MICROCOMPUTADORAS IBM PC"

Tesis que presenta el Ing. Uriel Tirado Rios para obtener el grado de **MAESTRO EN CIENCIAS** en la especialidad de **INGENIERIA ELECTRICA**. Trabajo dirigido por los Doctores Manuel Edgardo Guzman Renteria y Armando Maldonado Talamantes.

Becario del CONACYT

México D. F., Noviembre 1987.

Deseo expresar mis agradecimientos a las siguientes instituciones por la ayuda brindada:

Centro de Investigación y de Estudios Avanzados del Instituto Politécnico Nacional.

Consejo Nacional de Ciencia y Tecnología.

Instituo Mexicano del Petroleo.

Dedico el presente trabajo a mis padres y hermanos que han sido ejemplo de dedicación y esfuerzo, al Dr. Manuel E. Guzmán Rentería por el apoyo y confianza brindados.

I N D I C E

PAGINA

PAGINA

RESUMEN.

RESUMEN DEL TRABAJO

El presente trabajo consiste en el diseño e implementación de las rutinas necesarias para dar al lenguaje de programación secuencial Turbo Pascal, la facilidad de manejar procesos concurrentes.

Este lenguaje permite controlar "hardware" e interrupciones sin la necesidad de trabajar en ensamblador.

El trabajo esta estructurado en tres partes, que son la siguientes:

1.- Implementación de las rutinas básicas para el manejo de co-rutinas. A este módulo se le llamó NUCLEO.

2.- Implementación de un manejador de procesos junto con las herramientas necesarias para su comunicación y sincronización. A este módulo se le llamó KERNEL.

3.- Implementación de un manejador de disco flexible, como base para el diseño posterior de un manejador de archivos.

La primera parte es desarrollada en lenguaje ensamblador para lograr el manejo de recursos que no son manejables desde Pascal , como es el caso de los registros mismos del procesador. Estas rutinas son compiladas por separado y declaradas como externas desde módulos en Pascal.

La idea de este modulo es implementar el manejo de co-rutinas que son la base para la creación de procesos. En este nivel podemos ya conmutar de una co-rutina a otra mediante una primitiva llamada transfer y podemos también asociar una co-rutina a una interrupción mediante la primitiva iotransfer.

La segunda parte esta escrita en Pascal y permite tener un administrador de recursos como lo es el mismo procesador. Implementa herramientas de sincronización y comunicación entre procesos, como los son los semáforos, eventos, mensajes y monitores. Los procesos son administrados en base a colas dependiendo del estado del proceso (suspendido, listo para ejecución o ejecutándose), estas colas son formados de acuerdo a las prioridades asignadas a los procesos. Es posible asociar un proceso a una interrupción dada, pero solo un proceso puede esperar por esa interrupción a la vez.

Por último se implementa un manejador de disco flexible, que nos permite leer o escribir un bloque de 1024 bytes en disco. Sobre él se puede construir posteriormente un manejador de archivos. La razón de este manejador de disco se debe a que las rutinas normales de entrada/salida del sistema operativo DOS no son "reentrantes", por lo que es necesario construir nuevas rutinas que si lo sean.

INTRODUCCION.

INTRODUCCION

Todas las computadoras modernas pueden realizar varias tareas al mismo tiempo. Mientras ejecutamos un programa, el computador puede también estar accesando algún disco y desplegando en una terminal o pantalla. En los sistemas de multiprogramación , la CPU también conmuta de un proceso a otro , ejecutando cada uno por décimas o centésimas de segundo. Si se habla estrictamente, la CPU solamente ejecuta un programa a la vez, pero en el curso de un segundo puede trabajar con varios programas, dando así al usuario la ilusión del paralelismo. A la rápida conmutación de la CPU entre programas algunas veces se le llama pseudoparalelismo para diferenciarla del paralelismo que se da por "hardware" cuando el procesador trabaja mientras uno o más dispositivos de entrada/salida realizan su función.

Un concepto clave en todos los sistemas operativos concurrentes es el proceso. Un proceso es basicamente un programa en ejecución. Este esta constituido por código ejecutable, un área de datos y una pila, su propio contador de programa, su apuntador de pila y otros registros, y toda la información necesaria para que el programa se ejecute.

Cuando un proceso es suspendido temporalmente, se espera que sea reactivado exactamente en el mismo estado cuando se detuvo. Esto significa que toda la información acerca del proceso debe ser explícitamente guardada en algún lugar durante la suspensión. En muchos sistemas operativos, toda la información de cada proceso, es guardada en una tabla del sistema operativo llamada "tabla de procesos", la cual es un arreglo o una lista de estructuras, perteneciendo cada estructura a un proceso existente.

Así un proceso (suspendido) consiste de un apuntador al área donde se guardaron sus parámetros y su respectiva entrada en la tabla de procesos.

Un solo procesador puede ser compartido por varios procesos, en base a algún algoritmo de selección usado para determinar cuando detener un proceso para poner en funcionamiento otro diferente.

Una manera natural de introducirse al concepto de concurrencia es empezar con un lenguaje secuencial como Pascal y añadir las rutinas necesarias para manejar procesos sin alterar el compilador o las bibliotecas de soporte. Debido a que cuando se realiza la conmutación de procesos deben manipularse los registros del

procesador, las rutinas añadidas deberán implementarse en ensamblador.

Si el compilador de Pascal permite procedimientos externos, es posible aislar las operaciones de conmutación de contexto que no puedan ser escritas en Pascal y escribirlas en lenguaje ensamblador. En el capítulo III se describe un juego de rutinas escritas en ensamblador a las cuale se les ha llamado NUCLEO y son similares a las primitivas usadas en MODULA-2.

Las primitivas, operaciones indivisibles, del NUCLEO son de bajo nivel y no se pretende que se utilicen directamente en programas de aplicación por lo que en capítulo IV se desarrollaron primitivas de mayor nivel para programación en tiempo real, éstas fueron escritas en Pascal y se les llamó KERNEL EN TIEMPO REAL. Es posible también usar el NUCLEO para probar nuevas primitivas para programación concurrente. El KERNEL implementa herramientas de sincronización como semáforos, eventos y mensajes.

Debido a que las rutinas de entrada/salida del sistema operativo MS-DOS no fueron diseñadas para ser usadas por procesos, se implementaron rutinas para manejo de display y se implementó en el capítulo VI un manejador de disco que puede servir de base para un manejador de archivos.

IMPLEMENTACION DE CO-RUTINAS EN TURBOPASCAL

IMPLEMENTACION DE CO-RUTINAS EN TURBOPASCAL

En el presente trabajo se han implementado las primitivas necesarias para dar la característica de concurrencia a Turbo Pascal versión 3.0 (corre en computadores personales compatibles con IBM bajo versiones de sistemas operativos 2.0 o superiores).

Un concepto clave para la creación de procesos es la co-rutina, la co-rutina a diferencia de los procedimientos, no necesariamente debe ejecutarse de principio a fin cuando se le llama. Por ejemplo en un primer llamado pueden ejecutarse las tres primeras líneas de su código y en ese momento regresar al procedimiento llamador, en una segunda llamada a la cor-rutina se continuará con la ejecución de su cuarta línea, sin tener que pasar por las tres líneas anteriores. Usando este mismo concepto es posible hacer que procedimientos se comporten como procesos.

Como se mencionó en la introducción las primitivas fueron implementadas en ensamblador por lo que se considera pertinente examinar los detalles de "hardware" y ensamblador necesarios para comprender la implementación del NUCLEO.

ARQUITECTURA BASICA DE LA MICROCOMPUTADORA IBM-PC.

El microprocesador usado en la IBM-PC fue diseñado y construido por INTEL y es el Intel 8088, el cual es ligeramente diferente al 8086 del mismo Intel. Los procesadores 8086 y 8088 realizan las mismas instrucciones y desde el punto de vista del programador son idénticos.

Ahora explicamos la diferencia entre el microprocesador 8088 y 8086. El funcionamiento de ellos es el mismo, realizan las mismas operaciones . Pero cuando hablamos de lo que hay alrededor aparece la diferencia, el 8086 se conecta con circuitería que maneje 16 bits de datos a la vez, mientras que el 8088 lo hace pasando al exterior 8 bits a la vez. La diferencia pues entre estos dos procesadores es el bus de datos externo que manejan. Esto nos haria pensar que el 8088 no es completamente un procesador de 16 bits, lo cual es cierto pero no totalmente. El 8088 usa internamente una arquitectura de 16 bits. pero se comunica al exterior con un bus de 8 bits.

La diferencia del bus de datos del 8088 y el 8086 nos indica que el 8086 pasará dos palabras por una que pase el 8088, lo cual no significa necesariamente que el 8086 trabaje dos veces más rápido que el 8088, ya que unicamente una parte de tiempo es usada por el

procesador para esperar datos del medio externo y a veces solamente requiere de 8 bits.

Una segunda diferencia práctica es el diseño de circuitos y selección de componentes ya que es más fácil diseñar circuitos de 8 bits y existe mayor variedad de componentes a menor precio. De esta manera IBM simplificó el diseño y redujo el costo sacrificando algo de velocidad en procesamiento.

El procesador 8088 consta de dos unidades separadas de procesamiento: La unidad de ejecución (EU), la cual se encarga de ejecutar instrucciones y la unidad de interface de BUS (BIU), la cual es responsable de la comunicación del 8088 con el mundo exterior . La "EU" proporciona una dirección lógica al "BIU" , el cual la convierte a una dirección física . Esta operación llamada cálculo de dirección física, usa dos parámetros de 16 bits: un registro de segmento (un segmento es un bloque de 64 Kbytes) y un desplazamiento (offset) . La notación lógica usada esta dada por segmento:desplazamiento. Los registros de segmento (que forman parte del BIU) son : (CS) segmento de código, (DS) segmento de datos, (SS) segmento de stack, (ES) segmento extra. EL desplazamiento usualmente lo proporciona el EU.

Para calcular la dirección física , el 8088 realiza un corrimiento a la izquierda de cuatro bits sobre el registro de segmento y lo suma al desplazamiento en el sumador dedicado del BIU.

Los segmentos son bloques de memoria de 64 Kbytes relocalizables dentro de un bloque físico de 1 Mbyte de memoria y pueden traslaparse.

La dirección de la siguiente instrucción a ejecutar se encuentra en el par de registros CS:IP. Para incrementar eficientemente, el BIU guarda bytes en una cola (realiza búsqueda de instrucciones anticipada). Para facilitar esto el registro IP es guardado en el BIU.

EL EU contiene registros de 16 bits cualquiera de los cuales puede ser usado en cálculos. Cuatro registros forman este grupo. Estos son acumulador (AX), base (BX), contador (CX), y registro DX. EL 8088 puede accesar los 8 bits más significativos o los 8 bits menos significativos de estos registros de datos. Las dos mitades del registro AX son por ejemplo AH y AL.

Los dos siguientes registros generales, el apuntador de pila (stack pointer SP) y el apuntador al área de variables locales (base pointer BP), constituyen el grupo de apuntadores. Estos registros manipulan la pila. Cuando una subrutina es llamada SS:SP almacena la dirección de regreso en la pila. SP apunta al tope de la pila y BP a la base. SP se decrementa automáticamente por una llamada a una subrutina y se incrementa por un regreso de ésta. La pila también es usada para pasar parámetros de subrutinas y con la ayuda de BP se accesan estos parámetros.

Otros dos registros generales , el registro índice (SI) y el registro destino (DS) completan el grupo de apuntadores y son usados generalmente para manejo de cadenas de caracteres.

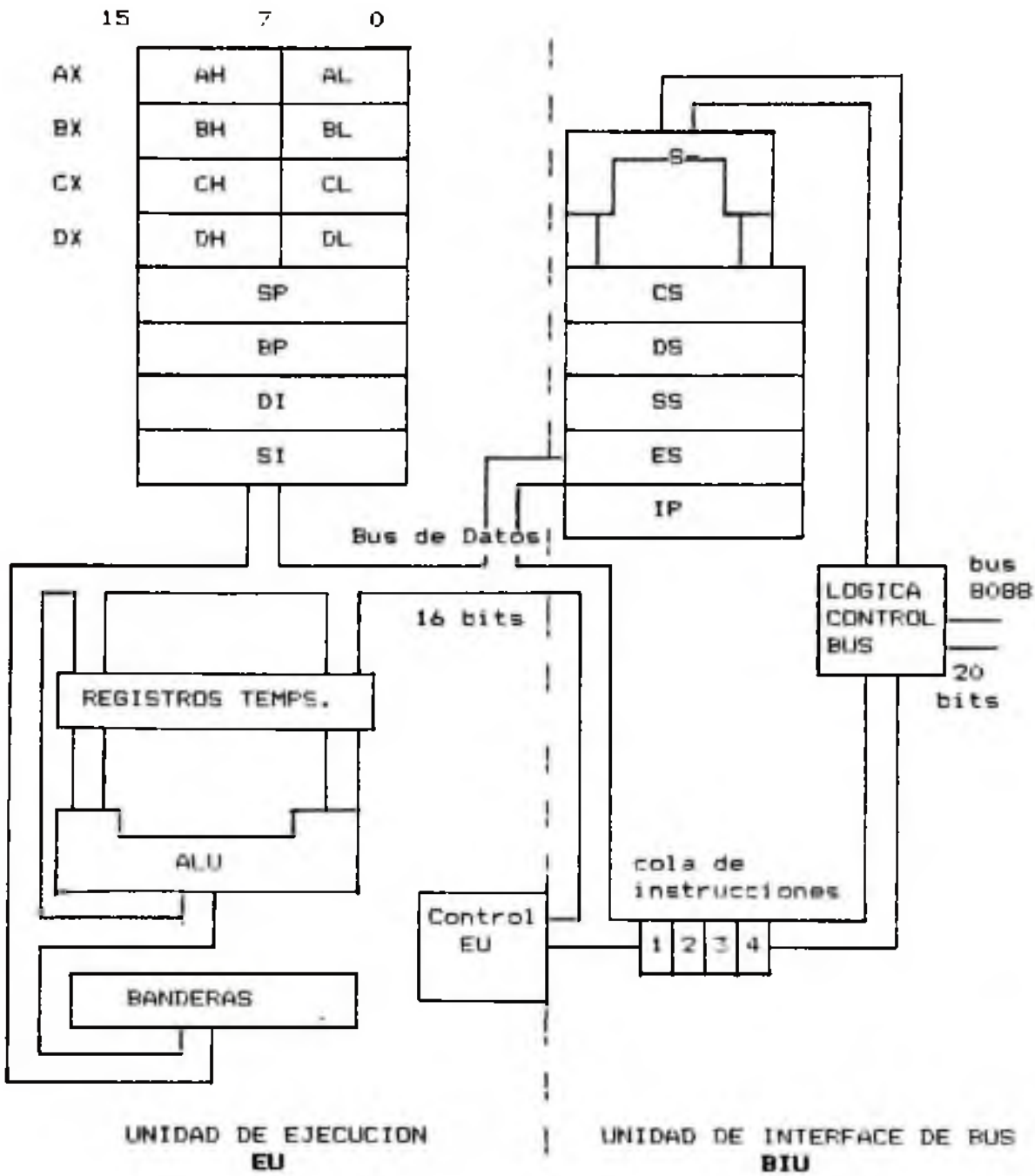


FIG. III.1 DIAGRAMA ELEMENTAL A BLOQUES DEL 8088

El hacer que un microprocesador trabaje requiere del auxilio de muchos otros componentes, por lo que daremos un breve panorama de los circuitos integrados importantes de la PC.

Las señales básicas de reloj son provistas por el generador de reloj **8284**. Estas señales son usadas por el sistema completo para controlar la duración de las operaciones. Relacionado con el reloj tenemos un circuito integrado temporizador **8253** y un **8255A5** el cual es usado para controlar la interface de "cassette" y la bocina. Para controlar las interrupciones del "hardware" se utiliza un **8259A**. Para transferir datos en el sistema se hace uso de un BUS común, por lo que es necesario un árbitro de acceso al BUS, esta función se realiza con un controlador **8288**.

Todos los circuitos mencionados anteriormente se localizan en la tarjeta madre. Ahora cuando analizamos las tajetas de las ranuras de expansion encontramos otros dispositivos interesantes. Existen dos tipos de adaptadores de "display". Uno diseñado para controlar monitores monocromáticos y otro para manejar monitores gráficos de color. Aunque los dos tipos de monitores operan de manera diferente y tienen diferentes capacidades, se utiliza el mismo circuito integrado para controlarlos. El controlador Motorola **6845** CRT es la parte principal de ambos controladores.

Una parte importante de una computadora es la forma en que maneja su memoria. La memoria de la computadora esta organizada en muchas localidades de memoria donde pueden guardarse datos en general, cada localidad de memoria es identificada por una dirección. La mínima localidad de memoria direccionable en la PC es el byte que consta de 8 bits. Ya que el byto es el tamaño exacto de memoria para almacenar un carácter, los termino byte o carácter son usados en forma indistinta.

Cada localidad de memoria tiene una dirección asociada para accesarla y empieza desde 00000 para la primera localidad hasta la localidad más alta que es precisamente el tamaño de memoria máximo de la computadora. Normalmente las microcomputadoras tiene menos memoria que la que son capaces de direccionar.

La IBM-PC hace uso del espacio total direccionable. Para el 8088 las direcciones se forman con 20 bits así que el procesador puede direccionar 1024 kbytes o sea más de un millón de bytes.

Como se vio anteriormente los registros del microprocesador son de 16 bits , con los cuales solo podemos direccionar hasta 64 kbytes de memoria, asi que para acompletar los 20 bits de direcciones se debe de usar algún método práctico. La solución fue dada con lo que se conoce como direccionamiento segmentado.

Si se toma un número de 16 bits , y se agregan cuatro ceros binarios a su derecha, se tendra un número de 20 bits, el cual puede ser usado como una dirección de 20 bits, en realidad hemos

multiplicado nuestro número por 16 y su rango ahora puede alcanzar hasta 1024 kbytes. Para completar la solución, se utilizan dos números de 16 bits. En el primero se considera que sus últimos cuatro dígitos binarios son cero, de tal manera que se formen los 20 bits, este número es llamado el segmento de la dirección . El segundo número se deja tal cual y se le llama desplazamiento. Sumados ambos números dan la dirección física dentro del área de memoria. El segmento especifica una localidad que es un múltiplo de 16 bits que es llamada frontera a párrafo.

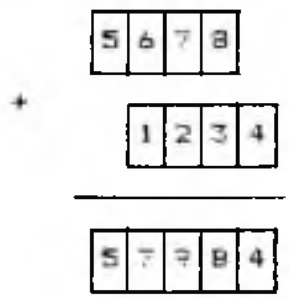
Ejemplo:

registro ordinario de 16 bits desplazamiento

16 bits = 4 dígitos hexadecimales
1 dígito hexadecimal = 4 bits

registro de segmento

registro de desplazamiento y registro de segmento combinados



Resultado de 20 bits

FIG. III.2 CALCULO DE DIRECCION FISICA.

Para hacer uso de estos segmentos, el 8088 tiene un juego especial de registros dedicados a guardar la parte base del segmento. Con algún valor en un registro de segmento, es posible direccionar cualquiera de las 64 kbytes localidades de memoria que le siguen. Si no se modifica el registro de segmento , la computadora tiene un espacio temporal de trabajo de 64 kbytes, localizado dentro del espacio total de memoria de 1024 kbytes.

Para que sea posible trabajar con más de 64 kbytes a la vez, el 8088 usa los cuatro registros de segmentos, cada uno de ellos con un uso especial . Una computadora usa su memoria para diferentes propósitos, uno de los cuales es contener los programas y datos que estos manejan. Así que dos de los segmentos estan dedicados al manejo de código y datos. Para el segmento de datos se usa el registro DS. Para el código del programa se usa el CS. Para el segmento de pila se usa el registro SS. Finalmente se provee un cuarto registro de segmento para el manejo de un segemento extra llamado ES.

Debe quedar claro que estos cuatro segmentos no tienen que referirse forzosamente como áreas de memorias separadas. Pueden estar en cualquier parte, cerca o lejos, o aún traslapadas.

Una computadora debe ser capaz de responder a eventos que suceden fuera de sus cálculos, por ejemplo enterarse de que se tecleo. Hay dos maneras de checar esto, una es examinando constantemente si el evento ocurrió. Este método es conocido como "POOLING" pero con este método el procesador gasta mucho de su tiempo checando si el evento ocurrió. El segundo método permite al procesador seguir con su trabajo a menos que algo en el exterior requiera de su atención, este método es conocido como "INTERRUPCION".

Algunos ejemplos de interrupciones se encuentran en el teclado, donde cada vez que se presiona una tecla se produce una interrupción. Otra más es el caso del tick (INT 08H) del reloj, que es construido en la PC. Un ejemplo más de uso de interrupción lo encontramos en el manejador de disco que produce una interrupción (INT 0EH) para indicar que se ha completado una operación del "diskette".

Algunas veces es importante que el procesador no sea interrumpido en su trabajo, debido a que se encuentra haciendo algo crítico. Para permitir esto el 8088 tiene una instrucción que permite temporalmente deshabilitar interrupciones y otra instrucción para volverlas a habilitar. Cuando se deshabilitan interrupciones no implica que estas se pierdan, estas son conservadas para cuando se habiliten interrupciones nuevamente.

El mecanismo bajo el cual trabajan las interrupciones es el siguiente: Cada interrupción tiene un número asociado a ella, por ejemplo la del reloj es la interrupción número 8. Almacenada en las direcciones más bajas del mapa de memoria existe una tabla de direcciones de los programas que serán activados cuando las diferentes clases de interrupciones ocurran. Estas direcciones deben ser con segmento, así que se requieren dos palabras para cada dirección. Estas direcciones son conocidas como vectores de interrupción. La interrupción 0 tiene su vector de interrupción almacenado en la dirección 00000, la interrupción 1 en la localidad 00004 y así sucesivamente. Cuando la interrupción X se presenta, el vector X*4 es cargado dentro de los registros CS e IP del procesador, y el procesador empieza a ejecutar la rutina de servicio de interrupción que esta cargada en esta dirección.

Cuando se ha completado la rutina de interrupción, se regresa el control al programa que estaba ejcutandose, usando una instrucción de regreso especial llamada IRET. Para hacer esto posible, antes de que el vector de interrupción sea cargado, los registros de direcciones CS:IP son salvados usando la pila.

Existen tres clases de interrupciones, las cuales llamaremos por "hardware", lógicas y por "software".

Las interrupciones por "hardware" son generadas por algún equipo que demanda atención. Existen sorprendemente pocas en la PC. Primero existe la llamada no mascarable, usada para indicar una falla de alimentación; esta interrupción es la número 2. Las siguientes interrupciones son: la 8 para el reloj, la 9 para el teclado y la 14 para el manejador de diskette. Existen otras sieto interrupciones reservadas, 6, 7, 10 a 13 y 15 que pueden ser usadas para futuras necesidades.

Las interrupciones lógicas son generadas por el mismo 8088, cuando encuentra situaciones anormales. Existen cuatro de ellas. La interrupción 0 es generada cuando el 8088 encuentra un intento de dividir entre cero. La interrupción 1 es usada para operar el procesador paso a paso, ejecutando una instrucción a la vez; esta interrupción se usa para prueba de programas. La interrupción 3 también se usa para prueba de programas y es generada por la instrucción "break-point". La interrupción 4 es generada por una condición de sobreflujo, por ejemplo cuando una instrucción aritmética produce un resultado demasiado grande para que quepa en los registros.

Las interrupciones por "software" son las más interesantes. La idea de estas es tener la capacidad de invocar una subrutina y regresar el control cuando se haya ejecutado, sin tener la necesidad de conocer las direcciones de las subrutinas.

Las interrupciones por "software" son usadas por los servicios que se quiera esten disponibles al usuario.

Existen dos razones porque son preferidas las interrupciones en lugar de llamados a subrutinas. La más importante es que permite que las rutinas a ser llamadas se modifiquen cuando sea necesario. Otra razón es que pueden substituirse cuando se requiera.

El uso de pilas es una de las características más importantes e interesantes de las computadoras modernas. Junto con las interrupciones las pilas hacen que una computadora sea eficiente.

Que es un pila?. Un pila (stack) es un lugar donde la computadora guarda sus notas de trabajo de tal manera que un juego de notas no interfiera con otro.

Cuando una computadora se encuentra trabajando, y recibe una interrupción, necesita un lugar donde guardar los registros con que estaba trabajando. Si recibe otra interrupción mientras estaba procesando la primera, necesita un lugar donde guardar el estado de ésta. Y cuando la segunda interrupción es atendida, la computadora regresa a seguir haciendo la cosa más reciente que se había suspendido. La pila es el mecanismo natural para que la computadora recuerde lo que estaba haciendo.

La forma en que trabaja la PC es tomando una porción de memoria para uso de pila, y se emplea un registro especial do segmento para indicar el área donde esta localizado esta pila. El tope de la pila es llevado por un registro llamadc apuntador de pila (stack pointer) o SP. Cuando se guarda un datc en la pila, el apuntador de pila es quien se actualiza.

Los datos se guardan en la pila mediante la operación PUSH y son removidos mediante la operación POP.

Cuando ocurre una interrupción, la dirección actual del programa guardada en CS e IP se guarda en la pila, y entonces la dirección de la rutina de interrupción es cargada en CS:IP para su ejecución. Sobre el apuntador de pila están todos los trabajos que han sido suspendidos esperando su reactivación. Adelante del apuntador de pila se abre un espacio para el área de trabajo de la rutina de servicio.

Cuando cada rutina que se llamo se termino , se vacía la pila. Por otro lado cualquier otra área de trabajo también es vaciada de la pila y los valores anteriores de CS:IP son restaurados.

La pila es usada no solo para manejo de interrupciones, también se usa cuando un programa llama a otro. Para llamados a interrupciones el principio es el mismo. En el proceso de llamar una subrutina es muy común que los parámetros sean pasados a través de la pila. Las pilas crecen de localidades altas a bajas, esto significa que las localidades altas de la pila contienen las primeras entradas.

Los puertos son el mecanismo que usa el microprocesador 8088 para simplificar y unificar la forma de comunicarse con el exterior. Los puertos son la única forma en que el 8088 puede pasar o recibir datos de otro lado que no sea la memoria.

Cualquier cosa que el microprocesador quiera decir al manejador de disco, al teclado, a la bocina, etc. lo hace a través de puertos. Un puerto es una camino hipotético que tiene un número asignado.

Los puertos pueden ser usados junto con las interrupciones. Por ejemplo cuando se presiona una tecla en el teclado, se genera la interrupción número 9 indicando que existe un dato disponible en el teclado. En respuesta a la interrupción, las rutinas del BIOS (rutinas básicas de entrada salida de la PC) hacen una instrucción IN en el puerto del teclado, solamente hasta entonces indicamos que tecla se presionó.

La dirección de un puerto es especificada con 16 bits, de tal manera que tenemos potencialmente hasta 64 kpuertos, pero la PC solo utiliza unos cuantos en la actualidad.

Una vez dadas la bases necesarias del "hardware" en que se desarrollará el trabajo, se describe a continuación un conjunto de procedimientos llamados NUCLEO que permiten manipular procesos en Turbo Pascal.

LAS PRIMITIVAS DEL NUCLEO.-

Cada proceso cuenta con su propia pila (stack). Toda la información necesaria para reanudar un proceso suspendido es almacenada en esta pila. De aqui que todo lo que se necesite conocer de un proceso es precisamente la localización de su pila. Process es implementado como Type **Process = ^ integer**.

La primitiva **Newprocess** usa la función Getmem para obtener un área de memoria donde crear la nueva pila del proceso, el sistema de ejecución de Turbo Pascal obtiene memoria para crear variables dinámicas por medio de Getmem. El arreglo predefinido MemW y las funciones Seg y Ofs son entonces usadas para modificar la nueva pila.

El procedimiento **resume** hace posible suspender la ejecución del proceso que esta ejecutandose y reanudar uno suspendido. Además un proceso puede esperar por una interrupción especifica a través del procedimiento **ioresume**, así cuando la interrupción ocurre el proceso es reactivado.

Los dos procedimientos **disableinterrupts** y **enableinterrupts** se utilizan para realizar operaciones indivisibles.

Los procedimientos del NUCLEO fueron compilados por separado y son declarados como procedimientos externos (ver en apéndice A el módulo NUCLEO.EXT).

A continuación describiremos la sintáxis de las primitivas del NUCLEO y mostraremos su uso en programas sencillos. Posteriormente se explicará como fueron implementadas.

INICIALIZACION.-

Antes de poder hacer uso de las primitivas del NUCLEO deberá inicializarse el NUCLEO mediante un llamado a la primitiva **initnucleus**. No existe un pre-examinador del código asi que el orden y la sintáxis del llamado a las primitivas del NUCLEO es responsabilidad del programador.

procedure **initnucleus**(ofnuc : integer; var p : process);

donde:

ofnuc - es el desplazamiento del NUCLEO dentro del código de Pascal.

p - es la variable tipo proceso (process) asociada al programa principal.

Este procedimiento hace que el programa principal actue como un proceso asociado a la variable p. Se pasa como párametro el desplazamiento del NUCLEO dentro del módulo de Pascal porque en el NUCLEO se construyen llamados a direcciones absolutas, las cuales hay que calcular sumando este desplazamiento al de la primitiva del NUCLEO a llamar. En el caso del programa principal no se requiere un estimado de memoria para pila ya que este utilizara la memoria restante para pila que no halla sido asignada a la hora de crear nuevos procesos.

CREACION DE PROCESOS.-

Para crear un proceso se hace uso de la FUNCION **newprocess**. la cual tiene la siguiente sintáxis:

function **newprocess**(proc, memreq : integer) : processref ;

donde:

proc - es el desplazamiento del procedimiento a ser convertido en proceso
memreq - es el área requerida en bytes para la pila del proceso a crear y debe ser un número par mayor de 100.

Esta FUNCION crea un proceso con sus propias pilas (heap y stack). El código a ser ejecutado esta apuntado por proc. El proceso creado nunca deberá alcanzar su instrucción final. El proceso así creado podrá ser activado por llamados a las primitivas del NUCLEO **transfer** o **iotransfer**.

Para calcular el tamaño de la pila en bytes deberá conciderarse lo siguiente:

- Area para paso de parámetros y direcciones de regreso, en el caso de que el proceso llame nuevos procedimientos.
- Area para declaración de variables locales.
- Area para creación de variables dinámicas, si es que el proceso creará nuevas variables.
- Finalmente conciderar que un proceso que sea sorprendido por una interrupción deberá salvar todos sus registros y variables de control en su pila que ocupan aproximadamente 40 bytes.

Se recomienda no crear proceso con pilas menores de 100 bytes, aunque no manejen variables dinámicas o no llamen nuevos procedimientos.

El mismo procedimiento puede ser utilizado para diferentes llamados de **newprocess** con diferentes variables de proceso.

Un proceso deberá crearse a partir de procedimientos sin parámetros. Si fuera necesario pasarlos, deberan crearse un procedimiento con los parámetros requeridos y un proceso desde donde se llamará el procedimiento con estos parámetros.

Un process puede ser declarado en cualquier nivel y por lo tanto un proceso puede ser creado en cualquier lugar, de donde pueda ser llamado en forma ordinaria. Para crear "heaps" independientes se usa la variable pública del compilador "heapptr", que indica la posición actual del heap, así que basta definir valores de esta variable para cada proceso.

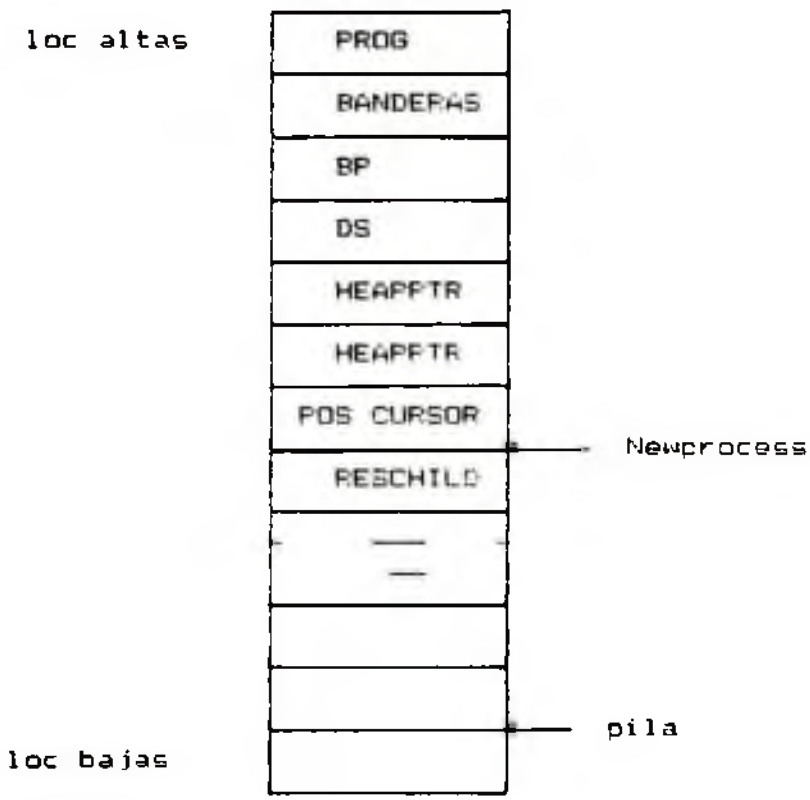


FIG. III.3 PILA DE PROCESO CREADO.

REACTIVANDO LA EJECUCION DE UN PROCESO.-

La ejecución de **transfer** reemplaza la pila en uso por la del proceso a transferir y salva el apuntador de la pila anterior en una variable. Al salir de Transfer, el control del procesador lo tendra el nuevo proceso y el proceso anterior podrá retomar el control a partir de donde se suspendió con un nuevo llamado a la primitiva Transfer.

procedure **transfer**(var p1:process);

donde:

p1 - Es el proceso que se reactivará.

La primitiva transfer (p1) deberá ser ejecutada unicamente cuando ya haya sido creado el proceso p1, de no hacerse así el procesador tratará de ejecutar código inexistente.

MANEJO DE INTERRUPCIONES.-

El procedimento **iotransfer** hace que el proceso en ejecución se suspenda esperando una interrupción especifica, debe tomarse en cuenta que solo un proceso a la vez puede esperar por una misma interrupción y esto deberá garantizarlo el programador.

procedure **iotransfer**(numint : integer; var p1 : process);

donde:

numint - Es el número de interrupción esperada por el proceso.
p1 - Es el proceso que será reactivado.

Cuando se llama a la primitiva **iotransfer** el proceso en ejecución se suspende de la misma manera que en la primitiva resume y se realizan los preparativos necesarios para manejar la interrupción deseada. Al salir de **iotransfer** se reactivará la ejecución de p1.

El NUCLEO contiene dos procedimientos los cuales modifican el estado del procesador con respecto a las interrupciones mascarables y son:

procedure **enableinterrupts;**

procedure **disableinterrupts;**

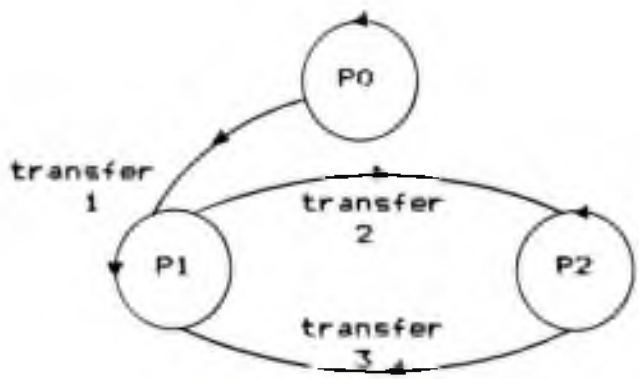
El primer procedimiento habilita interrupciones y el segundo las deshabilita.

EJEMPLOS.

Ejemplo 1.

El siguiente ejemplo nos muestra como usar las primitivas del NUCLEO para crear procesos y transferir el control entre ellos. Observe el uso del nuevo tipo de datos **process** y la forma sincrona de transferir el control entre prog1 y prog2.

Esquema de procesos del ejemplo 1:



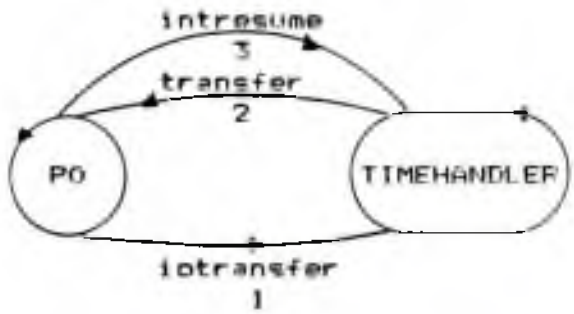

```
{*************************************************************************}
{*                      PROGRAMA DE DEMOSTRACION NUMERO 1                *}
{*    Se demuestra el uso de un núcleo para creación de procesos,        *}
{*la conmutación se lleva expresamente por la primitiva TRANSFER          *}
{*************************************************************************}

{$K-}   {No cheques estructura de la pila}
Program Multitest;
{$I nucleo.ext}                 {rutinas para procesos concurrentes}

var p0,p1,p2: process;          {variables de procesos}
    answer:string[2];

{-------------------------------------------------------------------}

Procedure prog1;                {procedimiento a convertir a proceso}
begin
while true do
   begin
   writeln('hi');
   transfer(p2);                {conmuta a proceso p2}
   writeln('he');
   transfer(p2);                {conmuta a proceso p2}
   end;
end;
```

```
{-------------------------------------------------------------------}

Procedure prog2;
begin
while true do
   begin
   writeln('ho');
   transfer(p1);                {conmuta a proceso p1}
   end;
end;

{-------------------------------------------------------------------}

Procedure main;
begin
while answer <> 'si' do
   begin
   write('empezamos ?'); readln(answer);
   end;
clrscr;
initnucleus(ofs(nuc),p0);
p1:=newprocess(ofs(prog1),1000);   {crea proceso p1}
p2:=newprocess(ofs(prog2),1000);   {crea proceso p2}
transfer(p1);
end;

begin
main
end.
```

Ejemplo 2.

En el segundo ejemplo observamos como es posible asociar un proceso a una interrupción con ayuda de la primitiva **iotransfer**. El proceso incrementer lleva una cuenta de los ticks generados por el temporizador (el tick es la unida minima de tiempo manejada por el 8253 y es aproximadamente igual a 50 mseg) de la microcomputadora, nótese que en este proceso incrementer no se llamó ninguna rutina de Turbo Pascal para entrada/salida.

Esquema de procesos del ejemplo 2:




```
{*************************************************************************}
{*                      PROGRAMA DE DEMOSTRACION NUMERO 2                *}
{*    Se demuestra el uso de un núcleo para creación de procesos.        *}
{*Un proceso puede esperar una interrupción a través de la primiti       *}
{*va IOTRANSFER                                                          *}
{*************************************************************************}

{$K-}   {No cheques estructura de la pila}
Program interrupttest;
{$I nucleo.ext}                    {rutinas de procesos concurrentes}

var count: integer;
    p0,timehandler:process;

{*************************************************************************}

procedure incrementer;
begin
while true do
  begin
  iotransfer($1c,p0);              {esperando interrupción del reloj}
  count:=succ(count);
  end;
end;
```

```
{*************************************************************************}

begin
clrscr;
initnucleus(ofs(nuc),p0);                    {inicializa nucleo}
timehandler:=newprocess(Ofs(incrementer),1000);  {crea proceso}
count:=0;
transfer(timehandler);                       {reactiva timehandler}
while true do
     begin
     gotoxy(25,12);
     writeln('main ',count);
     end;
end.
```

Ejemplo 3.

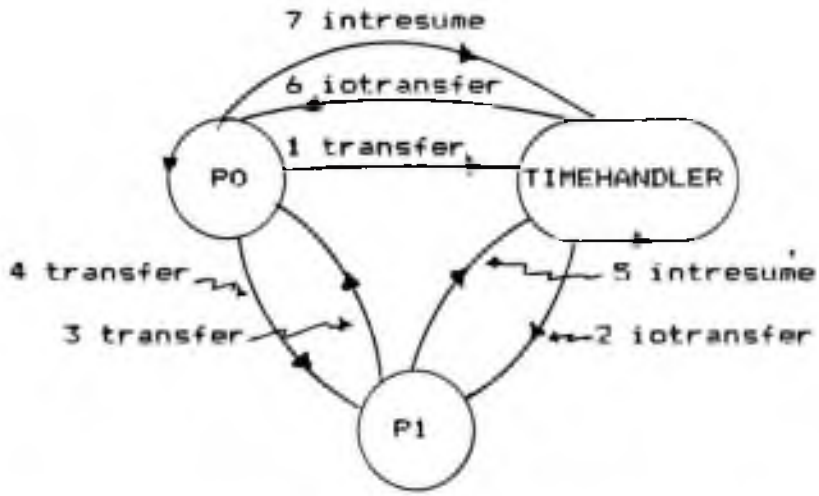
El código generado por Turbo Pascal es completamente interrumpible de tal manera que no enmascara interrupciones, así que se pueden escribir procedimientos para manejar interrupciones en Turbo Pascal.

Una rutina de interrupción no deberá emplear ninguna operación de entrada/salida usando los procedimientos estandares de Turbo Pascal ya que BDOS (interface entre Turbo Pacal y MS-DOS) no es reentrante.

En el tercer ejemplo se demuestra que puede transferirse el control del procesador desde un proceso asociado a una interrupción a otro proceso. Sin embargo como se mencionó en el párrafo anterior no es confiable usar rutinas de entrada/salida en un manejador de interrupciones.

Si se observa detenidamente el código del proceso incrementer, en este no se usan las rutinas de entrada/salida, sin embargo el hecho de regresar el control del procesador a un proceso diferente al interumpido puede causar problemas. Por ejemplo supongamos que el proceso principal asociado a la variable p1 se encontraba escribiendo cuando lo sorprendió la interrupción del reloj, en ese momento se conmuta contexto para que el proceso incrementer atienda la interrupción, como éste no hace uso de las rutinas de entrada/salida no existe ningún problema, pero una vez realizada su función (llevar la cuenta de ticks), transfiere el control de procesador ya no al proceso principal que estaba escribiendo sino al proceso escribe que trata de escribir y como el principal no terminó de escribir se produce el conflicto por tratar de usar las rutinas de entrada/salida que no son reentrantes. Pero ¿qué significa que un procedimiento no sea reentrante? , que el código de este procedimiento no puede ser usado por varios procesos a la vez, ya que un segundo llamado a este procedimiento destruirá el estadc que guardaba en el primer llamado (probablemente destruirá el juego de variables que manejaba para el primer proceso).

Esquema de procesos del ejemplo 3:




```
{*************************************************************************}
{*                      PROGRAMA DE DEMOSTRACION NUMERO 3                *}
{*    En este programa se demuestra que el hecho de usar rutinas  *}
{* de entrada/salida de las bibliotecas de Pascal en procesos aso-*}
{* ciados a interrupciones no es recomendable ya que estas rutinas*}
{* no son reentrantes, es decir no se asegura que puedan ser usa -*}
{* das por dos procesos al mismo tiempo.                          *}
{*************************************************************************}

{$K-}   {no cheques desbordamineto de pila}
Program interrupttest;
{$I nucleo.ext}      {rutinas de procesos concurrentes}
{$I misceláneos}

type
       processref =  processrec;
       processrec = record
          suc,pre,remite  : processref;
          proc     : process;
          end;

var count1: integer;
    timehandler,p0,p1,running:processref;
    answer:string[2];

{*************************************************************************}
{*   PROCEDURE STATUS8259                                                *}
{*   Muestra el contenido de los registros del controlador de inte*}
{*   rrupciones.                                                         *}
{*   IMR = máscara del 8259 (0 habilita int, 1 enmascara int)      *}
{*   IRR = registro de peticiones de interrupción, 1 = pidiendo    *}
{*   interrupción                                                        *}
{*   ISR = registro de interrupciones siendo atendidas, 1 = sir -  *}
```

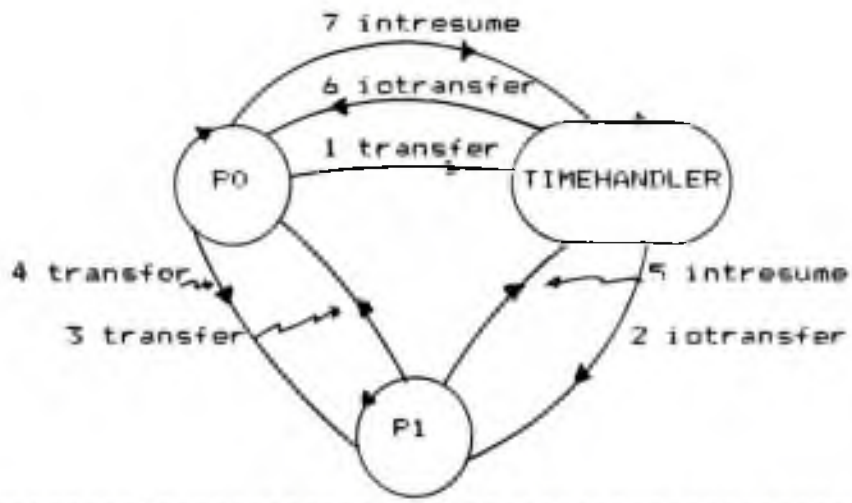
```
{*   sirviendo interrupción.                                          *}
{*****************************************************************}

procedure status8259;
begin
     write(' IMR= ',port[$21]:6);
     port[$20]:=$0a;
     write(' IRR= ',port[$20]:6);
     port[$20]:=$0b;
     write(' ISR= ',port[$20]:6);
end;

{*****************************************************************}

procedure incrementer;
begin
count1 := 0;
while true do
  begin
  count1:=succ(count1);
  if running = p0 then running := p1    {forzando a regresar el pro-}
  else running := p0;                   {cesador a un proceso dife -}
  enableinterrupts;                     {rente al interrumpido      }
  port[$20] := $20;
  iotransfer($08,running .proc);        {esperando interrupción 8}
  end;                                  {generada por el 8253}
end;

{*****************************************************************}

procedure escribe;
begin
count3 := 0;
while  true do
   begin
   count2:=succ(count2);
   gotoxy(10,10);
   writeln('cuenta 2 := ',count2:6);
   status8259;
   disableinterrupts;
   running := p0;
   transfer(running .proc);
   enableinterrupts;
   end;
end;

{*****************************************************************}
```

```
begin  {main}
clrscr;
new(p0);
initnucleus(ofs(nuc),p0 .proc);
new(p1);
p1 .proc:= Newprocess(ofs(escribe),1024);
new(timehandler);
timehandler .proc:=newprocess(Ofs(incrementer),2048);
count4:=0;
while answer <> 'si' do
   begin
   writeln('Empezamos? '); readln(answer);
   end;
clrscr;
running := timehandler;
transfer(running .proc);
  while true do
     begin
     count3:=succ(count3);
     gotoxy(10,14);
     writeln('cuenta 3 := ',count3:6);
     status8259;
     disableinterrupts;
     running := p1;
     transfer(running .proc);
     enableinterrupts;
   end;
end.
```

Ejemplo 4.

Para solucionar el problema de salida a pantalla se construyó un pequeño módulo llamado io.pas (ver módulo io.pas en apéndice B) el cual es probado en el cuarto ejemplo. Como el código generado para io.pas si es reentrante este puede ser llamado desde cualquier proceso.

Esquema de procesos del ejemplo 4:




```
{*************************************************************************}
{*                    PROGRAMA DE DEMOSTRACION NUMERO 4                  *}
{*    En este programa se demuestra que el hecho de usar rutinas         *}
{* de entrada/salida de las bibliotecas de Pascal en procesos aso-       *}
{* ciados a interrupciones no es recomendable ya que estas rutinas       *}
{* no son reentrantes, es decir no se asegura que puedan ser usa -       *}
{* das por dos procesos al mismo tiempo. Para solucionar el proble       *}
{* ma anterior se construyeron rutinas básicas de salida a panta -       *}
{* lla (módulo io.pas) que si son reentrantes.                           *}
{*************************************************************************}

{$K-}   {no cheques desbordamineto de pila}

Program interrupttest;
{$I nucleo.ext}      {rutinas de procesos concurrentes}
{$I IO.pas}
```

```
type
        processref =  processrec;
        processrec = record                    {se crea un identificador}
           suc,pre,remite  : processref;       {de proceso}
           proc      : process;
           end;

var count1,count2,count3,count4: integer;
    timehandler,p0,p1,p2,running:processref;
    answer:string[2];

{*************************************************************************}

procedure incrementer;
begin
count1 := 0;
while true do
  begin
  if running = p0 then running := p1
  else
   if running = p1 then running := p2
   else
      running := p0;
  iotransfer($1c,running .proc); {regresa el control a un proceso}
  count1 := succ(count1);        {diferente al interrumpido}
  gotoxy(1,1);
  putstring('ticks = '); putint(count1,6);
  status8259;
  port[$20] := $20;
  enableinterrupts;
  end;
end;

{*************************************************************************}

procedure escribe;
begin
count3 := 0;
while  true do
   begin
   count3:=succ(count3);
   gotoxy(10,10);
   putstring('cuenta 3 := '); putint(count3,6);
   putstringln(' ');
   running := p0;
   transfer(running .proc);    {transfiere el control a p0}
   end;
end;
```

```
{*************************************************************************}

procedure uno;
begin
count2 := 0;
while  true do
   begin
   count2:=succ(count2);
   gotoxy(10,8);
   putstring('cuenta 2 := '); putint(count2,6);
   putstringln(' ');
   running := p0;
   transfer(running .proc);     {transfiere el control a p0}
   end;
end;

{*************************************************************************}

begin  {main}
clrscr;
new(p0);
initnucleus(ofs(nuc),p0 .proc);
new(p1);
p1 .proc:= Newprocess(ofs(escribe),1024);
new(p2);
p2 .proc:= Newprocess(ofs(uno),1024);
new(timehandler);
timehandler .proc:=newprocess(0fs(incrementer),2048);
count4:=0;
while answer <> 'si' do
   begin
   putstringln('Empezamos? '); readln(answer);
   end;
clrscr;
running := timehandler;
transfer(running .proc);     {transfiere el control a timehandler}
  while true do
     begin
     count4:=succ(count4);
     gotoxy(10,14);
     putstring('cuenta :4 = ');putint(count4,6);
     putstringln(' ');
     running := p1;
     transfer(running .proc);     {transfiere el control a p1}
   end;
end.
```

IMPLEMENTACION DE LAS PRIMITIVAS DEL NUCLEO.-

ORGANIZACION DE LA MEMORIA PARA TURBO PASCAL EN EL MOMENTO DE EJECUCION

Durante la ejecución de un programa en Turbo Pascal se manejan los siguientes segmentos por el programa:

- un segmento de código
- un segmento de datos y
- un segmento de pila.

Se manejan además dos estructuras tipo pila el heap y el stack. El heap se usa para almacenar variables dinámicas y es controlado mediante los procedimientos estandares New, Mark y Release. Al iniciar el programa el apuntador a heap **heapptr** apunta a la parte baja del segmento de pila y crece hacia el área de stack. La variable predefinida **heapptr** contiene el valor del apuntador al heap y permite al programador controlar la posición de éste.

El stack (pila) es usado para almacenar variables locales, resultados intermedios durante la evaluación de expresiones y para transferir parámetros a procedimientos y funciones. Al iniciar el programa, el apuntador de pila es puesto en la localidad más alta del segmento de pila.

En cada llamado al procedimiento New el sistema checa si existe colisión entre el heap y la pila a menos que la directiva del compilador K se encuentre desactivada ({$K-}).

ORGANIZACION DE LA MEMORIA PARA PROCESOS QUE UTILICEN EL NUCLEO

Un requerimento básico para manejar procesos concurrentes es que cada uno de ellos tenga su propia pila para manejo de variables locales y direcciones de regreso. Esto se puede lograr facilmente cambiando el valor del registro SP cuando se transfiera el control del procesador entre dos procesos.

La manera más sencilla de comunicar procesos es mediante el uso de variables globales , esto es variables accesadas por los procedimientos bajos las reglas estandares de Pascal. No existe problema con el uso de variables globales ya que todas ellas son indexadas con respecto al registro base DS.

Las variables manejadas en la pila son accesadas con ayuda del registro BP (registro que apunta a la base de variables locales). El procedimiento **newporcess** deberá colocar un nuevo BP una vez que el proceso ha sido creado. Es importante aclarar que en cada regreso del llamado a un procedimiento implica un cambio en el valor del registro BP. Esta es la razón, por la cual un proceso no deberá alcanzar su fin.

CONMUTACION DE CONTEXTO USANDO EL NUCLEO

Un proceso se suspenderá por un llamado a una primitiva del NUCLEO o por una interrupción. El estado del proceso (contexto) es entonces almacenado en la pila del proceso. El apuntador a esta pila es salvado justamente en la variable tipo process. El área de memoria de un proceso suspendido se muestra en la figura siguiente:

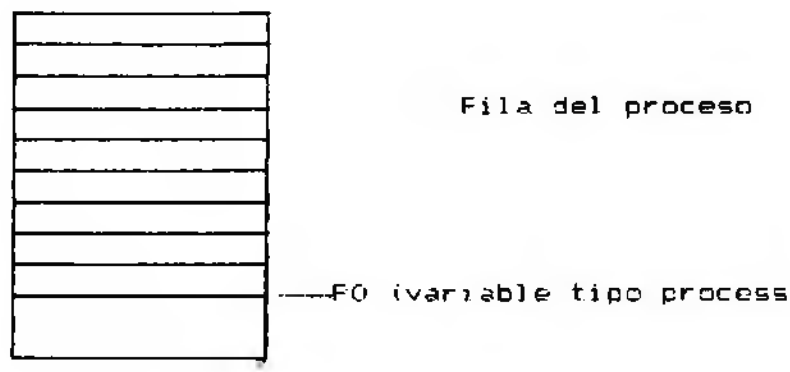


FIG. III.4 AREA DE MEMORIA DE UN PROCESO SUSPENDIDO.

La información guardada en la pila depende de la razón por la cual se suspendio el proceso.

El NUCLEO esta organizado en una forma simétrica. Para cada parte de código que almacena información referente a un proceso, existe una parte correspondiente de código que recupera la información del proceso. El almacenamiento y la recuperación de la información toma lugar en cuatro casos dentro del código; donde el proceso es creado, donde se reactiva un proceso, espera por una interrupción o es interrumpido. La ejecución de un proceso puede así lograse mediante la ejecución del código marcado por la siguientes etiquetas (ver apéndice A) RESCHILD, RES, RESDRIVER y RESINT.

MANEJO DE INTERRUPCIONES.-

Como se mencionó anteriormente la primitiva **iotransfer** suspende el proceso que hace el llamado y reactiva otro proceso. Cuando la interrupción especificada ocurre, el proceso suspendido debera reactivarse. Lo que se desea lograr con esta primitiva es que al llegar la interrupción de alguna manera se haga un llamado al procedimiento.

```
intresume(iosuspended : process);
```

El procedimiento **intresume** es similar a **resume**. La diferencia es que deben salvarse todos los registros y algunas variables de la biblioteca de Pascal para momento de ejecución cuando la interrupción ocurra. El argumento iosuspended es el proceso que hizo el llamado a **iotransfer** con la dirección del vector correspondiente a la interrupción.

Sin embargo, por interrupciones de hardware solo podemos llamar procedimientos sin parámetros. Una solución a este problema es crear dinámicamente un pequeño procedimiento para que cada proceso en espera de una interrupción contenga un llamado a **intresume** con el argumento apropiado. En efecto lo único que hay que hacer es insertar la instrucción:

```
call intresume
```

en el tope de la pila del procedimiento que llame **iotransfer** y la dirección de esta instrucción en la dirección del vector correspondiente.

Cuando la interrupción ocurra y se ejecute la instrucción CALL, el procesador guardará la dirección de regreso en la pila. Sin embargo esta dirección de regreso corresponderá al nuevo valor del apuntador de pila del proceso que esperaba la interrupción. Esta solución permite que el proceso sea reasumido cuando la interrupción ocurre conociendo de una manera eficiente la dirección de la rutina de servicio.

IMPLEMENTACION MANEJADOR DE PROCESOS EN TIEMPO REAL PARA PASCAL.

IMPLEMENTACION MANEJADOR DE PROCESOS EN TIEMPO REAL PARA PASCAL.

En este capitulo se describe un modulo de rutinas el cual hemos llamado KERNEL desarrollado en Turbo Pascal que usa como base las primitivas descritas anteriormente, para el manejo de procesos concurrentes con la flexibilidad del Pascal. Las interrupciones también podrán ser controladas desde Pascal.

El KERNEL soporta programación concurrente en la versión de Turbo Pascal 3.0 desarrollado por Borland. El KERNEL implementa semáforos para exclusión mutua y eventos para otras sincronizaciones.

Un programa que use este KERNEL podrá ser estructurado de una manera similar a programas en Pascal concurrente. Por ejemplo un monitor en Pascal concurrente corresponde a un registro y algunos procedimientos reentrantes asociados. El concepto de cola corresponde a un evento. El KERNEL también ofrece la posibilidad de programar manejadores de entrada-salida.

LAS PRIMITIVAS DEL KERNEL

Declaración, creación, elección y terminación de procesos:

Un proceso se declara como un procedimiento sin parametros, el cual en adelante será referido como process. Si se desean pasar parámetros al proceso, se deberá declarar un procedimiento con parámetros en forma standard y se usará un pequeño process el cual llamará a su vez al procedimiento con los parámetros adecuados.

Un proceso en primera instancia puede ser creado desde cualquier lugar del programa , donde el procedimiento pueda ser llamado en forma ordinaria mediante el llamado del procedimiento **createprocess.**

Function createprocess(proc, memreq:integer):processref;

donde :

proc - es la dirección de procedimiento en Pascal
(ofs(progama))
memereq - es la memoria requerida en bytes para stack.

Un tamaño estimado de pila es requerido para la pila del proceso a crear, los errores de desbordamiento de la pila no son detectados por las rutinas de chequeo a la hora de ejecución. Es responsabilidad del usuario declarar suficiente área de memoria de pila para cada proceso que se cree. Deberá tomarse en cuenta para ello: area

necesaria para guardar registros al momento de conmutación, área usada por llamados a procedimiento, paso de parametros en estas llamadas, área por declaración de variables locales, área para declarción de procesos hijos, etc. Por lo anterior es difícil calcular con exactitud el tamaño de la pila pero un tamaño mínimo razonable son 100 bytes, si no crearemos procesos hijos, ni variables dinámicas.

Tamaño de pila = área para registros + área para paso de parámetros + área de variables locales + área para procesos hijos.

El espacio utilizado para salvar el contexto de un proceso es aproximadamente de 50 bytes. El espacio que ocupa cada variable local o pasada por parametro en el llamado a un procedimiento depende del tipo, por ejemplo una variable tipo entero utiliza una palabra, un caracter un byte, un apuntador dos palabras etc.

Los procesos son lanzados a ejecución de acuerdo a sus prioridades. Un proceso empieza con su prioridad igual a uno (la más alta) y puede ser cambiada dinamicamente por un llamado a **setpriority.**

Procedure **setpriority**(priority : Integer);

donde:

priority – es la nueva prioridad del proceso.

La prioridad puede ser menor que 'maxpriority', la cual es una constante predefinida. Un número de prioridad mayor o igual a maxpriority causa la terminación del proceso y es la forma normal de terminar un proceso, pero la memoria que este proceso ocupa no será liberada.

Dentro del programa principal deberá llamarse a la primitiva **initkernel** antes de que otros procesos sean creados.

Procedure **initkernel;**

El procedure initkernel crea dos procesos de usos especial un manejador del reloj y un proceso ocioso para asegurarse que al menos siempre exista un proceso en ejecución.

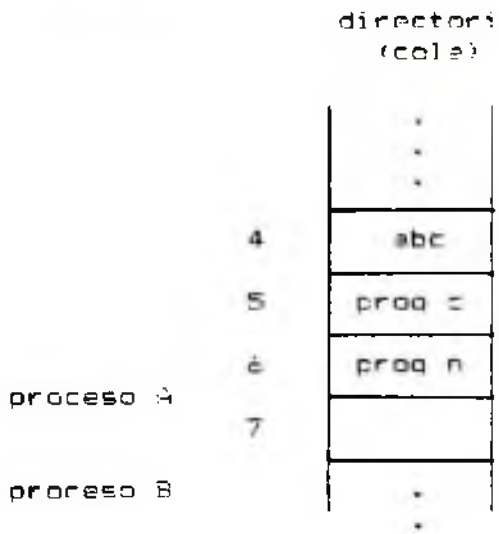
COMUNICACION Y SINCRONIZACION ENTRE PROCESOS

La comunicación entre procesos se logra usando variables, las cuales sean accesibles para los procesos, via parametros de variables o via apuntadores. Las variables globales son frecuentemente empleadas para comunicación.

En algunos sistemas operativos los procesos que estan trabajando a menudo comparten áreas de memoria comunes que cada uno puede leer y escribir. Este recurso compartido puede estar en memoria o puede ser un archivo compartido; la localización de la memoria compartida no cambia la naturaleza de la comunicación o los problemas que involucra. Para ver practicamente como trabaja la comunicación entre

procesos consideremos un ejemplo simple pero comun, una cola de impresión. Cuando un proceso desea imprimir un archivo, guarda el nombre del archivo en una cola. Otro proceso examina la cola periodicamente para ver si hay algún archivo que imprimir y si hay lo imprime removiendo el nombre del archivo de la cola de impresión.

Imagine que nuestro directorio de impresión tiene un gran número de entradas, numeradas 0,1,2,etc capaces de guardar un nombre de archivo. También imagine que existen dos variables comunes, out, que apunta al siguiente archivo a imprimir e in que apunta a la siguiente entrada libre del directorio. En un momento dado las entradas del 0 al 3 se encuentran libres porque ya se imprimieron estos archivos y las entradas de la 4 a la 6 se encuentran llenas. Más o menos en forma simultánea los procesos A y B deciden mandar a imprimir. La situación se muestra en la figura:



En un momento dado se podria presentar la siguiente situación. El proceso A lee la variable in y almacena el valor 7, en una variable local llamada next-free-slot. En ese justo momento llega la interrupción del reloj y el KERNEL decide que el proceso A no siga ejecutandose, así que conmuta al proceso B. El proceso B también lee la variable in y también obtiene un valor de 7, de tal manera que almacena el nombre del archivo a imprimir en la entrada 7 del directorio y actualiza la variable in a 8. Entonces continua haciendo otras cosas.

En algún momento dado el proceso A recupera el control del procesador y continúa en el lugar que se quedo. Para este proceso la siguiente entrada libre del directorio se encuentra en next-free-slot y esta es 7, por lo que guarda en esta entrada el nombre del archivo a imprimir borrando la anterior del proceso B. Las entradas en el directorio son consistentes internamente por lo que el proceso que vacia la cola de impresión no se da cuenta de lo ocurrido y continúa en forma normal su labor, pero el proceso B nunca logró su impresión.

¿Cómo podemos evitar estos problemas?. La clave es encontrar alguna forma de prohibir que mas de un proceso escriba o lea al mismo tiempo, en otras palabras lo que nosotros necesitamos es exclusión mutua, alguna manera de que si un proceso esta usando una variable compartida asegurarse que nadie mas la use al mismo tiempo.

SEMAFORUS.-

Con el propósito de asegurar exclusión mutua se crea un nuevo tipo de variable llamado semáforo. Un semáforo tendrá un valor de cero en el caso de que el recurso compartido no este disponible y tendrá un valor positivo si esta disponible.

Existen dos operaciones sobre un semáforo. La operación wait en un semáforo checa si el valor es mayor que cero . Si es así decrementa el valor del semáforo y continua. Si es cero el proceso se suspende. El checar el valor, actualizarlo y posiblemente suspender el proceso es realizado como una operación indivisible. Con esto garantizamos que una vez que ha empezado la operación sobre un semáforo, ningun otro proceso pueda accesar el semaforo hasta que la operación se complete.

La operación signal incrementa el valor del semáforo, y en caso de haber uno o más procesos suspendidos en ese semáforo, se escogerá uno para que termine su operación wait. Así después de un signal sobre un semaforo con procesos suspendidos pudiera ser todavia cero, pero habrá un proceso menos suspendido en él. Los semáforos que son inicializados con un valor de uno y son usados por dos o mas procesos para asegurar que solo uno de ellos pueda estar dentro de una región crítica, son llamados semáforos binarios. Si cada proceso hace un llamado a wait antes de entrar a la region crítica y un llamado a signal al salir de esta garantizamos la exclusión mutua.

El programador deberá asegurarse exclusión mutua mediante el uso de semáforos o deshabilitando interrupciones temporalmente. Estos semáforos pueden ser operados mediante los siguientes tres procedimientos.

```
Procedure initsem(var sem : semaphore; initval : integer);

Procedure wait(sem : semaphore);

Procedure signal(sem : semaphore);
```

```
sem - variable semaforo
initval - valor inicial del semaforo
```

Un semáforo deberá ser inicializado mediante un llamado al procedimiento initsemaphore . El efecto de '**signal**' es incrementar el valor del semáforo en uno, este incremento es realizado como una operacion indivisible. El efecto de '**wait**' es decrementar el valor del semaforo en uno en el caso de que el valor del semáforo fuera no negativo.

Un llamado de wait implica un retardo potencial. La cola de procesos en espera es ordenada de acuerdo a las prioridades de los procesos.

EVENTOS.-

La sincronización entre procesos tal como la espera de una condición en una variable compartida, se logra mediante el uso del concepto evento. Existen tres operaciones con eventos.

Procedure **initevent**(var e:event; sem : semaphore);

Procedure **await**(e : event);

Procedure **cause**(e : event);

donde:

e - es la variable tipo evento
sem - es el semáforo asociado para la exclusion mutua.

Un evento deberá ser inicializado mediante un llamado a '**initevent**', el cual esta asociado con un semáforo para la exclusión mutua. Un llamado al procedimiento '**await**' retarda o suspende el proceso en espera de algún "evento" y realiza un implícito '**signal**' al semáforo asociado, esto último con la idea de desbloquear aquellos procesos que estuvieran en la cola del semáforo asociado con el evento. Un llamado al procedimiento '**cause**' por otro proceso, mueve todos los procesos suspendidos (en espera de que sucediera el evento) a la cola asociada con el semáforo.

MENSAJES.-

Una forma de comunicar procesos es a través de mensajes, ya sea que se envíe un mensaje de proceso a proceso, o se envíe un mensaje a algun manejador para realizar una tarea (por ejemplo al controlador de disco).

procedure **send**(dest:processref;var mes:message);

donde:

dest - es el proceso al que mandamos el mensaje.
mes - es el mensaje enviado.

Cuando un proceso realiza un **send**, el KERNEL checa para ver si el destinatario esta esperando el mensaje del que envía (o de cualquiera (ANY)). Si es así se copia el mensaje del buzon del que envía al buzon del que recibe y los dos procesos son formados en la cola de listos para ejecución. Si no existe destinatario esperando el

mensaje del que lo envia, el proceso que envia el mensaje es marcado como suspendido y puesto en una cola de procesos que esperan enviar mensaje.

procedure **receive**(sender:processref; VAR mes:message);

donde:

sender - es el proceso del cual esperamos mensaje.
mes - es la variable que guardará el mensaje a recibir.

Cuando un proceso realiza un **receive**, el KERNEL checa si algún proceso se encuentra encolado tratando de enviarle. Si es asi se copia el mensaje del buzon del proceso suspendido en la cola de envio al buzon del destinatario y los dos procesos son marcados como listos para ejecución en la cola de listos. Si no existe ningún proceso tratando de enviarle, el remitente se bloquea hasta que el mensaje arribe.

MANEJO DEL RELOJ.-

El procedimiento **waittime** provoca que el proceso que lo llame espere un intervalo de tiempo especificado.

Procedure **waittime**(time : integer);

donde:

time - es el tiempo que se bloqueará el proceso.

La unidad de tiempo es el tick (1/18 de seg). Existen predefinidos los siguientes tipos de unidades de tiempo:

```
tick = 1;
sec = 18;
min = 1092;
```

Esto significa que es posible especificar un intervalo de tiempo de la siguiente manera: waittime(2*min+10*sec);. Note que el intervalo de tiempo maximo esta dado por el maximo entero representable (ticks).

MANEJO DE INTERRUPCIONES

El procedimiento **waitio** hace que el proceso que realiza el llamado espere por una interrupción especifica. Unicamente un proceso a la vez puede esperar por una interrupción y esto deberá ser garantizado por el usuario.

Procedure **waitio**(typeint : int ; int8259 : byte);

donde:

typeint - es el tipo de interrupcion que se espera.
int8259 - es la linea correspondiente usada por el controlador de interrupciones 8259.

El procedimiento habilita la linea correspondiente del registro de mascara del 8259, entonces algun otro proceso es seleccionado para su ejecucion.

Cuando la interrupción ocurre se restaura el valor original de la linea correspondiente del registro de mascara del 8259. Si es necesario se manda el comando de fin de interrupción al 8259. El proceso que esperaba la interrupcion es puesto en la cola de procesos listos para ejecucion y el proceso de mayor prioridad es seleccionado para su ejecucion.

TEMPORIZADORES

En algunas aplicaciones es necesario contar con procedimientos que se ejecuten despues de un tiempo determinado, podria pensarse que con la primitiva waittime tenemos solucionado el problema, sin embargo no siempre se desea la conmutación de contexto por lo que esto implica: que el proceso actual pierda el procesador, el tiempo perdido en la conmutacion etc. Un ejemplo de esto podria ser el caso del controlador del disco. Despues de un acceso a disco hay que esperar un tiempo razonable con el motor prendido, previendo que no tengamos que prender y apagar este motor en cada acceso, una vez que expire este tiempo se deberá apagar.

function **createproced**(proced : integer):processref;

donde:

proced - es la dirección de procedimientoa ejecutar.

Createproced crea un identificador para el procedimiento que se mandara ejecutar despues de que transcurra un cierto periodo de tiempo especificado en waitproced.

procedure **waitproced** (p2:processref ;t : integer) ;

donde:

p2 es el identificador de procedimiento a ejecutar.
t es el tiempo a esperar antes de ejecutarlo.

Waitproced forma el identificador de procedimiento creado por createproced en la cola de procedimientos a ser ejecutados despues de agotar su tiempo de espera.

procedure **removeproced**(p2:processref);

donde:

p2 es el identificador del procedimiento a remover.

Removeproced permite remover un temporizador antes de que se agote su tiempo de espera, de esta manera ya no se ejecuta el procedimiento.

ORGANIZACION DE LOS PROGRAMAS

El usuario del KERNEL debera de compilar su programa junto con la interface del KERNEL.

Ya que el programador debe asegurar por el mismo la exclusion mutua, es importante organizar el programa de tal manera que ayude a usar semaforos de una manera adecuada. Una solucion natural es colectar todos los datos, y los semaforos para exclusion mutua y las variables y eventos en un registro.

```
type data = record
     mutex : semaphore;
     cond : event;
     ...
     end;
var data1 : data;
```

Las operaciones en los datos son entonces realizadas mediante el uso de la instruccion with.

```
with data1 do
     begin
     wait(mutex);
     while .... do await(cond);
     ....
     signal(mutex);
     end;
```

Los procedimientos ordinarios en Pascal son reentrantes. Esto significa que es posible construir un conjunto de procedimientos que operen en los datos compartidos, y siendo esta la unica forma. Sobre esta idea trabaja el concepto de monitor.

RELACION ENTRE EL KERNEL Y PASCAL CONCURRENTE

Cuando usamos el KERNEL, es necesario que se declare explicitamente cualquier semáforo y hacer los llamados de wait y signal. Una diferencia primordial es que el compilador de Pascal Concurrente asegura la exclusión mutua.

Una variable tipo evento corresponde a una variable de tipo standard queue en Pascal Concurrente, con las siguientes grandes diferencias: unicamente un proceso a la vez puede ser retardado en una variable queue y un llamado a continue implica un regreso implicito desde el procedimiento de entrada. Para el caso del KERNEL se utiliza un with para accesar los campos de la variable, mientras

que en el Pascal Concurrente es hecho implicitamente. Por ejemplo un procedimiento de cierto monitor es llamado con notacion punto.

```
outbuffer.send(ch);
```

Mientras que usando Turbo Pascal el monitor se da como un argumento ordinario. (analizar ejemplo numero dos)

```
send(outbuffer,ch);
```

PROCESOS

Para los procesos en Pascal Concurrente esta permitido tener parametros formales de tipo monitor para lograr "accesos correctos" a los procedimientos. En el caso de Turbo Pascal con el KERNEL esto corresponde al uso de variables formales del tipo registro correspondiente a los monitores. Sin embargo recordemos que a los procesos no se les permite el paso de parametros, lo cual significa que un procedimiento interface tiene que ser declarado para cada proceso con diferentes argumentos para cada proceso.

El uso de procesos es demostrado por un ejemplo. Un proceso se encuentra recibiendo caracteres de un monitor tipo buffer. La descripción de esta situacion se muestra primero para el Pascal Concurrente.

```
type consumer = process(buff:buffer);
     var ch:char;
     begin
        cycle
        buff.receive(ch);
        ...
        end
     end;
var cons:consumer;
...
     init outbuffer,
     cons(outbuffer);
```

La correspondiente descripcion cuando se usa Turbo Pascal con el KERNEL se muestra a continuacion.

```
var outbuffer:buffer;

procedure consumer(var buff : buffer);
var ch:char;
begin
while true do
     begin
     receive(buff,ch);
     ...
     end;
end;
```

```
procedure cons;
begin
consumer(outbuffer);
end;
...
initbuffer(outbuffer);
createprocess(cons,...);
```

IMPLEMENTACION

La introduccion de procesos concurrentes implica que tanto el código, como procesador y memoria son recursos compartidos. Ahora se considera el problema de manejar y proteger estos recursos.

RUTINAS COMPARTIDAS

Una rutina (procedimiento o funcion) compilado por Turbo Pascal es reentrante, debido a que Pascal permite rutinas recursivas. Esto significa que dicha rutina puede ser usada por varios procesos a la vez. Sin embargo, no se asegura que rutinas de la biblioteca de Pascal (por ejemplo sin, cos, write, writeln, etc) sean reentrantes, ademas Turbo Pascal permite ensamblar codigo en linea o en forma separada y esto no significa que una rutina, no pueda ser usada por varios procesos a la vez, asi que como un recurso comun tendra que ser protegido por un semáforo.

MANEJO DEL PROCESADOR

El KERNEL debera decidir cual de los procesos sera ejecutado. Con respecto a la administracion del procesador, los procesos existentes pueden ser divididos dentro de tres grupos, ejecutándose, listos para ejecutar y suspendidos. En el estado de listos para ejecucion los procesos compiten por el procesador de acuerdo a su prioridad.

Un proceso puede estar en espera de una señal de sincronización. (semáforo, evento, tiempo determinado, o una interrupción). Si la señal de sincronización no ha arribado, el proceso es transferido al estado de suspendido y el proceso pasa al estado de listo cuando la señal arriba. Una señal de sincronización es enviada cuando un proceso llama a signal o await, llega un tick del reloj o sucede una interrupcion. Cuando un proceso llama a cause, los procesos que esperaban tal evento, son todos transferidos para esperar la señal de sincronización del semáforo asociado. Para que el KERNEL sea capaz de realizar la transición , el KERNEL debera conocer la prioridad y las variables del proceso (apuntador de pila , apuntador de programa, etc.) de cada proceso.

Es conveniente crear un registro para cada proceso.

Cuando se produce una señal de sincronizacion , el KERNEL debera encontrar al proceso en espera. Todos los procesos en espera de una señal son por ello organizados en listas doblemente encadenadas.

Existe asi una lista de registros de procesos asociado con cada semaforo ('en espera') y con cada evento ('suspendido '). Todos los procesos en espera de un tiempo especifico son conservados en una lista sencilla ('timequeue'). Estos son ordenados en forma ascendente de acuerdo al tiempo de espera. Los records de proceso contienen un campo ('time') el cual contiene el tiempo de espera relativo al proceso precedente en la lista timequeue'. El tiempo de espera relativo del primer proceso en la lista es relativo al tiempo corriente. Los tiempos de espera son calculados por el procedimiento waittime.

Solamente se permite que un proceso espere por una determinada interrupcion, asi que no es necesaria una lista.

Los procesos en estado de listos para ejecucion son también guardados en una lista ('readyqueue') y una variable ('running') de tipo processref conserva el proceso ejecutandose.

Para el manejo de los temporizadores se ha creado una cola de procedimientos en espera de ejecutarse de acuerdo al tiempo que esperan y es manejada dentro del proceso clock del KERNEL. Una vez que el tiempo de espera de un procedimiento ha terminado, este se ejecuta y regresa al proceso clock a continuar con su trabajo normal (no existe conmutación de contexto en un temporizador). No es valido que dentro de un temporizador se llamen primitivas que provoquen cambio de contexto en ese momento.

LA ESTRUCTURA DEL KERNEL

La estructura de los datos del KERNEL es un recurso compartido y la exclusión mutua es garantizada deshabilitando interrupciones. El código del modulo KERNEL se encuentra disponible en el apendice B.

EJEMPLO 1.-

El siguiente ejemplo clasico nos muestra como funcionan las primitivas del KERNEL para garantizar la sincronizacion entre procesos.

Cinco filósofos pasan la vida pensando y comiendo. los filosofos comparten una mesa circular con cinco sillas, cada una perteneciendo a un filosofo. En el centro de la mesa hay un tazon de arroz y en la mesa hay cinco palillos. Cuando un filosofo piensa, no interactua con sus colegas. De vez en cuando a un filósofo le da hambre y trata de tomar los dos palillos más cercanos a el (los palillos que estan entre su vecino de la izquierda y él y entre sus vecinos de la derecha y él). Un filosofo puede tomar un solo palillo a la vez. Obviamente el no puede tomar un palillo que ya esta en la mano de su vecino. cuando un filosofo hambriento tiene ambos palillos en su poder, come sin soltarlos. Cuando ha terminado suelta ambos palillos y empieza a pensar de nuevo.

Una simple solución se logra representando cada palillo como un semaforo. Un filosofo trata de tomar un palillo mediante una operacion wait en ese semaforo y lo libera ejecutando un signal en el semáforo.

```
{*************************************************************************}
{*              PROGRAMA DE DEMOSTRACION FILOSOFOS COMELONES              *}
{*En este programa se demuestra el uso de semáforos para exclusión mutua*}
{*************************************************************************}

{$R-}
program comelon(input,output);
{$I nucleo.ext}
{$I kernel.pas}
var
    answer : string[2];
    chopstick : array [0..4] of semaphore; {Un semáforo por filosofo}
    mutex : semaphore;    {semáforo para exclusión mutua en regiones
                           criticas}
    status : array [0..4] of string[12];
    m,k:integer;

{*************************************************************************}
```

```
procedure escesta;
  var  nf:integer;              {Escribe el estado en que se encuentran los}
  begin                         {filosofos, los estados posible son: }
  gotoxy(1,8);                  {hambriento, comiendo o pensando}
  for nf:= 0 to 4 do
     begin
       write(status[nf]);
     end;
  writeln;
  end;

{*************************************************************************}

function left(i:integer):integer;
begin
left:=(i+4) mod 5;
end;

{*************************************************************************}

function right(i:integer):integer;
begin
right:=(i+1) mod 5;
end;

{*************************************************************************}

procedure test(i:integer);
begin
if(status[i]='hambriento  ') and (status[left(i)]<>'comiendo    ')
  and (status[right(i)]  <>  'comiendo    ') then
  begin
  status[i] := 'comiendo    ';
  escesta;
  signal(chopstick[i]);
  end;
end;

{*************************************************************************}

procedure take_chopstick(i:integer);
begin
wait(mutex);                            {entrando a region critica}
status[i]:='hambriento  ';              {filosofo i esta hambriento}
escesta;
test(i);                                {trata de tomar 2 palillos  }
signal(mutex);                          {saliendo de region critica}
wait(chopstick[i]);
end;
```

```
{*************************************************************************}

procedure put_chopstick(i:integer);
begin
wait(mutex);
status[i]:='pensando    ';
escesta;
test(left(i));
test(right(i));
signal(mutex);
end;

{*************************************************************************}

procedure philosopher(i:integer);
begin
while true do
  begin
  waittime((random(5)+3)*sec);              {tiempo para pensar}
  take_chopstick(i);
  waittime((random(5)+3)*sec);              {tiempo para comer}
  put_chopstick(i);
  end;
end;

{*************************************************************************}

{procesos} procedure phil0 ;
     begin
     philosopher(0);
     end;
procedure phil1 ;
     begin
     philosopher(1);
     end;
procedure phil2 ;
     begin
     philosopher(2);
     end;
procedure phil3 ;
     begin
     philosopher(3);
     end;
procedure phil4 ;
     begin
     philosopher(4);
     end;
```

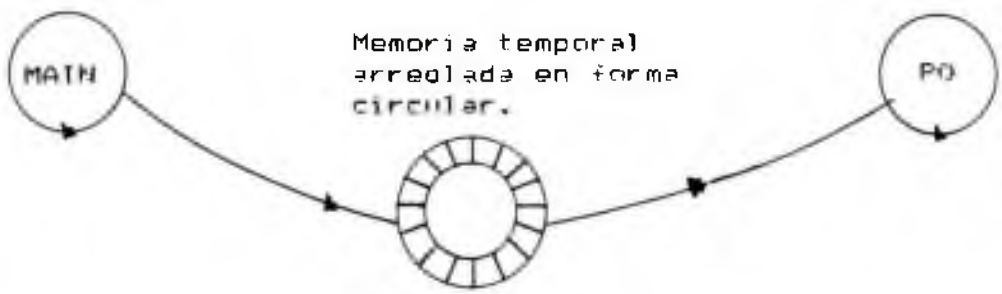
```
var p0,p1,p2,p3,p4:processref;

{*****************************************************************************}

begin {main}
while answer <> 'si' do
begin
writeln('  empezamos (si no)?');
readln(answer);
end;
initkernel;
p0:=createprocess(ofs(phil0),2048);
p1:=createprocess(ofs(phil1),2048);
p2:=createprocess(ofs(phil2),2048);
p3:=createprocess(ofs(phil3),2048);
p4:=createprocess(ofs(phil4),2048);
k:=0;
initsem(mutex,1);
for k:= 0 to 4 do
  begin
  initsem (chopstick[k],0);
  status[k]:='pensando    ';
  end;
clrscr;
while true do
   begin
   gotoxy(1,5);
   writeln('filósofo1   filosofo2   filosofo3   filosofo4   filósofo5');
   escesta;
   waittime(4*sec);
   end;
end.
```

Un monitor es una colección de procedimientos, variables y estructuras de datos los cuales se encuentran dentro de un mismo módulo o paquete. Los procesos pueden llamar los procesos dentro del monitor cuando lo deseen, pero no pueden accesar directamente las estructuras de datos internas del monitor desde procedimientos fuera de él. Los monitores tienen una propiedad importante que los hace usuales para lograr la exclusión mutua: solamente un proceso puede estar activo en el monitor en un momento dado.

EJEMPLO 2

Un ejemplo del uso de monitores con Pascal y KERNEL es el siguiente: Concideremos el problema del productor - consumidor. Dos procesos comparten un buffer circular de tamaño fijo. Uno de ellos el productor pone información dentro del buffer y el otro los consume de ahí. Los problemas surgen cuando el productor desea poner un nuevo elemento dentro del buffer, pero lo encuentra lleno. La solución para el proceso es bloquearse, y que sea liberado cuando se retire un elemento o más del buffer. De manera similar si el consumidor quiere remover un elemento del buffer y este se encuentra vacío, deberá bloquearse y esperar a que el productor ponga uno o más elementos.

Gráficamente el problema se vería así:




```
{$k-}
program fox(input,output);
type
    process =  integer;
    ioprocess =  integer;
var
    letrero: string[30];
    i : integer;
    answer : string[2];
{$I nucleo.ext}
{$I kernel.pas}
/* ---------------------------------------------------------------*/
```

```
(monitor tipo buffer)
const buffersize = 100;
type buffer =
     record
     guard:semaphore;
     change : event;
     charbuff : array [1..buffersize] of char;
     count:0..buffersize;
     inp,outp:1..buffersize;
     end;
procedure putbuffer(var buff:buffer; ch:char);
     begin
     with buff do
        begin
        wait(guard);          {entrando a region critica}
        while count = buffersize do  await(change);
        charbuff[inp] := ch;
        inp := (inp mod buffersize) + 1;
        count := count+1;
        cause (change);
        signal(guard);        {saliendo de region critica}
        end;
     end;
procedure getbuffer(var buff:buffer;var ch:char);
     begin
     with buff do
        begin
        wait(guard);          {entrando a region critica}
        while count = 0 do await(change);
        ch := charbuff[outp];
        outp := (outp mod buffersize) + 1;
        count := count-1;
        cause (change);
        signal(guard);        {saliendo de region critica}
        end;
     end;
procedure initbuffer(var buff:buffer);
     begin
     with buff do
        begin
        initsem(guard,1);
        initevent(change, guard);
        count := 0;
        inp :=1;
        outp := 1;
        end;
     end;
/*-------------------------------------------------------*/
```

```
{procesos}        {proceso consumidor}
procedure driver(var outbuffer : buffer);
var ch:char;
   begin
   while true do
      begin
      getbuffer(outbuffer,ch);
      write(ch);
      end;
   end;
{monitores}
var printer1, printer2: buffer;
{procesos}
procedure driver1 ;                    {proceso consumidor 1}
   begin
   driver(printer1);
   end;
procedure driver2;                     {proceso consumidor 2}
   begin
   driver(printer2);
   end;
var p0,p1,p2,p3,p4:processref;
{*************************************************************************}
   begin {main}                    {proceso productor}
   while answer <> 'si' do
      begin
      writeln('  empezamos? (si/no)'); readln(answer);
      end;
   initkernel;
   initbuffer(printer1);
   initbuffer(printer2);
   p0:=createprocess(ofs(driver1), 1024);
   p1:=createprocess(ofs(driver2), 1024);
   while true do
      begin
      clrscr;
      gotoxy(5,5);
      writeln('PROBANDO MONITORES TIPO BUFFER');
      letrero:= 'THE QUICK BROWN FOX JUMPED    ';
      for i :=  1 to ord(letrero[0]) do
          putbuffer(printer1,letrero[i]);
      letrero:= 'OVER THE LAZY DOG"S BACK      ';
      for i :=  1 to ord(letrero[0]) do
          putbuffer(printer2,letrero[i]);
      end;
   end.
```

EJEMPLO 3

La comunicación de procesos puede lograrse mediante el intercambio de mensajes de tamaño fijo. En el KERNEL disponemos de dos primitivas para enviar y recibir mensajes; send y receive. Cuando un proceso envía un mensaje a otro que no lo espera, el que lo envía se suspende hasta que el destinatario lo recibe. En otras palabras el KERNEL se evita problemas de almacenar mensajes que no han sido recibidos. A continuación se muestra un ejemplo muy sencillo de comunicación de procesos a traves de mensajes.

```
{*************************************************************************}
{*                              DEMOPC3.PAS                              *}
{*    Comunicación de procesos a traves de mensajes.                     *}
{*************************************************************************}

{$k-}
program mensaje(input,output);

var
    answer : string[2];

{$I nucleo.ext}
{$I kernel.pas}

var p0,p1,p2 : processref;
    mes0, mes1, mes2 : message;

procedure recep0;
begin
while true do
   begin
   receive(main,mes0);                    {Espero mensaje de proceso main}
   GOTOXY(15,2);
   writeln('recibi este mensaje');
   writeln(mes0.m_m2.m2ca1);
   end;
end;
procedure recep1;
begin
while true do
   begin
   receive(any,mes1);                    {Espero mensaje de cualquier proceso}
   GOTOXY(15,6);
   writeln('recibi este mensaje');
   writeln(mes1.m_m2.m2ca1);
   end;
end;
```

```
{*************************************************************************

  begin {main}
  while answer <> 'si' do
     begin
     writeln('  empezamos? (si/no)'); readln(answer);
     end;
  initkernel;
  p0:=createprocess(ofs(recep0),1024);
  p1:=createprocess(ofs(recep1),1024);
  clrscr;
     while true do
     begin
     mes2.m_m2.m2ca1 := 'mensaje para p0';
     send(p0,mes2);                    {Envia mensaje a proceso p0}
     mes2.m_m2.m2ca1 := 'mensaje para p1';
     send(p1,mes2);                    {Envia mensaje a proceso p1}
     end;
  end.
```

EJEMPLO 4

Una herramienta que puede ser muy util es el temporizador, el cual ejecuta un procedimiento que debera ejecutarse al expirar un tiempo programado. A diferencia de la primitiva waittime que "despierta" un proceso en este caso no existe una conmutacion de contexto. En el ejemplo siguiente se demuestra el uso de estos temporizadores .

Se crean dos temporizadores, los cuales se programan para ejecutarse el primero después de un segundo y el otro después de un minuto. Como un temporizador es removido después de que ejecuto su tarea, en el ejemplo se programan por si solos para que esperen en forma periodica.

```
{*************************************************************************}
{*                            DEMOPC4.PAS                                *}
{*    Programa que demuestra el uso de temporizadores en Pascal          *}
{*    concurrente                                                        *}
{*************************************************************************}

{$k-}
program procedimientos(input,output);

var
    answer : string[2];
    i,j : integer;

{$I nucleo.ext}
{$I kernel.pas}

var p3,p4 : processref;

{*************************************************************************}

procedure minutos;

begin
i := (i+1) mod 60;
gotoxy(10,5);
writeln('minutos  =',i:3);
waitproced(p3,1*min);
end;

{*************************************************************************}
```

```
procedure segundos;

begin
j := (j+1) mod 60;
gotoxy(10,10);
writeln('segundos  = ',j:3);
waitproced(p4,1*sec);
end;

{*****************************************************************************}

begin {main}
while answer <> 'si' do
begin
writeln('  empezamos (si/no)?');
readln(answer);
end;
i := 0;
j := 0;
clrscr;
initkernel;
p3 := createproced(ofs(minutos));
p4 := createproced(ofs(segundos));
gotoxy(10,5);
writeln('minutos  =',i:3);
waitproced(p4,1*sec);
waitproced(p3,1*min);
while true do
  begin
  gotoxy(10,1);
  writeln('PROBANDO TEMPORIZADORES');
  waittime(4*sec);
  end;
end.
```

Debido a que el KERNEL modifica el vector de interrupcion 20 para llevar un reloj y que los procesos se asume que nunca alcanzan su fin. Los programas se deberan terminar con un reinicio de la máquina para restablecer este vector de interrupcion.

MANIPULACION DEL HARDWARE DESDE UN LENGUAJE DE ALTO NIVEL.

MANIPULACION DEL HARDWARE DESDE UN LENGUAJE DE ALTO NIVEL.

Se pensó necesario incluir este capítulo antes de pasar a la implementación de un manejador de disco y dejar las bases necesarias de "hardware" en el desarrollo de interfaces de entrada/salida.

Muchos de los circuitos integrados de las microcomputadoras PC son programables, generalmente los programas son elaborados en ensamblador, sin embargo se ha querido hacerlo desde un lenguaje de alto nivel a fin de ganar claridad y modularidad. A continuación se explican brevemente las funciones de algunos de los controladores que estan incluidos en el hardware de la PC y que tiene que ver con la transferencia de información a disco flexible. Se incluye un programa en Pascal para realizar transferencia a disco.

CONTROLADOR DE INTERRUPCIONES 8259A.

En la PC el controlador de interrupciones 8259A provee ocho líneas de interrupción con prioridades, conectado como se muestra en la siguiente figura:

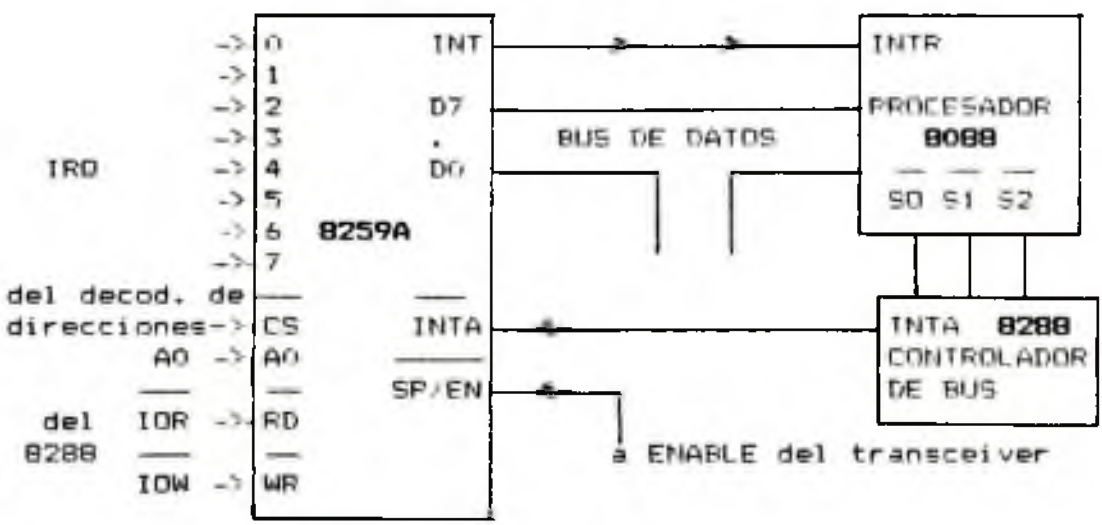


FIG. V.1 CONTROLADOR 8259A Y SUS CONEXIONES.

NIVELES DE INTERRUPCION.

Existen ocho lineas de peticion de interrupcion. Se identifican por IR0 y tienen numeros del 0 al 7. La linea 0 tiene la prioridad mayor y provee una interrupcion periodica para un reloj de propósito general. Esta activa la interrupcion 8 en un periodo de 18.2 veces por segundo. La siguiente en prioridad es la de nivel 1, la cual recibe las peticiones de interrupcion desde teclado cuando un codigo es enviado y activa la interrupcion tipo 9.

Las otras 6 lineas de petición provienen del canal de entrada/salida y son provistas para interrupciones de dispositivos conectados a través de ranuras de expansion. El nivel 6 corresponde a la interrupción tipo 0E y es usada por el sistema de disco flexible, el nivel 7 corresponde a linea de petición de la impresora. El controlador de comunicacion asincrona envia una peticion de interrupción a traves de la linea de nivel 4. En la figura IV.2 se presenta una tabla con las 8 lineas de interupción y sus funciones.

Nivel de interrupcion	Direccion del vector	Tipo de interrupcion	Funcion
IR00	20-23H	INT 08H	Temporizador
IR01	24-27H	INT 09H	Teclado
IR02	28-2BH	INT 0AH	
IR03	2C-2FH	INT 0BH	Comunicacion
IR04	30-33H	INT 0CH	Comunicacion
IR05	34-37H	INT 0DH	Disco duro
IR06	38-3BH	INT 0EH	Disco flexible
IR07	3C-3FH	INT 0FH	Impresora

FIG. V.2 NIVELES DE INTERRUPCION DEL 8259A.

INICIALIZACION.

Antes de empezar a funcionar normalmente, se deben escribir tres bytes al controlador. Cada uno es un comando de inicializacion ICW. En orden seran ICW1, ICW2, e ICW4. El byte ICW3 se aplica unicamente cuando existe más de un 8259 conectado, de esta manera en la PC se ignora.

ICW1. Con una instruccion OUT con A0 en cero y D4 en 1 el controlador lo interpreta como ICW1. Para el 8088 el byte es el siguiente:

D3 = 0 para entradas disparadas por flanco
D3 = 1 para entradas disparadas por nivel

FIG. V.3 ICW1 PARA UN SOLO 8259A.

Seleccionando ceros para valores sin importancia (X) se pueden seleccionar entradas disparadas por flanco escribiendo un 13H en el puerto 20H. Una ventaja de las entradas disparadas por flanco es que una peticion de interrupcion que permanezca en alto no activa una segunda interrupcion después de que esta ha sido procesada. El valor de D4 identifica ICW1, el valor de D1 indica un solo controlador y D0 siempre es 1 para el procesador 8088/8086.

ICW2. Cada nivel de interrupción provee un tipo de interrupcion n. Sin embargo se requieren valores consecutivos de n para los niveles de 0 a 7, estos valores pueden ser seleccionados en el rango de 0 a FFH. La seleccion se completa con un comando ICW2 como se muestra en la figura:

Los tipos de interrupcion son determinados por D7-D3
D3 = 1 para entradas disparadas por nivel

FIG. V.4 ICW2 PARA 8259A.

La direccion 21H y los bits del bus de datos no estan involucrados en el direccionamiento. Para la PC, el nivel 0 corresponde a INT 08. De acuerdo a esto ICW2 es 08. Con esta selección el controlador automaticamente genera los numeros de los tipos de interrupcion de la 08H a la 0FH correspondientes a los niveles de interrupcion IR00 a IR07.

ICW4.- Para la PC el byte de ICW4 en binario es 0000 1001, y la direccion del puerto donde se escribe es 21H. Los tres bit mas significativos siempre son 0. Los valores seleccionados para los bits D3 y D2 causan que una señal de salida sea generada en la pata SP/EN. Esta señal se pone en bajo cuando el tipo de interrupción es colocado en el bus de datos. Es usado en la PC para deshabilitar el "transceiver" de datos, de esta manera se aisla a la CPU de otros periféricos que no sean el controlador de interrupciones. Un 0 lógico para D1 especifica el modo normal del fin de interrupción (EOI), y el bit D0 en 1 indica que se trata de un procesador 8088.

El controlador es inicializado durante los procedimientos que siguen a un encendido del sistema. Las instrucciones de inicializacion son expresadas en Pascal en la siguiente figura:

```
port[$20] = $13  {ICW1}
port[$21] = $08  {ICW2}
port[$21] = $09  {ICW4}
```

Despues de ejecutar esta secuencia, el controlador esta listo para aceptar interrupciones externas.

FIN DE INTERRUPCION (EOI).

Cuando se ha aceptado una interrupcion externa para procesarla, el bit correspondiente del registro ISR (interrupciones siendo servidas) es prendido deshabilitando todas las interrupciones de menor prioridad. Las interrupciones de prioridad mayor son permitidas. Al finalizar una rutina de interrupcion deberá incluirse una operacion que apague el bit del registro ISR, esto se logra escribiendo una palabra de comando OCW2. La direccion es 20H con los bits D4 y D3 en 0. Si no se hace esto, no se podran recibir nuevas interrupciones con prioridades igual o menor a la linea indicada por el bit del ISR. La intruccion a insertar seria:

```
port[$20] = $20;
```

Escribiendo una palabra de control OCW1 en el registro de máscara del 8259 (IMR) es posible enmascarar cualquiera de las 8 lineas de interrupcion. Un 1 en el bit correspondiente deshabilita la interrupción y un 0 la habilita.

ACCESO DIRECTO A MEMORIA Y SISTEMA DE DISCOS FLEXIBLES

En la PC la transferencia de informacion entre un disco flexible y memoria se realiza por un método conocido como acceso directo a memoria (**DMA**). Este metodo es ampliamente usado en la mayoria de los sistemas de computo.

ACCESO DIRECTO A MEMORIA.

Una forma de mover informacion a memoria consiste en mover primero una palabra dentro del acumulador del procesador y luego transferirla a la localidad de memoria o a al dispositivo de E/S. Un procedimiento mas rapido es a través de DMA, mediante el cual se "flotan" las lineas de datos, direcciones y de control apropiadas del CPU para permitir que un controlador externo transfiera los datos directamente entre el periférico y la memoria via el bus de datos. Se utiliza DMA en algunas ocasiones para transferencias a gran velocidad de un bloque de memoria a otro. la cual es una operacion normal en grandes sistemas con muchos usuarios concurrentes. Las transferencias se pueden realizar de periféricos a memoria o de memoria a periféricos y no se involucran los registros del procesador, pero el CPU debe programar la inicialización del controlador de DMA con información apropiada. Aun asi se incrementa la velocidad de operacion y se reduce el "software".

CONTROLADOR DE DMA 8237A

El acceso directo a memoria es controlado por un integrado, el cual tiene un buffer para datos y una unidad logica de control más un canal de transferencia para cada periférico que se atienda. Cada canal tiene su propio registro de dirección de memoria y un registro contador de bytes que almacena el numero de bytes a transferir. El controlador es inicializado por el procesador. El controlador empleado por la PC es el **8237A**, es programable y tiene cuatro canales independientes de DMA.

CANALES.

El canal 0, con la mayor prioridad de los cuatro, es programado para refrescar la memoria dinámica de la tarjeta madre y de las tarjetas de expansión. Los manejadores de disco usan el canal 2. Los otros dos canales estan disponibles para futuros periféricos que se conecten.

OPERACION BASICA.

Cuando un periférico requiere servicio de DMA una de las cuatro lineas (DREQ) al controlador de DMA es activada, el

controlador manda una peticion (HRQ) al controlador de los buses del sistema, asumiendo que el bit que enmascara el canal no este puesto. La señal de "handshake" es enviada al generador de estados de espera de tal manera que se complete el ciclo de bus actual. Después de recibir el regreso de la señal de "handshake" HLDA, el circuito notifica al periférico en particular por medio de la linea DACK que su peticion de DMA ha sido reconocida y que puede empezar a tranferir. El ciclo cambia de ocioso a activo. La transferencia de un byte requiere de 1.05 microsegundos o cinco periodos de reloj. Cuando la transferencia termine el controlador de DMA inactiva la linea HRQ y el procesador toma nuevamente el control de los buses.

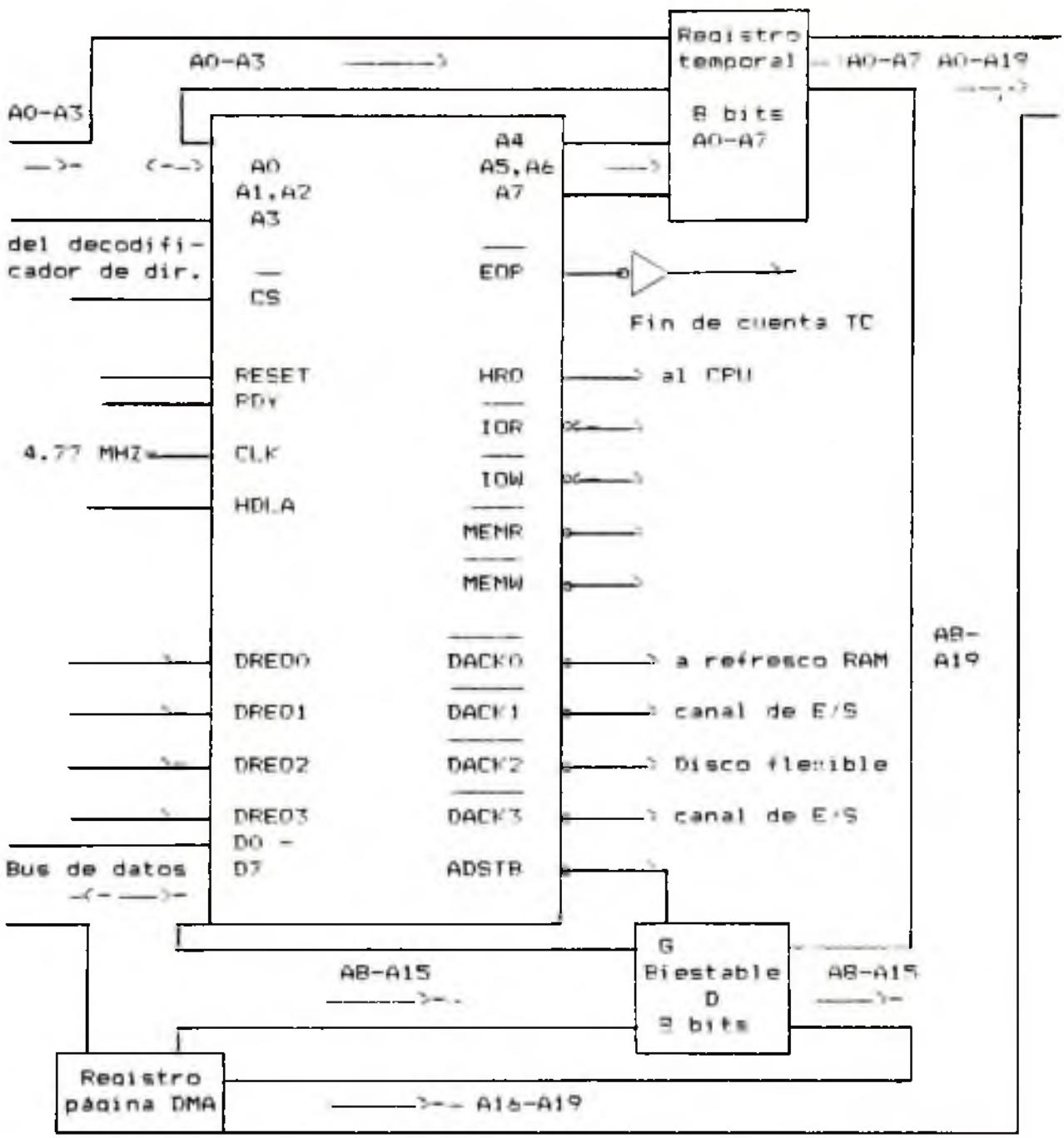


FIG V.5 DIAGRAMA A BLOQUES

El controlador se muestra en la figura anterior omitiendo algunas lineas. El integrado tiene 15 registros direccionables, una unidad temporizadora y de control, una unidad de control de comandos, un decodificador de prioridades y buffers. Este permite que dispositivos externos transfieran informacion hacia o desde el sistema de memoria sin la intervencion del procesador: también es posible transferir datos de una region de memoria a otra. El diseño del circuito es tal que el procesador asociado normalmente permanece ocioso por cuatro periodos de reloj durante cada transferencia de byte. Este diseño puede ser cambiado suministrando una señal a la pata de entrada RDY desde un generador de estados de espera. Aunque no se muestra en la figura anterior, el generador de estados de espera de la PC inserta un estado adicional de espera en todas las transferencias de bytes.

MODOS DE OPERACION.

Antes de realizar una transferencia de DMA, los canales a usarse deberan inicializarse. Existen cuatro modos de operacion: Transferencia sencilla, transferencia de bloque, transferencia en demanda y modo de transferencia en cascada. En el modo de transferencia sencilla se transfiere un solo byte, se decrementa el contador de palabra, se cambia la direccion en 1 y se liberan los buses. De esta manera se puede transferir un bloque de bytes por medio de transferencias sencillas sucesivas. Este es el método más usado.

La transferencia de bloques es altamente deseada para periféricos de gran velocidad, en este método no se liberan los buses hasta no haber terminado la transferecnia del bloque. Sin embargo, la transferencia de bloques no permite que otros dispositivos usen el DMA hasta que el bloque entero halla sido movido, y esto es inaceptable en la mayoria de los casos. Si se usara en la PC podria retardar demasiado el refresco de memoria. En la transferecnia en demanda se pueden mover datos hasta que el dispositivo de e:s halla agotado su capacidad. El modo en cascada se aplica para conectar multiples controladores que dan canales adicionales de DMA.

Transferencia Sencilla

Debido que es muy común y es el método usado en la PC, pondremos especial atención a la transferecnia sencilla. Es a menudo llamado ciclo de "robo", con los ciclos de transferencia de DMA obtenidos como un robo al procesador. Entre transferencias existe al menos un ciclo completo de maquina. En muchos casos el cilclo de robo ocurre cuando el procesador no esta usando los buses, en tal caso la operación de DMA es transparente para el procesador. Sin embargo no es posible hacer que un dispositivo ospere esta condicion para ahorrar ciclos. La mayoria de los periféricos son incapaces de esperar mas alla de un tiempo muy

corto entre cada transferencia de byte sin que cause un error en el sistema. Un ejemplo es el disco flexible.

Solamente se transfiere un byte durante una transferencia. Si DREQ se mantiene activa durante la transferencia sencilla, la linea HRQ permanece activa hasta después de la tranferencia, liberando los buses a la CPU. Sin embargo HRQ una vez mas se activa en espera de un nuevo HLDA para que otra transferencia sencilla ocurra. La operacion es repetida una y otra vez hasta que se halla movido el número de bytes especificado. Muchos periféricos, incluyendo los discos flexibles son suficientemente lentos para requerir multiples ciclos de bus entre transferencias.

Otro dispositivos pueden ser servidos con DMA entre transferencias sencillas. Si dos o más periféricos requieren DMA, el de mayor prioridad se servira primero. Claramente el maximo retardo que un periferico que tenga la mayor prioridad puede tener es de un ciclo de maquina. Las interrupciones pueden ocurrir entre transferencias activas de DMA y estas transferencias pueden ser implementadas durante rutinas de interrupción.

Cada canal tiene un registro contador de palabras el cual es decrementado cada vez que se mueve un byte. Cuando este llega a cero, significa que todos los bytes han sido movidos y un pulso de fin de cuenta es generado (TC). Este pulso termina la transferencia. Junto con el registro de cuenta actual, cada canal tiene un registro no direccionable llamado registro base de cuenta que contiene la cuenta inicial.

Como se muestra en la figura anterior la señal EOP (End of process) es invertida y alimentada al controlador de disquette como linea de control TC (terminated count), notificandole al controlador de disco flexible (FDC), que la operacion ha terminado. El FDC entonces realiza algunas tareas como poner ciertos datos dentro de algunos registro para que sean leidos por el procesador , seguido por una peticion de interrupcion (INT OEH) a través de la linea 6 del controlador de interrupciones. Esta sencilla interrupción notifica a la CPU que la operacion ha terminado y que ciertos resultados y cierta información se encuentra disponible para ser leidos.

DIRECCIONAMIENTO EN MEMORIA.

Para todas las transferencias se envia una direccion inicial en memoria desde la CPU al controlador de DMA via los buses de datos. Debido a que los registros de direcciones del controlador de DMA unicamente son de 16 bits, se requiere un registro externo de cuatro bits. Este es conocido como registro de pagina (bits A19-A16). Sus buses son flotados mientras HLDA esta inactivo.

Cada canal tiene su registro de direccion actual y un registro no direccionable de dirección inicial. La palabra de

direccion actual es automaticamente incrementada o decrementada despues de cada transfernecia de byte.

PROGRAMANDO EL 8237A

Todos lo puertos válidos son listados en la siguiente figura:

Direccion	Registro	bits	Lectura/Escritura
00H	Ch 0 dirección actual	16	lectura/escritura
01H	Ch 0 cont palabra actual	16	lectura/escritura
02H	Ch 1 direccion actual	16	lectura escritura
03H	Ch 1 cont palabra actual	16	lectura/escritura
04H	Ch 2 direccion actual	16	lectura/escritura
05H	Ch 2 cont palabra actual	16	lectura/escritura
06H	Ch 3 direccion actual	16	lectura/escritura
07H	Ch 3 cont palabra actual	16	lectura/escritura
08H	Registro de estado	8	solo lectura
08H	Registro de comandos	8	solo escritura
09H	Registro de peticiones	8	solo escritura
0AH	Registro de máscara(bit)	8	solo escritura
0BH	Registro de modo	8	solo escritura
0CH	Limpia first-last ff	1	solo escritura
0DH	Registro temporal	8	solo lectura
0DH	Limpieza maestra	0	solo escritura
0FH	Registro de mascara	8	solo escritura

FIG. V.6 PUERTOS UTILIZADOS POR EL CONTROLADOR DE DMA.

Para las direcciones de lo bits y registros de contadores, el byte menos significativo es accesado cuando el biestable "primero/ultimo" es 0, el byte mas significativo es accesado cuando el biestable es 1. Cada acceso de uno de estos registros automaticamente conmuta el estado del biestable. La instrucción OUT 0CH,AL limpia el biestable , no importando el valor que AL tenga. Se permite tanto escritura como lectura para registros de direcciones y contadores. Los valores iniciales deberan escribirse antes de comenzar la transferencia de DMA.

Registro de estado.- Los bits 7 a 4 del registro de estado estarán prendidos si una peticion de DMA se encuentra presente en los canales respectivos 3 a 0, y los bits 3 a 0 seran prendidos cada vez que un TC sea alcanzado en el canal correspondiente, Todos los bits son apagados por una operacion read o reset.

Registro de comandos.- La decodificacion del registro de comandos se da en la siguienete figura:

Bit 7 - estado de DACK	0 bajo	1 alto
Bit 6 - estado de DREQ	0 alto	1 bajo
Bit 5 - ancho pulso de escritura	0 normal	1 extendido
Bit 4 - Prioridad	0 fija	1 rotante
Bit 3 - Temporizador	0 normal	1 comprimido
Bit 2 - Fata CS	0 habilitado	1 deshabilitado
Bit 1 - Ch 0 address hold	0 deshabilitado	1 habilitado
Bit 0 - Transf memoria a memoria	0 deshabilitado	1 habilitado

FIG. V.7 DECODIFICACION DEL REGISTRO DE COMANDOS.

El bit 4 del registro de comandos puede ser prendido para establecer una prioridad rotante. El orden es siempre 01234, con el canal 0 teniendo siempre la mayor prioridad. Sin embargo despues de una operacion de DMA, los canales son puestos en la siguiente prioridad. La politica de prioridades fijas es la usada generalmente.

Registro de modo.- La decodificacion del registro de modo se da en la siguiente figura:

Bits 7,6 - Transferencia	00 demanda	01 sencilla	10 de bloque	11 cascada.
Bit 5 - Direccion	0 incrementa		1 decrementa	
Bit 4 - Autoinicializacion	0 habilitada		1 deshabilitada	
Bit 3,2 - 00 verify; 01 write; 10 read ; 11 no permitida				
Bit 1,0 - Seleccion de canal 00.0; 01.1; 10.2; 11.3				

FIG. V.8 DECODIFICACION DEL REGISTRO DE MODO.

Para refrescar la memoria escribimos un 58H (0101 1000B) en este registro, especificando transferencia sencilla, incremento en direccion, habilitacion de autoinicializacion, lectura de memoria y canal cero. La operacion es una pseudolectura sin transferencia de datos. La autoinicializacion se requiere para que la cuenta continue indefinidamente. Para una operacion de lectura a diskette se escribirá un 4eH (0100 1100B) . Este comando selecciona el canal 2 para una transferencia sencilla, con incremento de dirección y autoinicialización deshabilitada.

Registro de Peticion.- Los bits 1 y 0 de este registro son usados para seleccionar uno de los cuatro canales de DMA, y el bit 3 da una peticion activa de DMA si esta prendido. Todos los demas bits no tienen significado. El registro de peticion puede ser usado para inicializar por "software" una operacion de DMA. Este es limpiado por un reset y por un fin de cuenta TC.

Registro de mascara .- Esta asociado con dos numeros de puertos . Puerto 0AH para enmascarar o desenmascarar un solo canal, seleccionandolo con los bits 1 y 0. Un cero en el bit 3 apaga el bit de mascara y un 1 lo prende. Los bits restantes no tienen significado. Para el refresco de memoria debera escribirse un 00H en el puerto AH despues de la inicialización. Esta operacion limpia la máscara del canal cero, permitiendo que comience el refresco de memoria. Para la operacion del disco flexible debera escribirse un 02H en este mismo puerto.

Usando el puerto 0FH podemos escribir en todos los bits de máscara al mismo tiempo. Bits 3,2,1, y 0 se aplican a los respectivos numeros de canal, los restantes bits no tienen significado.

CONTROLADOR DE DISCO FLEXIBLE

La interfaz entre el "software" del procesador y el sistema mecanico del disco flexible es provista por la tarjeta controladora de disco. Esta tarjeta esta diseñada para manejar hasta cuatro unidades de disco flexible denominadas A, B, C y D.

Conexiones de la Tarjeta

En la siguiente figura se muestran las conexiones de la tarjeta controladora de disco flexible:

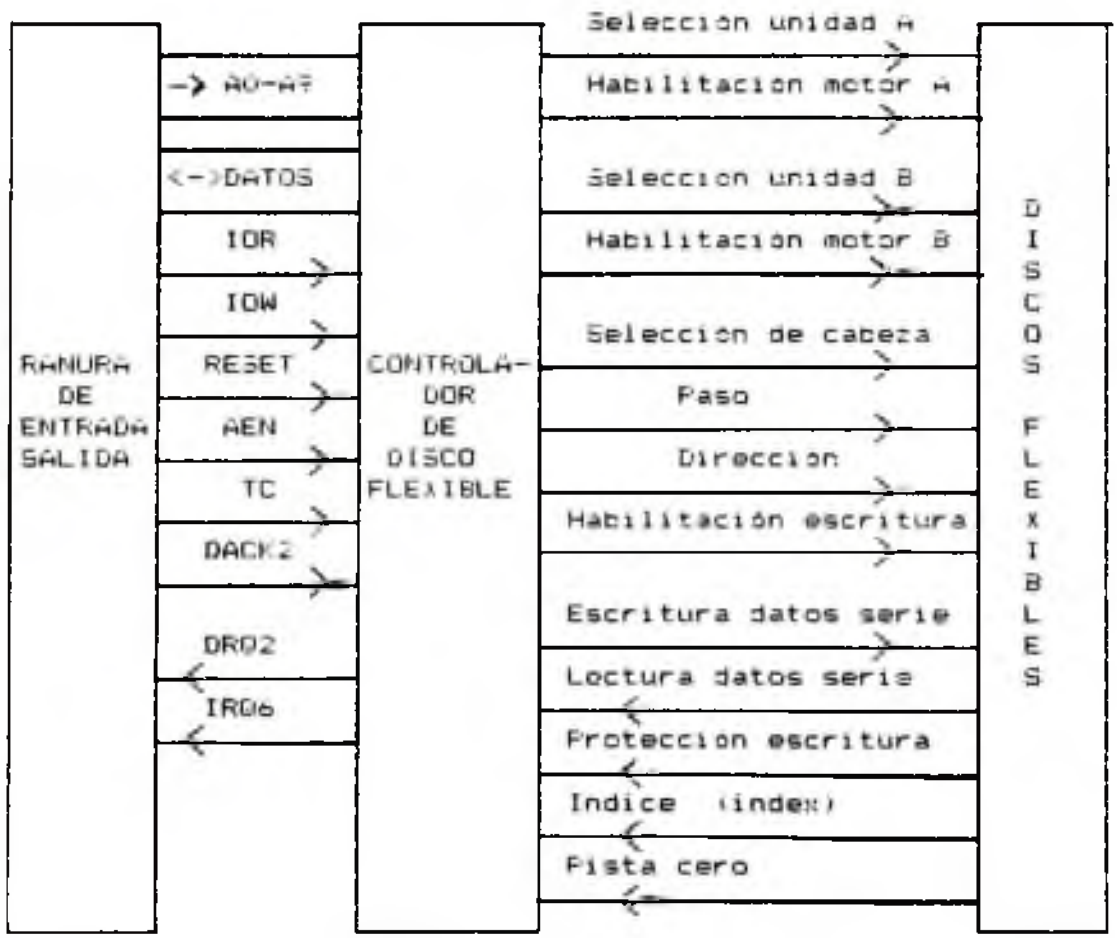


FIG. V.8 CONEXIONES DEL CONTROLADOR DE DISCO FLEXIBLE.

Las señales de lectura, escritura y reset son controladas por la CPU; habilitacion de direccion AEN es AEN invertida del generador de estados de espera; y las lineas TC y DACK2 proporcionan las señales de handshake del controlador de DMA. Estas seis señales de control son alimentadas del canal de E/S via el conector de la ranura de expansion. Tambien del lado izquierdo existen dos salidas del adaptador al canal de E/S. Una linea de petición de DMA (DRQ2) al canal 2 del controlador de DMA y una linea de interrupción (IRQ6) al nivel 6 del controlador de interrupciones.

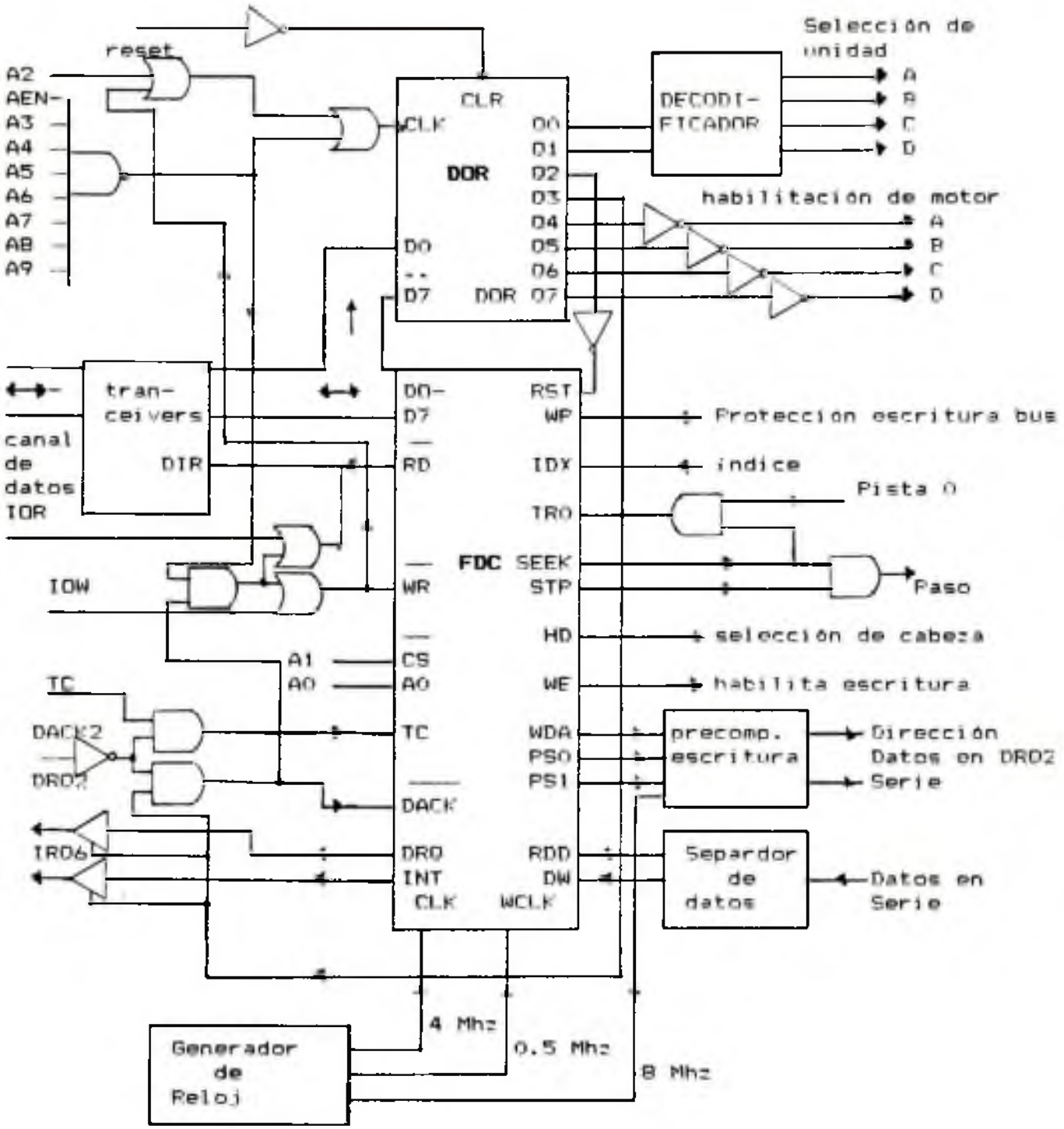


FIG. V.10 DIAGRAMA A BLOQUES

Dentro del controlador de disco flexible existen 8 registros accesibles a través del bus de datos; el registro de datos y el registro de estado. Para leer o escribir en el registro de datos se usa la dirección 3F5H y leemos el registro de estado en la 3F4H.

Registro digital de salida (DOR). Los bits 0 y 1 de este registro seleccionan la unidad A, B, C o D. Para prender el motor deberá mantenerse su línea de seleccion activa y la línea de habilitación de motor deberá estar en estado bajo. Un estado bajo en 02 pone el controlador en estado ocioso y reinicializa sus salidas de control. La línea de petición de interrupción y la de petición de DMA son habilitadas por el bit 03 y el medio octeto más alto provee las señales de habilitacion del motor. Todas las salidas del registro DOR son limpiadas por un reinicio del sistema.

Registros del controlador (FDC). En el registro de estado se guarda un byte para un control efectivo del procesador a través del puerto 3F4H. Su decodificacion se muestra a continuación:

Bit 7 (Registro de datos listo?)	0 no	1 si
Bit 6 (Registro de datos esc/lec)	0 esc.	1 lect.
Bit 5 (Modo NO DMA)	0 no	1 si
Bit 4 (LEC/ESC en progreso)	0 no	1 si
Bit 3 (Unidad 3 en modo posicionamiento)	0 no	1 si
Bit 2 (Unidad 2 en modo posicionamiento)	0 no	1 si
Bit 1 (Unidad 1 en modo posicionamiento)	0 no	1 si
Bit 0 (Unidad 0 en modo posicionamiento)	0 no	1 si

FIG V.11 DECODIFICACION DEL REGISTRO DE ESTADO.

En especial son importantes los bits 7 y 6 . El bit 7 indica cuando esta listo el registro de datos para recibir o enviar, este bit deberá examinarse antes de leer o escribir en este registro. Entonces el bit 6 indica si la siguiente operacion en el registro de datos será una lectura o escritura.

Si el FDC es programado por el comando SPECIFY para que opere en modo de NO DMA, el bit 5 del registro de estado es prendido en el registro de estado y el controlador de disco no proporcionara la señal de petición de DMA. En la PC el controlador de disco utiliza transferencia por DMA, y el bit 5 es apagado. El bit 4 indica si una operación de lectura o escritura se esta realizando. El bit correspondiente en el medio octeto menos significativo es prendido para indicar si la correspondiente unidad de disco se

encuentra ocupada moviendo su cabeza a una pista especificada en respuesta a un comando de SEEK.

El registro de datos del puerto 3F5H es un registro con una arreglo de pila. Algunos registros de la pila proporcionan informacion suplementaria al registro de estado del puerto 3F4H y otros almacenan parametros, comandos y resultados. Tambien el registro de datos es la fuente o destino del bus de datos durante las transferencias de DMA. El controlador puede ejecutar hasta 15 diferentes comandos. Para la mayoria de ellos, el resultado depende de ciertos parametros almacenados en la pila, los cuales deberán ser escritos dentro del registro de datos antes de la ejecución del comando. Los parametros proporcionan la identificación del sector y otra información.

Precompensacion de escritura.- La figura V.10 muestra un bloque de precompensacion de escritura. Esta técnica es usada para reducir el efecto del interferencia del flujo magnetico.

Separador de datos. Los bits leidos de un disco flexible incluyen no solo bytes de datos sino que también bits de sincronizacion y reloj, el patron especificado de bits usado como espacio intersectores (gap), y un identificador (ID) con información en forma de bytes indicando, el numero de cilindro, el número de cabeza, el numero de sector y la longitud física. En una lectura se lee un sector completo. El circuito integrado separador de datos provee una ventana que separa los datos de toda la información leida. Separados los datos entran al controlador FDC por la pata DW, mientras que todos los bits que fueron leidos entran por la pata RDD. Un registro de corrimiento de 8 bits dentro del FDC transforma cada byte serie a paralelo.

COMANDOS DEL FDC.

Algunos comandos tales como READ DATA, READ ID, y READ A TRACK transfieren datos del disco flexible al sistema principal, tambien existen varios comandos de escritura. Incluidos en el juego estan los comandos SCAN que comparan byte a byte datos almacenados en memoria con datos leidos del disco flexible. El comando SEEK posiciona la cabeza sobre la pista seleccionada del disco flexible. Otro comando RECALIBRATE posiciona la cabeza sobre la pista cero. El comando FORMAT A TRACK formatea una pista completa en un número de sectores y bytes especificados. Antes de que un disco flexible nuevo pueda ser usado deberan formatearse todas sus pistas.

La inicialización del sistema de disco se implementa meidante la escritura de una serie de datos al sistema de disco. Esta operación causa que se escriban datos apropiados al registro DOR y que el comando SPECIFY sea ejecutado. SPECIFY manda al registro de datos 3 bytes 03H, DFH y 02H. El primero de estos es el comando, el segundo indica un paso de 2ó milisegundos y un tiempo de desenergización de cabeza de 480 milisegundos, el tercer

byte indica una energización de cabeza de 4 milisegundos en operaciones de DMA. El comando RECALIBRATE deberá ejecutarse para cada unidad de disco flexible antes de la primera operación de lectura o escritura. Este inicializa la posición de las cabezas moviéndolas hasta la pista cero.

Resultados.- Después de su ejecución la mayoría de los comandos tienen una fase de resultados, provocada ya sea por la finalización del comando o por alguna causa de error. En esta etapa la CPU puede analizar los resultados del registro de datos del FDC. Estos deberán ser leídos sin demora, una vez que la cabeza esta en la identificación (ID) del campo siguiente a la última lectura, de otra manera el sistema de disco tendrá que ser reinicializado antes de que un nuevo comando sea aceptado. Una vez que se complete una operación, el ID del siguiente sector es almacenado, en cuatro bytes de resultados en registros de estado de la pila de datos.

UNIDAD DE DISCO FLEXIBLE.

Cada unidad de disco flexible cuenta con un motor de giro y un motor de paso, asociados con señales analógicas y digitales. Un pulso en la línea de entrada STEP moverá la cabeza magnética una pista hacia dentro o hacia fuera dependiendo del estado de la línea DIRECTION. Cuando la línea WRITE ENABLE esta en estado alto , una transición de bajo a alto en la línea de WRITE DATA causa un cambio en el flujo que es grabado por la cabeza en el disco. El el caso de lectura, el cambio de pulso encontrado por la cabeza genera un pulso que es suministrado a la línea de salida READ DATA.

Ejemplo:

PROGRAMA DE LECTURA EN DISCO

A continuación se muestra un programa que involucra transferencia entre disco flexible y memoria. Suponga que se desean leer 1024 bytes de los sectores 3 y 4, pista 7, lado 0 del disco 8, para ser almacenados en la dirección absoluta 0600H.

La transferencia deseada se puede lograr fácilmente con la INT 25H (lectura absoluta de disco), la cual usa la INT 13H (e/s de disco) del BIOS. Sin embargo deseamos describir el protocolo usado por la CPU, el controlador de DMA, el controlador de disco flexible, el controlador de interrupciones y el reloj, razón por la cual no usamos el BIOS.

```
program disk_read;
const
  dor          = $3f2;          {Digital output register}
  fdc_status   = $3f4;          {Status register }
  fdc_data     = $3f5;          {Data register}
  int_flag     = $80;           {Interrupt flag (1000 0000b)}
  motor_wait   = 37;
type phys_bytes =  byte;
var
  seek_status      : byte absolute $0040:$003e;
  motor_status     : byte absolute $0040:$003f;
  motor_count      : byte absolute $0040:$0040;
  diskette_status  : byte absolute $0040:$0041;
  nec_status       : array [0..6] of byte absolute $0040:$0042;
  answer           : string[2];
  buffer,apt_buffer:phys_bytes;
  i,j : integer;

{*****************************************************************}
{* Procedure real_address.- Este procedimiento calcula la direccion  *}
{* absoluta en 20 bits de la localidad de memoria apuntada por una va*}
{* riable tipo apuntador. El procedimiento inline de Turbo Pascal per*}
{* mite escribir codigo de maquina en linea.                         *}
{*****************************************************************}

procedure real_address(p:phys_bytes;var p2:phys_bytes);
begin
inline($5D/ $C4/ $4e/ $08/ $8C/ $C3/ $E1/ $E3/ $F000/ $33/ $d2/ $8a/
$D7/ $D1/ $E2/ $D1/ $E2/ $D1/ $E2/ $D1/ $E2/ $8C/ $C1/ $81/ $E1/ $0FFF/
$D1/ $E1/ $D1/ $E1/ $D1/ $E1/ $D1/ $E1/ $F3/ $13/ $C1/ $83/ $D2/ $00/
$C4/ $6E/ $04/ $26/ $89/ $5e/ $02/ $2e/ $89/ $46/ $00/ $5B/ $5D);
end;
{*****************************************************************}
function phys_address(p:phys_bytes):phys_bytes;
var apt_aux : phys_bytes;
begin
real_address(p,apt_aux);
phys_address := apt_aux;
end;
{*****************************************************************}
procedure enableinterrupts;
begin
inline($fb);
end;
{*****************************************************************}
procedure disableinterrupts;
begin
inline($fa);
end;
```

```
{*************************************************************************}
procedure fdc out(al:byte);
begin
while (port[FDC STATUS] and $80) = 0 do;{mientras no este listo DR}
port[FDC DATA]:= al;                    {escribe en registro de datos}
end;

{*************************************************************************}
{* Function wait for int. Permite saber si ya se completo una opera -*}
{* cion en disco.                                                      *}
{*************************************************************************}

function wait for int : boolean;
var intentos : integer;
begin
enableinterrupts;
intentos := 0;
writeln('esperando interrupcion');
while (intentos <=20) and (seek status or $80 = 0) do
  begin
  delay(100);
  intentos := intentos + 1;
  end;
if intentos = 21 then wait for int := false
else
  begin
  writeln('sillego interrupcion');
  SEEK STATUS := SEEK STATUS and $7f;
  wait for int := true;
  end;
end;
{*************************************************************************}
procedure dma setup(buffer:phys bytes);
var apt aux:phys bytes;
begin
apt aux := phys address(buffer);
port[$0c] := $46;
port[$0b] := $46;                       {modo de tranmision sencilla }
port[$81] := hi(seg(apt aux ));         {4 bits mas altos de direccion}
port[$04] := lo(ofs(apt aux ));         {8 bits mas bajos de direccion}
port[$04] := hi(ofs(apt aux ));         {8 bits siguientes de direccion}
port[$05] := $ff;                       {contador de bytes a mover bajo}
port[$05] := $03;                       {contador de bytes a mover alto}
port[$0a] := $02;                       {limpia canal 2 de DMA}
end;
{*************************************************************************}
procedure start motor;
begin
seek status := seek status and $7f;
motor count := 5*18;
port[dor] := $2d;                       {prende motor de unidad B}
delay(400);
end;
{*************************************************************************}
```

```
function recalibrate : boolean;
var intentos : byte;
   aux : boolean;
begin
aux := false;
intentos := 0;
while(intentos <= 10) and (aux = false) do
  begin
  fdc_out(07);                              {comando RECALIBRATE}
  fdc_out(01);                              {unidad B}
  aux := wait_for_int;
  intentos := intentos + 1;
  end;
recalibrate := aux;
end;
{*************************************************************************}
function seek : boolean;
var i : integer;
begin
fdc_out($0f);                 {comando SEEK}
fdc_out(01);                  {unidad B}
fdc_out(09);                  {pista 9}
seek := wait_for_int;
delay(2);                     {retardo de 2 milisegundos}
end;
{*************************************************************************}
function lee : boolean;
begin
fdc_out($66);                 {comando de lectura}
fdc_out($01);                 {unidad b}
fdc_out(09);                  {pista 9}
fdc_out(00);                  {cabeza 0}
fdc_out(03);                  {sector de inicio 3}
fdc_out(02);                  {codigo para 512 bytes por sector}
fdc_out(09);                  {ultimo sector de la pista}
fdc_out($2a);                 {gap #3 longitud = 42 bytes}
fdc_out($ff);                 {sin significado}
lee := wait_for_int;
end;
{*************************************************************************}
procedure results;
var i :byte;
begin
for i := 0 to 6 do
  begin
  while (port[FDC_STATUS] and $80) = 0 do;
  nec_status[i] := port[fdc_data];
  end;
end;
{*************************************************************************}
```

```
procedure accesa disco;
begin
writeln('dma setup');
dma_setup(buffer);
writeln('start motor');
start_motor;
writeln('recalibrate');
if recalibrate then
  begin
  writeln('seek');
  if seek then
     begin
     writeln('lee');
     if lee then
        begin
        results;
        for i := 0 to 6 do
          writeln('NEC_STATUS[',i,']:= ',NEC_STATUS[i]:2);
        end
     else
        writeln('error en lectura');
     end
  else
     writeln('ERROR EN POSICIONAMIENTO');
  writeln(seg(apt_buffer^),':',ofs(apt_buffer^));
  apt_buffer := buffer;
  for i:=1 to 1024 do
     begin
     write(char(apt_buffer^));
     apt_buffer := ptr(seg(apt_buffer^), ofs(apt_buffer^)+1);
     end;
  end
else
  writeln('ERROR EN RECALIBRACION');
end;
{*************************************************************************}
begin {main}
getmem(buffer,1024);
while answer <> 'si' do
begin
  writeln('  empezamos (si/no)?');
  readln(answer);
end;
clrscr;
start_motor;
for j := 1 to 5 do
  begin
  writeln('main');
  accesa_disco;
  end;
end.
```

Los mismos nombres de variables usados por el BIOS son usados con el atributo absolute. El bit más significativo de SEEK STATUS

es usado como una bandera de interrupcion, la cual esta designada como INT_FLAG. Cuando INT 0EH es activada por el controlador de disco flexible (FDC) a través de la linea 6 del controlador de interrupciones, este bit es prendido. En efecto INT 0EH no hace ninguna otra cosa. INT FLAG puede ser apagada por la instruccion SEEK_STATUS := SEEK_STATUS AND $7F; . Los otros bits de SEEK STATUS no tienen importancia para este programa. Una vez que se halla terminado una operacion de escritura o lectura en disco, deberan leerse 7 bytes de resultado del registro de datos del FDC.

Procedure FDC_OUT

Este procedimiento permite escribir al registro de datos. Prueba el bit 7 del registro de datos para ver si podemos escribir, si no es asi realiza una prueba más.

Function WAIT_FOR_INT

Esta funcion es verdadera si se produjo la interrupcion 0EH y es falso en caso contrario. Inmediatamente despues de la ejecucion de los comandos RECALIBRATE, SEEK y READ, el programa debera esperar hasta que el FDC este listo. Esto sucede prendiendo la bandera INT FLAG por medio de la interrupcion 0EH.

Procedimiento DMA_SETUP.

Las seis instrucciones port al DMA limpian el biestable first/last, especifican un modo de transferencia sencilla, escriben la direccion real en 20 bits de la direccion del buffer en memoria a recibir los datos, ponen la cuenta del registro contador en 5FFH (1023 BYTES), y limpian la mascara del canal 2 de DMA.

Modo de Transferencia Sencilla.

Durante una operacion de lectura o escritura, el disco flexible gira a una velocidad de 5 revoluciones por segundo. Las especificaciones indican que los bits son leidos a un promedio de 32 000 bytes o 256 000 bits por segundo. De tal manera que el tiempo requerido para leer un byte es de aproximadamente 31 microsegundos. Unicamente 5 periodos de reloj o 1 microsegundo son necesarios para mover un byte a memoria, dejando un intervalo de cerca de 30 microsegundos entre transferencias de DMA.

Usando transferencia sencilla , el procesador controla los buses durante los intervalos entre transferencias de bytes. Además pueden implementarse otras operaciones de DMA, en particular el refrescamiento de memoria a traves del canal 0, el cual tiene la mayor prioridad.

Despues de que un comando READ ha sido ejecutado por el FDC, el procesador examina la bandera INT FLAG hasta que indica que todos los bytes han sido transferidos. Durante este intervalo de tiempo, los bytes estan siendo movidos del disco flexible a

memoria a traves de DMA. Solo en el periodo individual de transferencia de un byte el procesador permanece ocioso.

Cuando el ultimo de los 1024 byte ha sido transmitido, el integrado de DMA envia un final de cuenta TC. Esta accion genera una señal alta en la pata INT, la cual activa INT 06H del controlador de interrupciones.

INT FLAG y arranque de motor

Es importante limpiar INT FLAG antes de ejecutar los comandos RECALIBRATE , SEEK y READ, ya que debera ser puesta en el momento apropiado por INT 0EH. El motor é es prendido con MOTOR COUN = 5*15. Este valor permite que el motor permanezca encendido por 5 segundos, lo cual es considerablemente mayor al tiempo necesario para realizar la lectura. Para prender el motor se escribe el byte 2DH al 03F para habilitar las salidas del controlador IRQ6 y DRQ2 y la entrada DACK2 del controlador de DMA.

Function RECALIBRATE.

El comando RECALIBRATE es implementado escribiendo dos bytes en se cuencia en el registro de datos del FDC, estos bytes son 07H, y 01H. El primero de estos bytes es el codigo de RECALIBRATE y el segundo selecciona la unidad de disco. Despues de que la cabeza ha sido desplazada al track 0, el FDC manda una interrupcion INT 0EH a traves de la linea IRQ6. Esta operacion prende el bit 7 de SEEK STATUS, el cual corresponde a INT FLAG. El llamado a WAIT FOR INTERRUPT pone el procedimiento en un lazo de espera hasta que la recalibración sea hecha.

Function SEEK

El comando SEEK mueve la cabeza a la pista deseada. Despues de que se mandan los tres bytes al registro de datos, el programa debera esperar hasta que el comando sea ejecutado. Esto es indicado por el nivel 6 de interrupción.

Operacion de lectura

Para activar el comando READ deben escribirse 9 bytes en el registro de datos del FDC, el significado de cada byte se indica en el programa. El byte 8 es el valor de gap (distancia entre sectores).

El ultimo byte no tiene significado pero debe ser enviado, despues de esta operación, el comando es ejecutado automaticamente y el programa entonces espera por la señal del FDC que indique que la operacion de lectura ha terminado.

La transferencia de datos principia cuando el ultimo byte ha sido recibido por el registro de datos y se utiliza la transferencia sencilla. El procedimiento WAIT FOR INTERRUPT puede ser rediseñado, para ejecutar procedimientos dentro del lazo de espera. For ejemplo, entre transferencias de bytes es posible leer datos recibidos por el puerto de comunicaciones o enviar datos a la impresora. El único requerimento es que la rutina

debera examinar periódicamente la bandera INT FLAG al menos una vez cada 30 microsegundos mientras la lectura se realiza.

Resultados

Mientras el motor se encuentre prendido, deberan leerse 7 bytes de resultados del registro de datos del FDC. Estos datos deberán almacenarse en memoria. Antes de cada lectura al registro de datos, es necesario esperar hasta que este listo, lo cual es indicado por un uno en el bit 7 del registro de estado del FDC.

Los resultados deberan ser leidos sin retardo, mientras la cabeza este leyendo todavia informacion del ID del disco flexible. De otra manera sera necesaria la reinicializacion del FDC antes de que pueda realizarse otra escritura o lectura, lo cual se logra con los comandos SPECIFY y RECALIBRATE.

Los resultados leidos en orden son 1, 0, 0, 5, 0, 5 y 2. Los 3 primeros bytes de estado indican que la unidad B no tuvo errores. El byte 4 indica la pista actual en la que esta posicionada la cabeza, el 0 siguientes indica la cabeza correspondiente al lado cero, el byte 5 indica el sector actual donde esta la cabeza y el ultimo byte 2 indica 512 bytes por sector. Después de leer el ultimo byte, el bit 6 de registro de estado cambia de 1 a cero para indicar que el registro de datos esta listo para recibir un nuevo comando.

Escritura a Disco

Con ligeras modificaciones el programa puede ser usado para escribir a disco, pero debera insertarse un retardo de .5 segundos inmediatamente despues de prender el motor.

Existen dos cambios adicionales que deberan hacerse en el programa. El comando 46H en la rutina de DMA debera cambiarse por 4AH, y el comando de lectura al FDC 6cH debera cambiarse por el comando de escritura 45H. Con estos cambios el programa escribira 1024 bytes de memoria a los sectores 3 y 4 de la pista 4 lado 0 de la unida B.

IMPLEMENTACION DE UN MANEJADOR DE DISCO.

IMPLEMENTACION DE UN MANEJADOR DE DISCO.

Cuando se manejan procesos las rutinas de entrada/salida deben quedar disponibles para todos, pero en el caso de las de Pascal y DOS estas rutinas no fueron pensadas asi, por lo que es necesario diseñar y construir manejadores de entrada/salida con esta filosofia.

En este capitulo en especial abordamos el problema del manejador de disco que puede servir de base para la implementacion del manejador de archivos, por ahora solo se construyen rutinas para accesar un bloque seleccionado en disco flexible.

Todos los discos estan organizados en cilindros, cada uno con tantas pistas como cabezas verticales tenga. Las pistas estan divididos en sectores, tipicamente entre 8 y 32. Todos los sectores contienen el mismo numero de bytes. Aunque las pistas mas cercanos a la orilla del disco son fisicamente mas grandes este espacio extra no se utiliza.

Los parametros del disco flexible de la PC se muestran en la figura VI.1. Estos son los parametros de los discos de doble lado, doble densidad usados por el manejador de disco construido. El manejador utiliza bloques de 1024 bytes, asi que los bloques usados por este "software" consisten de dos sectores consecutivos, los cuales siempre son leidos o escritos como una unidad (cluster).

		msec
Numero de cilindros : 40	Tiempo de posicionamiento (vecino)	6
Pistas por cilindro : 2	Tiempo de posicionamiento (promedio)	77
Sectores por pista : 9	Tiempo de rotacion	200
Sectores por disco : 720	Tiempo de arranque/parada de motor	250
Bytes por sector : 512	Tiempo de transferencia 1 sector	22
Bytes por disco : 368640		

FIG VI.1 PARAMETROS DISCO FLEXIBLE IBM PC.

El tiempo de lectura o escritura de un bloque a disco esta determinado por tres factores : el tiempo de posicionamiento de la cabeza en el cilindro deseado, el tiempo de rotacion (tiempo necesario para que el sector deseado gire debajo de la cabeza), y el tiempo requerido para la transferencia. Para la mayoria de los sistemas el mas significativo es el de posicionamiento de la cabeza.

MANEJO DE ERRORES.

Los discos flexibles estan expuestos a una gran variedad de errores. Algunos de los mas comunes son :

- Errores de programacion (solicitar un sector inexistente)
- Errores de verificación (por ejemplo causado por cabeza sucia)
- Error de verificacion permanente (block fisicamente dañado)
- Error de posicionamiento (por ejemplo la cabeza fue enviada al cilindro c pero fué al 7)
- Error del controlador (el controlador se rehusa a aceptar comandos).

Es tarea del manejador de disco controlar estos errores lo mejor posible.

Los errores de programación suceden cuando el manejador le dice al controlador que busque un cilindro inexistente, lea de un sector inexistente, use una cabeza inexistente o transfiera de o hacia una localidad de memoria inexistente. La mayoria de los controladores checan los parametros.

Los errores de checksum son causados por cabezas sucias. La mayoria de las veces este error puede ser corregido intentando la operacion varias veces. Si el error persiste hay que marcar el bloque como malo y habra que evitarlo.

Una manera de evitar los bloques dañados es escribir un programa muy especial que forme una lista con los bloques malos y cuidadosamente los maneje en un archivo de bloques dañados. Una vez que este archivo ha sido creado, el localizador de disco no los podra usar ya que se marcarán como ocupados.

Los errores de posicionamiento son provocados por problemas mecanicos en la cabeza y el controlador se brinca pistas a la hora de posicionar la cabeza. Para realizar el posicionamiento de la cabeza (seek) , se proporciona una serie de pulsos al motor de paso, un pulso por cada cilindro hasta llegar al cilindro deseado. Cuando la cabeza alcanza su destino , el controlador lee el numero de cilindro actual (escrito cuando el disco fué formateado), si la cabeza esta en un lugar erroneo sucedio un error de posicionamiento.

Algunos controladores corrigen los errores de posicionamiento automaticamente, pero otros (incluyendo el de la IBM PC) unicamente prenden un bit de error y dejan el resto al manejador de disco. El manejador trata este error mandando un comando de RECALIBRACION, para mover la cabeza hasta el cilindro mas externo correspondiendo al 0. Normalmente esto resuelve el problema pero si no es asi la unidad de disco flexible tendra que ser reparada.

Como hemos visto el controlador es realmente un pequeño procesador especializado, con "software", variables y buffers. Algunas veces una secuencia inusual de eventos, causa que el controlador quede en un bucle o pierda la pista de lo que estaba haciendo, tal como cuando ocurre una interrupcion en una unidad de disco simultaneamente con una recalibracion en otro. Los diseñadores generalmente preveen lo peor y dejan una pata en el integrado o tableta, que cuando se pone en alto, forza al controlador a olvidar cualquier cosa y se reinicia. Si todo falla, el manejador de disco puede prender un bit para invocar esta señal y reiniciar el controlador.

MANEJADOR DE DISCO EN TURBO PASCAL.

El manejador de disco implementado acepta y procesa dos tipos de mensajes: para escribir un bloque y para leer un bloque. Un bloque es de tamaño BLOCK_SIZE, el cual es definido como 1024 bytes. El tamaño del sector en el disco es de 512 bytes, de tal manera que siempre son leidos o escritos dos sectores consecutivos. La ventaja de un bloque de mayor tamaño es una reducción en el numero de accesos a disco requeridos y por lo tanto un mejor rendimiento (performance). El precio que se paga es que para traer un solo caracter se manejaran 1024 bytes.

El mensaje aceptado por el manejador de disco usa el siguiente formato :

Destino (manejador de disco)	Sector (0 a 7)
Clase (escritura, lectura)	Bytes (1024)
Dispositivo (unidad A o B)	Address (direccion de buffer)
cabeza (0 o 1)	Count(bytes transferidos)
cilindro (0 a 39)	Rep_satus (resultado de operac.)

FIG VI.2 FORMATO DE MENSAGE MANEJADO POR EL MANEJADOR DE DISCO.

El mensaje de respuesta al proceso que solicito el acceso a disco contiene el numero de bytes que se tranfirieron o un código de error si su petición no se llevó a cabo.

El manejador de disco es estrictamente secuencial, acepta una petición a disco y no recibe otra hasta terminar la primera. La razon

por la cual se tomo esta decision es que por ser implementado el Pascal Concurrente para PC es dificil que en un momento dado existan muchos procesos activos y siendo asi la probabilidad de acceso a disco es pequeña y no tiene caso complicar el algoritmo de acceso a disco. Un manejador de disco para un sistema grande en tiempo compartido debera obviamente ser implementado en forma diferente.

El procedimiento principal del manejador de disco (ver apendice C), **floppy task**, acepta mensajes para realizar el trabajo y envia respuestas en un lazo sin fin (ver apendice C). El trabajo para realizar una escritura a disco es casi identico al de lectura a disco, asi que son manejados por el mismo procedimiento **do rdwt**.

La figura VI.2 muestra la relación entre los procedimientos principales del manejador de disco. Bajo condiciones normales (cero errores) **do rdwt** llama cuatro procedimientos más, cada uno realizando una parte del trabajo de la transferencia.

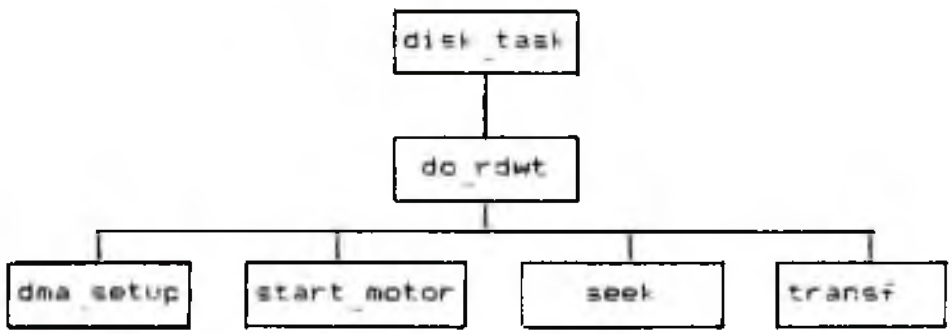


FIG. VI.4

El primer procedimiento llamado **dma setup** programa los registros del controlador de DMA para que lleve a cabo la transferencia de memoria a disco o de disco a memoria sin intervención de la CPU.

El siguiente procedimiento, **start motor** checa si el motor esta prendido. Si es asi el procedimiento no hace nada, si el motor esta apagado lo prende.

El procedimiento **seek** checa si la cabeza se encuentra posicionada sobre el cilindro adecuado, si no, instruye al controlador de disco para que lo haga, y entonces espera la interrupcion del controlador de disco que indique que la operación se completo.

El comando de lectura o escritura es ejecutado por el procedimiento **transf** y tambien espera por la interrupcion de controlador de disco que indica que la transferencia se lleve a cabo. Cuando se ha completado el comando **transf** inspecciona los registros de estado del controlador para ver si no existio algun error. Si ocurrió algún error de verificación , el procedimiento regresa un código de error a **do rdwt**, de tal manera que **transf** pueda intentarlo una vez más.

Por ultimo debe apagarse el motor despues de una operacion. Los discos flexibles no pueden leerse o escribirse si el motor esta apagado. El encendido y apagado de un disco lleva su tiempo, podria pensarse en dejar prendido el motor todo el tiempo, pero esto provocaria el prematuro desgaste de cabezas y discos, asi que el compromiso es dejar el motor prendido algunos segundos (por ejemplo 3) despues de cada lectura o escritura, de tal manera que si la unidad de disco es usada una vez mas dentro de esos tres segundos, el tiempo se extiende otro periodo de tiempo igual, pero si ya no se usa el disco en este periodo, se apaga.

El proceso **clock** del KERNEL cada vez que se cumple un tick decrementa el tiempo que esta encendido el motor de la unidad de disco flexible, si este se agote instruye al controlador para que apague el motor de la unidad de disco.

Algunos procedimientos complementarios usados en el controlador de disco, se listan abajo.

1.- phys_address.- Dado un apuntador a memoria calcula su dirección fisica (20 bits).
2.- fdc_out.- Manda un comando al controlador.
3.- fdc_results.- Extrae los resultados de un comando.
4.- recalibrate.- Recalibra una unidad de disco despues de un error de posicionamiento.
5.- resetf.- inicializa el controlador despues de un error serio.

IMPLEMENTACION

La estructura principal usada por el controlador de disco, es llamada **floppy** (ver apendice C), la cual es una arreglo de estructuras (una por unidad de disco flexible). Cada una contiene informacion acerca del estado actual de su unidad de disco y el comando a ejecutar, de tal manera que guardar la dirección en el disco, la direccion en memoria, la informacion acerca del estado del controlador y el estado de su calibración.

El procedimento que lleva el verdadero trabajo es **do_rdwt**, y maneja como parametro el mensaje precisamente recibido. Lo primero que hace es calcular la estrcutura correspondiente a la unidad de disco a manejar, entonces copia los parametros de cilindro, pista, sector y cabeza a la estructura. De aqui en adelante la estructura apuntada por **fp** contiene toda la informacion necesaria para la operacion. Dentro de este procedimiento existe un lazo que permite repetir varias veces una operacion en caso de error y tambien checa si es necesario reinicializar el controlador, si es asi llama el procedimiento **resetf**. Si alguno de los procedimientos llamados por **do_rdwt** descubre que el controlador ya no responde, prende la bandera need_reset y en el siguiente ciclo se "reiniciara" el controlador. La transferencia se lleva a cabo por un llamado a **transf** y si esta se completa exitosamente se abandona el lazo.

El procedimiento **dma setup** carga la direccion de memoria y la cuenta de bytes a transferir dentro del controlador de DMA. La dirección dada mas la cuenta no deben rebasar fronteras de 64 kbytes, esto es un buffer de 1 kbyte de DMA puede empezar en la direccion 6451D pero no en la 6514 porque este se extenderia mas alla de la frontera de 6553z. Esta restriccion se debe a que la PC utiliza un viejo controlador de DMA, que contiene contadores de 16 bits en lugar de los 20 necesitados ya que DMA utiliza direcciones absolutas y no direcciones relativas a segmentos. Los 16 bits menos significativos de la direccion de DMA son cargados dentro del 9237A y los 4 mas significativos en "latches", el pasar de la cuenta FFFF a la 0000 no genera acarreo.

El procedimiento **start motor** controla los motores de las unidades de disco flexible. Al entrar deshabilita interrupciones temporalmente mientras checa el estado del motor y calcula su nuevo estado. Los dos bits de menor orden de la variable motor goal contienen el numero de unidad seleccionada. Los dos siguientes bits ponen al controlador en modo normal (interrupciones habilitadas). Los cuatro bits de mayor orden contienen el estado de los cuatro motores que puede manejar el controlador, un 1 significa que el motor esta prendido y un cero que el motor esta apagado.

Si el motor esta apagado es necesario provocar un retardo mientras se arranca (aprox. 250 mseg).

El procedimiento **seek** primero checa si la unidad de disco esta calibrada, si no lo esta la recalibra. Si el cilindro actual es el deseado unicamente regresamos del procedimiento, de otra manera manda un comando seek y espera por la interrupcion del FDC. Despues de que la interrupcion llegó checa los resultados llamando a **fdc results**. Desafortunadamente aun el reporte de estado puede fallar, ya que él mismo es un comando que el FDC puede aceptar o no. Si el reporte de estado regresa en forma normal e indica un error en el comando seek, entonces la unidad de disco debera recalibrarse.

El procedimiento **transf** es quien comanda propiamente al FDC para que empieze la lectura o escritura. Para ejecutar el comando se mandan 9 bytes de información al controlador. Despues de que se mando el comando se espera la interrupcion del FDC, se obtienen los resultados y se checa para detectar errores.

El leer los resultados no implica solamente leer una palabra o dos del controlador de disco. Se requiere un complejo protocolo de comunicación con el controlador en **fdc results.** Todas las cosas que pudieran ir mal deberan de checarse y si esto no fuera suficiente el tiempo de negociacion de este protocolo tambien es importante.

Aún el solo hecho de sacar un byte de informacion al controlador es complicado y requiere un procedimiento completo **fdc out.** El problema es que el controlador piensa en forma independiente y no se le puede obligar a aceptar un comando. Existe una negociación compleja para determinar cuando esta en condiciones de aceptarlo y

cuando no. La recalibracion de una unidad de disco y el "reinicio" son realizados por **recalibrate** y **resetf** respectivamente.

En general la tarea del manejador del disco es conceptualmente simple, pero llena de detalles, algunos de los cuales son inherentes al dispositivo de entrada/salida y otros a que el controlador **PD765** es demasiado primitivo.

En la implementación del manejador de disco se utilizaron dos opciones para recibir la interrupcion generada por el controlador de disco. La primera siguiendo la filosofía del BIOS (BASIC INPUT OUTPUT SYSTEM) de IBM y la segunda recibiendo la interrupcion directamente a través de un llamado a IOTRANFER. A continuación se explican cada una de ellas a detalle.

Primera opcion:

```
{=============================================================}
{=                        WAIT FOR INT                       =}
{=============================================================}

procedure wait_for_int;
var retries:integer;
begin
enableinterrupts;
retries := 0;
if not need_reset then
   begin
   while (retries < 36) and (seek_status and $80 = 0) do
      begin
      retries := succ(retries);
      waittime(tick);
      end;
   if (retries >= 36 ) or (retries < 0)then need_reset := true;
   end;
Seek_status := seek_status and $7f;
end;
```

En la rutina anterior observamos que la variable seek_status se examina periodicamente cada tick (aprox. cada 55 milisegundos), si la interrupcion se produjo, esta variable tendra el bit más significativo con un valor de uno de otra manera será cero. ¿Pero quien prende esta bandera?, en realidad esta bandera la sigue manejando una rutina del BIOS mucho muy sencilla que lo unico que hace es precisamente prender esta bandera cuando se produce la interrupción del disco 0EH. Ahora la diferencia de nuestra rutina wait_for_int de la del BIOS es que entre cada vez que se examina la bandera liberamos el procesador con la primitiva waittime para que otro proceso haga uso de el, cosa que en BIOS no sucede.

El procedimiento wait_for_int debera apagar la bandera una vez que tome conocimiento de ella. En caso de que la interrupcion no llegará por alguna razón (como por ejemplo que la unidad de disco

tenga la compuerta abierta), al intento 35 se abandonara la tentativa de acceso a disco.

Segunda opcion:

```
{=============================================================}
{=                        WAIT_FOR_INT                       =}
{=============================================================}

procedure wait_for_int;
begin
if seek_status and $80 = 0 then waitio($0e,6);
Seek_status := seek_status and $7f;
end;
```

En la segunda rutina observamos que la variable seek status se examina unicamente una vez y si esta no ha sido prendida por FIOS, nuestro procedimiento hace a un lado la rutina de bios y espera directamente la interrupcion 0EH a traves del primitiva waitio.

El procedimiento wait for int todavia debera apagar la bandera una vez que tomo conocimiento de ella. En este caso seria critico que la interrupcion no llegara ya que nos quedariamos esperandola para siempre, para prevenir esto el manejador de disco antes de llamar este procedimiento siempre se asegura que el FDC nos este escuchando. Una solucion a esto podria implementarse haciendo uso de las rutinas para manejo de temporizadores, poniendo a funcionar un temporizador que despues de un tiempo dado provoque la interrupcion e indique lo que paso.

Las primitivas que se dejan finalmente para manejar un bloque de disco y que serian usadas por el manejador de archivos son las siguientes:

```
function leetrack(drive,head,cilynder,sector,buffer1):integer;

function escribetrack(drive,head,cylinder,sector,buffer):integer
```

donde:

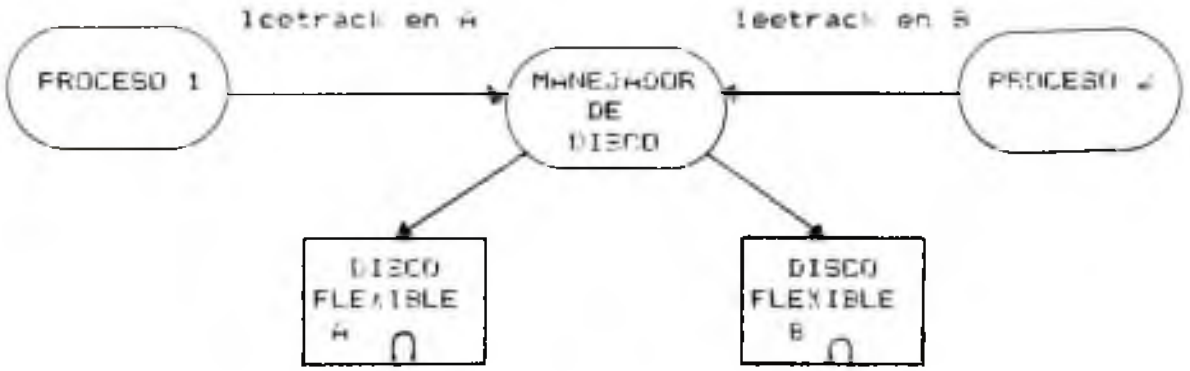
drive .- Es la unidad de disco (0 = A, 1 = B).
head .- Cabeza o lado del disco (0 o 1).
cylinder .- Cilindro (0-39).
sector .- Sector (0-7).

A continuacion se muestra dos ejemplos que fueron probados con las dos rutinas de wait_for_int y se muestran los resultados obtenidos.

EJEMPLO 1.-

Se crearon dos procesos que accesan aleatoriamente los discos A y B respectivamente. La lectura de sectores se hace en todo el disco con la intención de que la cabeza se desplace libremente por todo la superficie del disco y darnos una idea de los tiempos de busqueda de sectores. Recordemos que el tiempo de posicionamiento es uno de los que mas influye en la eficiencia de un controlador de disco. Para contabilizar el numero de accesos se crea un tercer proceso que cada minuto despliega la cuenta de accesos realizados. Por su parte el programa principal incrementa un contador y lo despliega mostrando que trabaja concurrentemente con los procesos que accesan disco.

Esquematicamente el ejemplo 1 puede verse asi:




```
{*************************************************************}
{*                          DISK4.PAS                        *}
{* Programa que nos demuestra como leer disco directamente  sin *}
{*  hacer uso  de  la INT 13H  del BIOS. Se  crean dos  procesos *}
{*  que accesan disco,  uno del drive  A y el  otro del drive  B. *}
{* El proceso main  muestra el numero de accesos a disco       *}
{*************************************************************}

program disk_read;
{$k-}
{$I nucleo.ext}                 {Primitivas del nucleo}
{$I kernel.pas}                 {Primitivas del KERNEL}
{$I diskdriv.pas}               {Manejador de disco}
```

```
var
  count  : integer;
  accesos : integer;
  p1,p2,p3 : processref;
  answer          : string[2];
  buffer1,buffer2 :  byte;

{*************************************************************************}
procedure recep1;
begin
while true do
  begin
  if leetrack(0,random(2),random(40),random(3)+1,buffer1)=block size
  then
     begin
     accesos := succ(accesos);
     end
  else writeln('FALLO LECTURA');
  waittime(1*tick);
  end;  {end while true}
end;
{*************************************************************************}
procedure recep2;
begin
while true do
  begin
  if leetrack(1,random(2),random(40),random(3)+1,buffer2)=block size
  then
     begin
     accesos := succ(accesos);
     end
  else writeln('FALLO LECTURA');
  waittime(1*tick);
  end;  {end while true}
end;
{*************************************************************************}
procedure reloj;
var minutos : integer;
begin
minutos := 0;
accesos := 0;
while true do
  begin
  gotoxy(25,16);
  writeln(accesos,' accesos a disco en ',minutos:2,' minutos');
  gotoxy(33,18);
  writeln(count:6);
  waittime(1*min);
  minutos := succ(minutos);
  end;  {end while true}
end;
{*************************************************************************}
```

```
begin {main}
getmem(buffer1,1024); {direccion de buffer
getmem(buffer2,1024); {direccion de buffer
answer := 'no';
count := 0;
while answer <> 'si' do
begin
  writeln('  empezamos (si/no)?');
  readln(answer);
end;
initkernel;
initdiskdriver;
p3:=createprocess(ofs(reloj),2048);
p1:=createprocess(ofs(recep1),2048);
p2:=createprocess(ofs(recep2),2048);
clrscr;
gotoxy(25,08);
writeln(' PROBANDO MANEJADOR DE DISCO 1');
while true do
  begin
  gotoxy(32,14);
  writeln('count = ',count:5);
  waittime(1*tick);
  count := succ(count);
  end;
end.
```

Resultados obtenidos con el ejemplo 1 empleando las dos opciones del controlador de disco. La variable COUNT nos da una idea de la forma en que se compartió el procesador entre el manejador de disco y otros procesos:

	OPCION 1		OPCION 2	
MINUTOS	ACCESOS A DISCO	VARIABLE COUNT	ACCESOS A DISCO	VARIABLE COUNT
1	159	930	215	1065
2	341	1960	437	2158
3	526	3000	673	3197
4	697	4012	913	4235
5	866	4979	1159	5328
6	1038	5966	1411	6420
7	1211	6946	1663	7513
8	1378	7900	1914	8606
9	1550	8903	2157	9672
10	1725	9915	2415	10765
11	1905	10995	2661	11858
12	2066	11897	2912	12951
13	2223	12787	3155	14016
14	2385	13744	3389	15082
15	2569	14765	3643	16175
16	2744	15759	3883	17241
17	2927	16799	4123	18307
18	3085	17730	4368	19373
19	3250	18724	4614	20439
20	3410	19636	4861	21532

EJEMPLO 2.-

En esta ocasion se crean dos procesos uno que que lee sectores del drive B en forma aleatoria y otro que escribe el bloque leido en la misma posicion pero en el drive A. Al igual que en el ejemplo anterior un tercer proceso sigue llevando la cuenta de accesos a disco y el programa principal despliega un contador. Los procesos que leen y escriben solo estan sincronizados por tiempo.

```
{*************************************************************************}
{*                                DISK5.PAS                              *}
{* Programa que nos demuestra como leer y escribir disco direc- *}
{* tamente sin hacer uso de la INT 13H. Se crean dos procesos uno*}
{* que lee el disco del drive B y otro que escribe al drive A. El*}
{* proceso reloj muestra el numero de accesos a disco           *}
{*************************************************************************}

program disk_read;
{$k-}
{$I nucleo.ext}
{$I kernel.pas}
{$I diskdriv.pas}

var
  count  : integer;
  accesos : integer;
  p1,p2,p3 : processref;
  answer          : string[2];
  buffer1 :  byte;
  nhead,ncylinder,nsector : integer;
{*************************************************************************}
procedure recep1;
var ndrive : integer;
begin
ndrive := 1;
while true do     {unidad de disco B}
  begin
  nhead := random(2);
  ncylinder := random(40);
  nsector := random(8) + 1;
  if  leetrack(ndrive,nhead,ncylinder,nsector,buffer1)  =  block_size
  then
     begin
     accesos := succ(accesos);
     end
  else writeln('FALLO LECTURA');
  waittime(1*tick);
  end;  {end while true}
end;
{*************************************************************************}
```

```
procedure recep2;
var ndrive : integer;
begin
ndrive :=0 ;      {unidad de disco A}
while true do
  begin
  if  escribetrack(ndrive,nhead,ncylinder,nsector,buffer1)=block_size
  then
     begin
     accesos := succ(accesos);
     end
  else writeln('FALLO ESCRITURA');
  waittime(1*tick);
  end;  {end while true}
end;
{*************************************************************************}
procedure reloj;
var minutos : integer;
begin
minutos := 0;
accesos := 0;
while true do
  begin
  gotoxy(25,16);
  writeln(accesos,' accesos a disco en  ',minutos:2,' minutos');
  gotoxy(33,18);
  writeln(count:6);
  waittime(1*min);
  minutos := succ(minutos);
  end;  {end while true}
end;
{*************************************************************************}
begin {main}
getmem(buffer1,1024); {direccion de buffer}
answer := 'no';
count := 0;
while answer <> 'si' do
begin
  writeln('  empezamos (si no)?');
  readln(answer);
end;
initkernel;
initdiskdriver;
p3:=createprocess(ofs(reloj),2048);
p1:=createprocess(ofs(recep1),2048);
p2:=createprocess(ofs(recep2),2048);
clrscr;
gotoxy(25,08);
writeln(' PROBANDO MANEJADOR DE DISCO 1');
while true do
  begin
  gotoxy(32,14);
  writeln('count =  ',count:6);
  waittime(1*tick);
```

```
    count := succ(count);
    end;
end.
```

Resultados obtenidos con el ejemplo 2 empleando las dos opciones del controlador de disco:

	OPCION 1		OPCION 2	
MINUTOS	ACCESOS A DISCO	VARIABLE COUNT	ACCESOS A DISCO	VARIABLE COUNT
1	171	941	210	1039
2	351	1917	449	2095
3	520	2875	697	3150
4	715	3914	941	4207
5	884	4880	1192	5275
6	1067	5865	1432	6341
7	1236	6798	1672	7407
8	1397	7729	1933	8500
9	1580	8704	2189	9572
10	1749	9635	2435	10648
11	1921	10565	2670	11741
12	2100	11523	2925	12807
13	2287	12507	3172	13900
14	2464	13464	3421	14993
15	2621	14373	3672	15086
16	2798	15331	3914	17178
17	2965	16316	4172	18261
18	3105	17121	4416	19317
19	3284	18115	4650	20383
20	3460	19050	4883	21375

Observando los resultados de los dos ejemplos nos damos cuenta que la segunda opcion es más eficiente, ¿pero por que?, pues porque la primera opcion utiliza un metodo de "pooling", y así, si no se detecta la bandera prendida de interrupcion el procesador conmuta a otro proceso, pero si en ese momento se recibe la interrupcion el controlador de disco no se enterará inmediatamente sino hasta que halla transcurrido el tiempo indicado para volver a examinar la bandera.

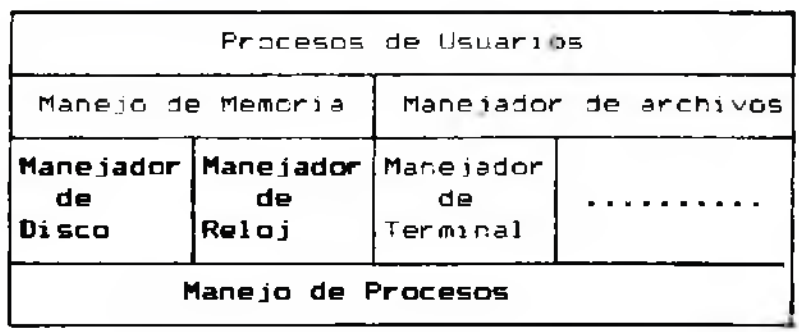
Con la segunda opción cuando se recibe la interrupcion se conmuta inmediatamente al controlador de disco para terminar la operación, esto se debe a que el manejador de disco tiene una prioridad mayor que la de los demas procesos, de tal manera que se asegure que se le atenderá inmediatamente que lo requiera.

CONCLUSIONES.

CONCLUSIONES.

Con este trabajo quedan las bases y herramientas para manejar procesos desde un lenguaje de alto nivel como Pascal. Puede aplicarse en general al monitoreo y control de eventos, sin embargo dependiendo de la aplicacion sera todavia necesario desarrollar el "software" de manejadores de dispositivos de entrada/salida, como pueden ser la impresora o el disco duro. Recordemos que las rutinas proporcionadas por el sistema operativo MS-DOS no son reentrantes.

En el siguiente esquema podemos darnos cuenta del alcance de este trabajo (letra en negrillas) y la parte que quedaria por implementar para tener un producto completo que fuera competitivo con sistemas operativos concurrentes.



En este momento los procesos no tiene asignados tiempos para ejecutarse, asi que la conmutación del procesador se hace explicitamente cuando al llamar una primitiva del KERNEL el proceso llamador se suspende o porque "desperto" algun proceso de mayor prioridad, sin embargo es factible implementar una poltica de "Round Robin" modificando la rutina del reloj en el modulo KERNEL.

BIBLIBOGRAFIA

(1) S. Krishnamoorthy, Mukkai; Agnarsson, Snorri. BYTE número 4 vol. 12. Mc Graw-Hill publications. Abril 1987.

(2) Erik Matsson, Sven; A REAL-TIME KERNEL FOR PASCAL. Documento Técnico. Department of Automatic Control, Lund Institute of Technology. Box 725, S-220 07 Lund, Sweeden

(3) S. Tanenbaum, Andrews; OPERATING SYSTEMS : design and implementation. Prentice Hall, Inc. Englewood Cliffs, New Jersey 07632. 1987

(4) Norton, Peter; INSIDE THE IBM-PC : Access to advanced Features and Programming. Prentice Hall, Inc. Robert J. Brady Co. , Browie, Maryland 20715.

(5) Holt, A. Charles; MICROCOMPUTER ORGANIZATION : Hardware and Software. Macmillan Publishing Co. New York, New York 10022.

(6) Turbo Pascal Reference Manual. Borland International Inc.

(7) Technical Reference for the IBM Personal Computer.

APENDICE A

NUCLEO.EXT
NUCLEO.ASM

```
{*****************************************************************************}
{*                            MODULO NUCLEO.EXT                              *}
{*****************************************************************************}
{*****************************************************************************}
{*                          INTERFACE CON NUCLEO.COM                         *}
{*****************************************************************************}

type
   process =^ integer;
procedure nuc; external 'nucleo.com';
procedure initnucleus(ofnuc:integer;var p:process); external nuc[$0a];
procedure initprocess(prog,segstack,ofsstack,ofsheap:integer); external
nuc[$0f];
procedure transfer(var p1:process); external nuc[$14];
procedure iotransfer(numint:integer;var p1:process); external nuc[$19];
procedure disableinterrupts; external nuc[$1e];
procedure enableinterrupts; external nuc[$23];
type registros= record case integer of
         1: (ax,bx,cx,dx,bp,si,di,ds,es,flags : integer);
         2: (al,ah,bl,bh,cl,ch,dl,dh          : byte);
         end;

{*****************************************************************************}
{*     FUNCTION NEWPROCESS                                                   *}
{*      Prog = offset dentro del segmento de código del procedimiento a ser *}
{*             convertido en proceso                                         *}
{*      Size = tamaño del stack que se le creará dado en bytes (debera ser  *}
{*             un numero par)                                                *}
{*****************************************************************************}

Function Newprocess(prog:integer; size:integer):process;
var stack: integer;
begin
getmem(stack,size);
initprocess(prog,seg(stack^),ofs(stack^)+size-6,ofs(stack^));
newprocess := ptr(seg(stack^),ofs(stack^)+size-24);
end;
```

```
;#########################################################################
;#                        MODULO NUCLEO.EXT                              #
;#########################################################################

;##################################################################
;# NUCLEO PARA TURBO PASCAL CONCURRENTE POR URIEL TIRADO RIOS     #
;# Se define un nuevo tipo (processo =  integer)                  #
;# Se crea con la funcion  Newporcess                             #
;# se conmuta con el procedimiento transfer                       #
;# se asocia con una interrupcion con el procedimiento iotransfer #
;##################################################################
Cseg            segment  'cgroup'
                assume cs:cseg
                current       dd       0        ;proceso ejecutándose
                offset nuc    dw       0        ;desplazamiento del nucleo
                apt heap      dw     22h
                heapptr       dw    13ah
    nuc         proc    near
                pushf                           ;salva registro de banderas
                cli                             ;deshabilita interrupciones
                jmp     initnucleus             ;brinca a initnucleus
                pushf                           ;salva registro de banderas
                cli                             ;deshabilita interrupciones
                jmp     initprocess             ;brinca a initprocess
                pushf                           ;salva registro de banderas
                cli                             ;deshabilita interrupciones
                jmp     transfer                ;brinca a transfer
                pushf                           ;salva registro de banderas
                cli                             ;deshabilita interrupciones
                jmp     iotransfer              ;brinca a iotransfer
                jmp     disableinterrupts
                jmp     enableinterrupts
;procedure initnucleus (c+nuc:integer; var c:process); external nuc[06];
    initnucleus:
                push    bp                      ;salva base pointer
                mov     bp,sp                   ;actualiza base pointer
                mov     ax,[bp+10]
                mov     cs:offset nuc,ax        ;saves desplazamiento nucleo
                les     bp,[bp+06]              ;es:bp = proc
                mov     word ptr cs:current+2,es ;current = proc
                mov     word ptr cs:current,bp
                mov     ax,22h
                mov     word ptr cs:aptrheap,ax
                mov     ax,13ah
                mov     word ptr cs:heapptr,ax
                pop     bp                      ;recupera base pointer
                popf                            ;recupera registro de banderas
                ret     6
```

```
;procedure initprocess(proc,seaproc,ofsproc:integer); external nuc[11];
Initprocess:
                push    bp                      ;salva base pointer
                mov     bp,sp                   ;actualiza base pointer
                les     bx,[bp+06]              ;es:bx = variable tipo process
                mov     ax,[bp+10]              ;desplazamiento del procedimiento

                mov     es:[bx],ax              ;es:bx = desplazamiento del
proced.
                mov     ax,[bp+02]              ;ax = registro de banderas
                mov     es:[bx-02],ax           ;llenando stack de proceso
                sub     bx,02                   ;salva csw
                mov     es:[bx-02],bx           ;salva BP
                mov     es:[bx-04],ds           ;salva DS
                mov     cx,[bp+14]
                mov     es:[bx-06],cx
                mov     ax,[bp+12]
                mov     es:[bx-08],ax
                mov     es:[bx-10],cx
                mov     es:[bx-12],ax
                xor     ax,ax
                mov     es:[bx-14],ax           ;posición inicial de cursor
                lea     ax,reschild             ;salva desplazamiento de reschild
                add     ax,cs:offset nuc
                mov     es:[bx-16],ax
                pop     bp                      ;recupera base pointer
                popf                            ;recupera registro de banderas
                ret     6                       ;vacia pila
Reschild:
                mov     dx,0040h                ;
                mov     es,dx                   ;
                mov     bx,0050h                ;
                pop     es:[bx]                 ;recupera posición de cursor
                mov     bx,cs:apt heap
                pop     [bx]                    ;recupera aptpheap
                pop     [bx+2]
                mov     bx,cs:apt heap
                pop     [bx]                    ;recupera heapptr
                pop     [bx+2]
                pop     ds                      ;recupera DS
                pop     bp                      ;recupera BP
                popf                            ;recupera banderas
                pop     bx                      ;desplazamiento de proceso
                sti                             ;habilita interrupciones
                call    bx                      ;llama proceso nuevo
Lazo:           jmp     lazo                    ;nunca regresará aqui
```

```
;procedure transfer(var p1:process);
Transfer:
            push    bp                  ;salva base pointer
            mov     bp,sp               ;actualiza base pointer
            push    ds                  ;salva DS
            mov     bx,cs:apt_heap      ;salva apt_heap
            push    [bx+2]
            push    [bx]
            mov     bx,cs:heapptr       ;salva heapptr
            push    [bx+2]
            push    [bx]
            mov     ax,0040h
            mov     es,ax
            mov     bx,0050h
            push    es:[bx]             ;salva posición del cursor
            lea     ax,res
            add     ax,cs:offset_nuc
            push    ax                  ;salva desplazamiento de res
            lds     bx,cs:current       ;ds:bx = current
            mov     [bx],sp             ;salva sp:ss de proc. a susp.
            mov     [bx+2],ss
;resume p1
            lds     bx,dword ptr[bp+06] ;proceso a reasumir
            mov     ax,[bx+2]           ;ax= sp de proceso a reasumir
            mov     dx,[bx]             ;dx= ip de proceso a reasumir
            mov     word ptr cs:current+2,ds ;actualiza current
             ;actualiza current;             mov     word ptr cs:current,bx
;
            mov     ss,ax               ;nuevo ss:sp (pila)
            mov     sp,dx
            pop     bx
            jmp     bx
Res:
            mov     dx,0040h            ;
            mov     es,dx
            mov     bx,0050h
            pop     es:[bx]             ;recupera posicion de cursor
            mov     bx,cs:apt_heap
            pop     [bx]                ;recupera aptpheap
            pop     [bx+2]
            mov     bx,cs:apt_heap
            pop     [bx]                ;recupera heapptr
            pop     [bx+2]
            pop     ds                  ;recupera DS
            pop     bp                  ;recupera base pointer
            popf                        ;recupera banderas
            ret     4
```

```
;procedure iotransfer(numint:integer;var p1 : process);
Iotransfer:
                push    bp
                mov     bp,sp                   ;actualiza base pointer
                push    ds
                mov     bx,cs:apt_heap          ;salva apt heap
                push    [bx+2]
                push    [bx]
                mov     bx,cs:heapptr           ;salva heapptr
                push    [bx+2]
                push    [bx]
                mov     ax,0040h
                mov     es,ax
                mov     bx,0050h
                push    es:[bx]                 ;salva posición del cursor
                lds     bx,cs:current           ;es:bx = current
                push    ds                      ;salva segmento de current
                push    bx                      ;salva desplazamiento de current
                mov     si,[bp+10]              ;numero de interrupcion
                shl     si,1
                shl     si,1                    ;calculando vector de int.
                xor     ax,ax
                mov     es,ax
                push    es:[si+2]               ;salva contenido del vector
                push    es:[si]                 ;de interrupcion
                push    es                      ;salva direccion de vector
                push    si                      ;de interrupcion
                lea     ax,resdriver
                add     ax,cs:offset_nuc
                push    ax                      ;salva resdriver
                mov     [bx],sp                 ;salva sp de proc. a suspender
                mov     [bx+2],ss               ;salva ss de proc. a suspender
        ;construyendo bloque de brinco
                mov     cx,cs                   ;bx = segmento de código
                xor     ax,ax
                mov     al,ch
                push    ax
                lea     dx,intresume
                add     dx,cs:offset nuc        ;dx = desplazamiento intresume
                mov     ah,cl
                mov     al,dh
                push    ax
                mov     ah,dl
                mov     al,9ah
                push    ax
        ;modificando vector de interrupcion
                mov     es:[si],sp              ;nuevo ss:sp de proceso a
                mov     es:[si+2],ss            ;ejecutar
```

```
        ;resume pl
              lds     bx,dword ptr[bp+04]     ;ds:bx dir. de var. de proceso
              mov     ax,[bx+2]               ;ax = nuevo ss
              mov     dx,[bx]                 ;dx = nuevo sp
              mov     word ptr cs:current+2,ds ;actualizando current con
              mov     word ptr cs:current,bx ;proceso a resumir
              mov     ss,ax
              mov     sp,dx
              pop     bx
              jmp     bx
Resorivor:
              pop     si                      ;recupera dir. vec. modificado
              pop     es                      ;
              pop     es:[si]                 ;recupera vector modificado
              pop     es:[si+2]
              pop     word ptr cs:current     ;recupera current
            pop     word ptr cs:current+2
              pop     ax                      ;recupera posicion de cursor
              mov     dx,0040h
              mov     es,dx
              mov     bx,0050h
              mov     es:[bx],ax
              mov     bx,cs:apt_heap
              pop     [bx]                    ;recupera actheap
              pop     [bx+2]
              mov     bx,cs:apt_heap
              pop     [bx]                    ;recupera heapptr
              pop     [bx+2]
              pop     ds                      ;recupera DS
              pop     bp                      ;recupera BP
              popf                            ;recupera banderas
              ret     6
```

```
;procedure intresume:
Intresume:
                push    ax                      ;salva todos los registros
                push    bx
                push    cx
                push    dx
                push    bp
                mov     bp,sp
                push    si
                push    di
                push    ds
                push    ss
                push    es
                mov     bx,cs:ant_heap          ;salva apt_heap
                push    [bx+2]
                push    [bx]
                mov     bx,cs:heapptr           ;salva heapptr
                push    [bx+2]
                push    [bx]
                mov     ax,0040h
                mov     es,ax
                mov     bx,0050h                ;salva posición de cursor
                push    es:[bx]
                lds     bx,cs:current
                push    ds                      ;salva segmento current
                push    bx                      ;salva desplazamiento current
                lea     ax,resint               ;salva desplazamiento resint
                add     ax,cs:offset_proc
                push    ax
                mov     [bx+2],ss
                mov     [bx],sp                 ;salva sptic de proc. suspend
                mov     ax,[bp+12]              ;cs dejado por call intresume
                mov     dx,[bp+10]              ;ip dejado por call intresume
                inc     dx
                mov     ss,ax                   ;ss:sp para proceso manejador
                mov     sp,dx                   ;de interrupciones.
                cld
                iret
```

```
Resint:
                pop     word ptr cs:current
                pop     word ptr cs:current+2   ;restaura current
                pop     ax                      ;recupera posicion de cursor
                mov     dx,0040h
                mov     es,dx
                mov     bx,0050h
                mov     es:[bx],ax
                mov     bx,cs:apt_heap
                pop     [bx]                    ;recupera aptheap
                pop     [bx+2]
                mov     bx,cs:apt_heap
                pop     [bx]                    ;recupera heapptr
                pop     [bx+2]
                pop     es                      ;recupera registros
                pop     ss
                pop     ds
                pop     di
                pop     si
                pop     bp
                pop     dx
                pop     cx
                pop     bx
                pop     ax
                add     sp,04
                iret                            ;regresa de interruccion.
Disableinterrupts:
                cli                             ;deshabilita interrupciones
                ret
Enableinterrupts:
                sti                             ;habilita interrupciones
                ret
Nuc             endp
Cseg            ends
                end
```

APENDICE B

KERNEL.PAS
IO.PAS

```
{$K-}
const
      maxpriority = 1000;
      tick = 1; sec = 18; min = 1092;
type  unsigned = integer;
      phys_bytes =  byte;
      phys_clicks = unsigned;
      vir_bytes = unsigned;
      vir_clicks = unsigned;
      ept_message = ^ message;
      processref =^processrec;
      semaphore =^semaphorerec;
      event =^eventrec;

      MESS_1 = RECORD              {mensaje usado cor el manejaddor de disco}
          m_device      : byte;
          m_head        : byte;
          m_cylinder    : byte;
          m_sector      : byte;
          m_bytes       : integer;
          m_count       : integer;
          m_address     : phys_bytes;
          m_rec_status  : integer;
          end;

      MESS_2 = RECORD
          m2r1,m2r2     : processref;
          m2i1,m2i3     : integer;
          m2l1,m2l2     : processref;
          m2b1          :  byte;
          end;

      MESS_3 = RECORD                {mensaje usado para texto}
          m3i1,m3i2     : integer;
          m3p1          :  char;
          m3ca1         : string[14];
          end;
```

```
MESSAGE = RECORD
    m_soudes   : processref;  {para quién? o quien envia el mensaje}
    m_type     : integer;  {que clase de mensaje es}
    m_m1       : mess_1;   {mensaje tipo 1}
    m_m2       : mess_2;   {mensaje tipo 2}
    m_m3       : mess_3;   {mensaje tipo 3}
    end;

processrec = record
   suc,pre  : processref;
   proc     : process;
   priority : integer;
   time     : integer;
   proced   : integer;
   mess    : apt_message;
   end;

semaphorerec = record
    counter : integer;
    waiting : processref;
 end;

 eventrec = record
    reentry : semaphore;
    delayed : processref;
    end;
{*****************************************************************************}
{*                  VARIABLES MANEJADAS FOR BIOS                             *}
{*****************************************************************************}
var
  seek_status : byte absolute $0040:$003e;
  motor_status : byte absolute $0040:$003f;
  motor_count : byte absolute $0040:$0040;
  motor_goal : byte;
  diskette_status : byte absolute $0040:$0041;
  nec_status : array [0..?] of byte absolute $0040:$0042;

{*****************************************************************************}
var timer,main,iddle,running,readyqueue,timequeue,
   receivingqueue, sendingqueue,any,procedqueue : processref;
{*****************************************************************************}
procedure ejecuta(proced:integer);
{ Dada la direccion de un procedimiento lo ejecuta }
begin
inline($8B/ $9E/ proced/   {mov    bx,[bp]}
      $FF/ $D3             {call   bx}
       );
end;
{*****************************************************************************}
```

```
procedure put(p,q : processref);
{inserta un record de proceso p antes de un record de proceso q en una
lista de q's}
   begin
   p^.suc := q;
   p^.pre := q^.pre;
   q^.pre^.suc := p;
   q^.pre := p;
   end;
{$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$}
procedure remove (p : processref);
{elimina un record de proceso p de su
lista}
   begin
   with p^ do
      begin
      pre^.suc := suc;
      suc^.pre := pre;
      suc := nil;
      pre := nil;
      end;
   end;
{$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$}
procedure putpriority(p,q : processref);
{inserta un record de proceso p en una cola q de acuerdo a su prioridad}
 var p1 : processref ; pri : integer;
   begin
   pri := p^.priority;
   p1 :=  q^.suc;
   while (p1 <> q) and (pri >= p1^.priority) and (p <> p1) do p1 := p1^.suc;
   if p1 <> p then put(p,p1); {si no existe ya en la lista inserta nodo}
   end;
{$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$}
procedure setpriority(priority : integer); forward;
procedure initsem(var sem : semaphore; initval : integer); forward;
procedure schedule; forward;
{$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$}
{proceso} procedure idleproc;
   begin
   setpriority(maxpriority-1);
   while true do
     begin
     enableinterrupts;
     end;
   end;
{$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$}
```

```
procedure clock :
const clockint = $08;
var p : processref;
   begin
   setpriority(-maxpriority);
   remove(running);
   motor_goal := $0c;
   while true do
     begin
     running := readyqueue^.suc;
     port[$20]:=$20;
     ioTransfer(clockint,running^.proc);
     if motor_count > 0 then
        begin
        motor_count := motor_count - 1;
        if motor_count = 0 then
           begin
           if (motor_goal and $F0) <> (motor_status and $F0) then
              begin
              port[$3F2] := motor_goal;
              motor_status := motor_goal;
              end;
           end;
     end;
     {decrementa tiempo de espera para primer proceso esperando}
     p := timequeue^.suc;
     if p <> timequeue  then p^.time := p^.time - 1;
     {mueve todos los procesos a lo largo de la fila ready }
     while (p^.time = 0) and (p <> timequeue) do
        begin
        remove(p);
        putpriority(p,readyqueue);
        p := timequeue^.suc;
        end;
     {decrementa tiempo de espera para primer procedimiento esperando}
     p := procedqueue^.suc;
     if p <> procedqueue then p^.time := p^.time - 1;
     {ejecuta todos los procedimeintos listos }
     while (p^.time = 0) and (p <> procedqueue) do
        begin
        remove(p);
        ejecuta(p^.proced);
        p := procedqueue^.suc;
        end;
      end;
   end;
{$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$
```

```
procedure schedule ;
   begin
   if readyqueue .suc <> running then
      begin
      running := readyqueue^.suc;
      transfer(running^.proc);
      end;
   end;
{*************************************************************************}
procedure createqueue(var queue : processref);
  begin
  new(queue);
  queue^.suc := queue;
  queue^.pre := queue;
  end;
{*************************************************************************}
{*                                                                       *}
{*          PRIMITIVAS DEL KERNEL PARA PASCAL CONCURRENTE                *}
{*                                                                       *}
{*************************************************************************}
function createprocess(prog , memreq : integer):processref ;
 var  child : processref;
   begin
   disableinterrupts;
   new(child);
   child^.priority := 3; {prioridad de default}
   child^.time := 0;
   child^.mess:= nil;
   putpriority(child,readyqueue);
   child^.proc := newprocess(prog,memreq);
   createprocess:=child;
   schedule;
   if running^.priority > 0 then enableinterrupts;
   end;
{*************************************************************************}

function createproced(proced : integer):processref;

var child : processref;

  begin
  new(child);
  child .time := 0;
  child .proced := proced;
  createproced := child;
  end;
{*************************************************************************}
```

```
procedure initkernel;
 const clockarea = 1024; idlearea = 1024;
   begin
   disableinterrupts;
   any := nil;
   new(running);
   new(readyqueue);
   readyqueue^.suc := running;
   readyqueue^.pre := running;
   running^.suc := readyqueue;
   running^.pre := readyqueue;
   createqueue(timequeue);
   main := running;
   running^.priority := 3;
   running^.mess     := nil;
   createqueue(procedqueue);
   createqueue(receivingqueue);
   createqueue(sendingqueue);
   initnucleus(ofs(nuc),running^.proc);
   timer:=createprocess(ofs(clock),clockarea);
   iddle:=createprocess(ofs(idleproc),idlearea);
   end;
{$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$}
procedure setpriority(priority:integer);
   begin
   disableinterrupts;
   if priority > running^.priority then
      begin
      running^.priority := priority;
      remove(running); {reordena la cola de ready}
      putpriority(running, readyqueue);
      schedule;
      end
   else
      running^.priority := priority;
   if running^.priority > 0 then enableinterrupts;
   end;
{$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$}
procedure initsem(var sem : semaphore; initval : integer);
   begin
   new(sem);
   with sem^ do
      begin
      counter := initval;
      new(waiting); {inicializa cola de semaforo}
      waiting^.suc := waiting; waiting^.pre := waiting;
      end;
   end;
{$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$}
```

```
procedure wait(sem:semaphore) ;
   begin
   disableinterrupts;
   with sem  do
      begin
      if counter > 0 then
         begin
         counter := counter -1 ;
         end
      else
         begin
         remove(running);
         putpriority(running,waiting);
         schedule;
         end;
      end;
   if running^.priority > 0 then enableinterrupts;
   end;
{*******************************************************************************}
procedure signal(sem : semaphore) ;
   var p : processref;
   begin
   disableinterrupts;
   with sem  do
      begin
      if waiting <> waiting^.suc then
         begin {pone el primer proceso en waiting en readyqueue}
         p := waiting^.suc;
         remove(p);
         putpriority(p,readyqueue);
         schedule;
         end
      else
         counter := counter+1
      end;
   if running^.priority > 0 then enableinterrupts;
   end;
{*******************************************************************************}
```

```
procedure signalc(sem : semaphore) ;
   var p : processref;
   begin
   disableinterrupts;
   with sem do
      begin
      if waiting <> waiting^.suc then
         begin {pone el primer proceso en waiting en readyqueue}
         p := waiting^.suc;
         remove(p);
         putpriority(p,readyqueue);
         end
      else
         counter := counter+1
      end;
   if running^.priority > 0 then enableinterrupts;
   end;
{************************************************************************}
procedure initevent(var e : event; sem : semaphore) ;
   begin
   new(e);
   with e^ do
      begin
      reentry := sem;
      new(delayed); {cola de bloqueados vacia}
      delayed^.suc := delayed;  delayed^.pre := delayed;
      end;
   end;
{************************************************************************}
procedure await(e : event) ;
var p : processref;
  begin
  disableinterrupts;
  remove(running);
  put(running, e^.delayed);        {bloquea proceso que llamo await}
  {signal asociada con el semaforo}
  with e^.reentry do               {si hay proceso en semaforo liberalo}
     begin
        if waiting <> waiting^.suc then
           begin
           p := waiting^.suc;
           remove(p);
           putpriority(p, readyqueue);
           end
        else
           counter := counter + 1;
        end;
        schedule;
 if running^.priority > 0 then enableinterrupts;
 end;
{************************************************************************}
```

```
procedure causeie : event) ;
var p : processref;
 begin
 disableinterrupts;
 with e  do
       begin
      {hace que todos los procesos bloqueados esperen al semáforo asocia
       do}
      while delayed <> delayed^.suc do
         begin
         p := delayed^.suc;
         remove(p);
         putpriority(p,readyqueue.waiting);
         end;
      end;
      if running^.priority > 0 then enableinterrupts;
end;
{**************************************************************************}
procedure waitio(typeint : integer; int8259:byte) ;
var driver : processref; shadow,status:byte;
   begin
   disableinterrupts;
   driver := running;
   status := 1;
   {salva apuntador del record de proceso que hizo el llamado}
   remove(running);
   running := readyqueue^.suc; {schedule}
   if (int8259 >= 0) and (int8259 <= 7) then
     begin
     shadow := port[$21];
     port[$21] := shadow and not (status shl int8259);
     end;
   iotransfer(typeint,running^.proc);
   port[$20] := $0b;              {leer ISR de 8259}
   if (port[$20] and (status shl int8259) <> 0) then port[$20] := $20;
   port[$21] := shadow;
   running := driver;
   putpriority(driver,readyqueue);
   schedule;
   if running^.priority > 0 then enableinterrupts;
   end;
{**************************************************************************}
```

```
procedure waittime(t : integer) ;
  var p : processref;
   begin
   disableinterrupts;
   remove(running);
   {encuentra la posición en la cola de tiempo y calcula el tiempo de es
   pera relativo al proceso precedente:
   p := timequeue .suc;
   while (t > p^.time) and (p <> timequeue) do
      begin
      t := t - p^.time;
      p := p^.suc;
      end;
   running^.time := t;
   put(running,p);
   {modifica el tiempo de espera relativo para el siguiente proceso}
   if p <> timequeue then p .time := p .time - t;
   schedule;
   if running^.priority > 0 then enableinterrupts;
   end;
{##########################################################################}

procedure waitproced (p2:processref ;t : integer) ;

  var p : processref;

   begin
   disableinterrupts;
   {encuentra la posición en la cola de tiempo y calcula el tiempo de es
   pera relativo al procedimiento precedente}
   p := procedqueue .suc;
   while (t > p^.time) and (p <> procedqueue) do
      begin
      t := t - p^.time;
      p := p^.suc;
      end;
   p2^.time := t;
   put(p2,p);
   {modifica el tiempo de espera relativo para el siguiente proceso}
   if p <> procedqueue then p^.time := p^.time - t;
   if running^.priority > 0 then enableinterrupts;
   end;

{##########################################################################}
```

```
procedure removeproced (p2:processref);

  var p : processref;

   begin
   disableinterrupts;
   (encuentra el proceso en la cola de tiempo y lo remueve, calcula el tiempo
    de espera relativo al procedimiento precedente)
   p := procedqueue .suc;
   while (p2 <> p) and (p <> procedqueue) do p := p .suc;
   if p = p2 then    (si estaba en la cola)
      begin
      p := p2^.suc;
      removeip2;
      if p <> procedqueue then p^.time := p^.time + p2^.time;
      end;
   if running .priority > 0 then enableinterrupts;
   end;

{##########################################################################}

procedure senditdest:processref;var mes:message);
var p:processref;
begin
  disableinterrupts;
  running^.mess:=addr(mes);          {dirección de mensaje a enviar}
  running^.mess .a_soudes:=dest;     {a quien envio el mensaje?}
  p:=receivingqueue^.suc;
  while (p<>receivingqueue) and (p<>dest) do p := p .suc; {Esperan mi mensaje?}
  if (p^.mess^.a_soudes = running) or
     (p^.mess^.a_soudes = any) then
     begin
     p^.mess  := running^.mess ;         {copia mensaje al buzon del
destinatario}
     p^.mess^.a_soudes := running;       {pone remitente}
     removeip;
     putpriorityip,readyqueue);          {reactiva proceso que esperaba mensaje}
     end
  else
     begin
     removerunning;                      {no existe destinatario}
     putpriorityrunning,sendingqueue);   {espera en cola de envios}
     end;
  schedule;
  if running^.priority > 0 then enableinterrupts;
end;
```

```
{*************************************************************************}

procedure receive(sender:processref; var aes:aessaze);
var p:processref;
begin
  disableinterrupts;
  running^.aess:=addr(aes);              {direccion de buzon a recibir aensaje}
  running^.aess .a_soudes := sender;     {de quien espero el aensaje}
  p := sendingqueue .suc;
  if sender = any then                   {alguien me envio aensaje?}
     while (p <> sendingqueue) and (p .aess .a soudes <> running) do p:= p .suc
  else                                   {esta aensaje que espero?}
     while (p <> sendingqueue) and (p <> sender) do p := p .suc;
  if (p^.aess^.a_soudes = running) then
     begin
     running^.aess :=p^.aess ;           {copia aensaje al buzon del
destinatario}
     running^.aess^.a_soudes := p ;      {de quien recibi el aensaje}
     removep);                           {reactiva proces que envio aensaje}
     putpriority(p,ready/queue);
     end
  else                                   {no me han enviado aensaje}
     begin
     remove(running);
     putpriority(running,receivingqueue);
     end;
  schedule;
  if running^.priority > 0 then enableinterrupts;
end;

{*************************************************************************}

procedure send recieve dst:processref; var aes:aessaze;
begin
send(src_dst,aes);
receive(src_dst,aes );
end;
```

```
{*************************************************************************}

{*************************************************************************}
{*                    MODULO IO.PAS                                       *}
{*   Estas rutinas de salida a display fueron creadas para podor usarlas *}
{*   aun en rutinas de interrupción (las de DOS no son reentrantes )     *}
{*                                                                        *}
{*      CONTENIDO:                                                        *}
{*        putchar - muestra un caracter en display                        *}
{*        putstring- muestra una cadena en display                        *}
{*        putstringln - muestra una cadena en display y avanza linea      *}
{*        putint - muestra un entero en display                           *}
{*************************************************************************}

type   mensaje = string[50];
var column : byte absolute $0040:$0050;
    row    : byte absolute $0040:$0051;

{*************************************************************************}
{*        PROCEDURE SCROLL                                                *}
{*          Realiza el scroll en la pantalla                              *}
{*************************************************************************}

procedure scroll;
var aux : registros;
begin
with aux do
    begin
    ax:=$0800;
    bx:=$0100;
    intr($10,aux);    {lee atributo}
    bh := ah;
    ax:=$0601;
    cx:=$0000;
    dx:=$184f;
    intr($10,aux);
    end;
column:=0;
row := 23;
end;

{*************************************************************************}
{*        PROCEDURE AVANZACURSOR                                          *}
{*          Actualiza posición  del cursor                                *}
{*************************************************************************}
```

```
procedure avanzacursor;
begin
if coluan <= 79 then
   begin
   if row <= 24 then scroll
   else
      row:= row + 1;
   coluan := 0;
   end
else
coluan := coluan + 1;
end;

{*************************************************************************}
{*       PROCEDURE PUTCHAR                                               *}
{*         caracter = caracter a desplegar                               *}
{*************************************************************************}

procedure putchar (caracter : char);
var aux : registros;
begin
  case ord(caracter) of
  $0a: begin
     coluan := 79;
     avanzacursor;
     end;
  $0d: coluan := 0
  else
     with aux do
       begin
       ax := $0a00+ord(caracter);
       bx := $0000;
       cx := $0001;
       intr($10,aux);
       avanzacursor;
       end;
   end;
end;

{*************************************************************************}
{*       PROCEDURE PUTSTRING                                             *}
{*         letrero = letrero a desplegar                                 *}
{*************************************************************************}

procedure putstring(letrero : mensaje);
var i :integer;
begin
for i := 1 to ord(letrero[0]) do putchar(letrero[i]);
end;
```

```
(*****************************************************************************)
(*        PROCEDURE PUTSTRINGLN                                              *)
(*          Letrero = letrero a desplegar                                    *)
(*****************************************************************************)

procedure putstringln(letrero : mensaje);
begin
putstring(letrero);
column := 79;
avanzacursor;
end;

(*****************************************************************************)
(*        PROCEDURE PUTINT                                                   *)
(*          Entero = entero a desplegar                                      *)
(*          Ancho  = numero de digitos para representar al entero            *)
(*****************************************************************************)

procedure putint(entero,ancho:integer);
var indice:byte;
es_neg : boolean;
cadena : mensaje;
num_blancos : byte;
begin
es_neg := false;
if entero < 0 then
   begin
   es_neg := true;
   entero := -entero;
   end;
indice := 30;
if entero = 0 then
  begin
  indice := indice - 1;
  cadena[indice] := '0';
  end;
while entero <> 0 do
  begin
  indice := indice - 1;
  cadena[indice] := char(entero mod 10) + ord ('0'));
  entero := entero div 10;
  end;
if es_neg then
  begin
  indice := indice - 1;
  cadena[indice] := '-';
  end;
num_blancos := ancho - 30 + indice;
while num_blancos > 0 do
begin
putchar(char($20));
num_blancos := num_blancos - 1;
end;
```

```
while indice < 30 do
  begin
  putchar(cadena[indice]);
  indice := indice + 1;
  end;
end;

{*************************************************************************}
{   PROCEDURE STATUS8259                                                  }
{    IMR = mascara del 8259 (0 habilita int, 1 enmascara int)            }
{    IRR = registro de peticiones de interrupcion, 1 = pidiendo interrupcion}
{    ISR = registro de interrupciones siendo atendidas, 1 = sirviendo     }
{*************************************************************************}

procedure status8259;
begin
     putstring(' IMR= ');
     putint(port[$21],6);
     port[$21]:=$0a;
     putstring(' IRR= ');
     putint(port[$20],6);
     port[$20]:=$0b;
     putstring(' ISR= ');
     putint(port[$20],6);
end;
{*************************************************************************}
```

APENDICE C

DISKDRIVE.PAS

```
{*****************************************************************}
{                          GLOBALS.PAS                            }
{ Manejador de disco que nos permite leer disco directamente sin hacer }
{ hacer uso de la INT 13H. Se crea un proceso que recibe mensajes para }
{ realizar operaciones de escritura o lectura en el disco flexible. El }
{ procedimiento WAIT_FOR_INT examina la bandera SEEK_STATUS cada 55 }
{ ticks (2.75 segs),permitiendo la ejecucion de otros procesos en ese }
{ intervalo de tiempo                                             }
{*****************************************************************}


const
  DISKINT       =     13;  {Código para disk interrupt}
  DISK_READ     =     2;   {Codigo para disk}
  DISK_WRITE    =     3;   {Código para disk}
  TASK_REPLY    =    65;
{*****************************************************************}
{ Puertos usados por el controlador de disco                      }
{*****************************************************************}
const
  dor         = $03F2;    {bits de control de motor}
  dsr_status  = $3F4;     {registro de estado del controlador de disco}
  fdc_data    = $3F5;     {registro de datos del controlador de disco}
  dma_addr    = $0004;    {puerto para 16 bits de direccion mas bajos DMA}
  dma_top     = $0081;    {puerto para 4 bits de direccion mas altos DMA}
  dma_count   = $0005;    {puerto para contador de DMA (cuenta = bytes -1)}
  dma_st      = $000C;    {puerto de estado de DMA}
  dma_sl      = $000B;    {puerto de estado de DMA}
  dma_init    = $000A;    {puerto de inicio de DMA}

{*****************************************************************}
{ Registros de estado regresados como resultado de una operacion  }
{*****************************************************************}
  ST0           =   $00;  {registro de estado 0}
  ST1           =   $01;  {registro de estado 1}
  ST2           =   $02;  {registro de estado 2}
  ST3           =   $04;  {registro de estado 3(regresado por DRIVE_SENSE)}
  ST_CYL        =   $03;  {entrada donde el controlador reporta cilindro}
  ST_HEAD       =   $04;  {entrada donde el controlador reporta cabeza}
  ST_SEC        =   $05;  {entrada donde el controlador reporta sector}
  ST_BCM        =   $01;  {entrada donde el controlador reporta cilindro actual}

{*****************************************************************}
{ Campos dentro de los puertos de entrada salida                  }
{*****************************************************************}
  DIRECTION     =   $40;  {esta el FDC tratando de leer o escribir}
  CTL_BUSY      =   $10;  {usado para ver cuando el FDC esta ocupado}
  CTL_ACCEPTING =   $80;  {cuando de bits nos el FDC da cuando esta ocioso}
  MOTOR_MASK    =   $F0;  {Estos bits controlan el registro DOR del motor}
  ENABLE_INT    =   $0C;  {usado para habilitar puerto DOR}
  ST0_BITS      =   $F8;  {checa 5 MSB de seek status}
```

```
  ST3_FAULT     =   $80;   {si este bit es 1, el drive esta danado}
  ST3_WR_PROTECT=   $40;   {1 cuando el diskette esta protegido}
  ST3_READY     =   $20;   {1 cuando el drive esta listo}
  TRANS_ST0     =   $00;   {5 MSB de ST0 para READ/WRITE}
  SEEK_ST0      =   $20;   {5 MSB de ST0 para SEEK}
  BAD_SECTOR    =   $05;   {si este bit es 1, entonces recalibra}
  BAD_CYL       =   $1F;   {si alguno de estos bits es 1, recalibra}
  WRITE_PROTECT =   $02;   {1 si diskette protegido contra escritura}
  CHANGE        =   $10;   {valor retornado por FDC despues de reset}

{*************************************************************************}
{* Bytes de comando para el controlador de disco                         *}
{*************************************************************************}
  FDC_SEEK      =   $0F;   {comanda al drieve para seek}
  FDC_READ      =   $E5;   {comanda al driver para read}
  FDC_WRITE     =   $E5;   {comanda al driver para write}
  FDC_SENSE     =   $08;   {comanda al controler para que diga su status}
  FDC_RECALIBRATE = $07;   {comanda al drive para que pase al cyl. 0}
  FDC_SPECIFY   =   $03;   {comanda al drive para aceptar parametros}

{*************************************************************************}
{* Comandos para el controlador de DMA                                   *}
{*************************************************************************}
  DMA_READ      =   $46;   {codigo de lectura de DMA}
  DMA_WRITE     =   $46;   {codigo de escritura de DMA}

{*************************************************************************}
{* Parametros para el manejador de disco                                 *}
{*************************************************************************}
  BLOCK_SIZE    =  1024;
  SECTOR_SIZE   =   512;   {tamano fisico del sector en bytes}
  NR_SECTORS    =   $09;   {numero de sectors por track}
  NR_HEADS      =   $02;   {dos cabezas (i.e., dos tracks/cylinder}
  GAP           =   $2A;   {tamano de espacio entre sectores}
  DTL           =   $FF;   {determina la longitud del sector}
  SPEC1         =   $DF;   {primer parametro para SPECIFY}
  SPEC2         =   $02;   {segundo parametro para SPECIFY}

{*************************************************************************}
{* Codigos de error                                                      *}
{*************************************************************************}
  ERR_SEEK      =    -1;   {posicionamiento erroneo}
  ERR_TRANSFER  =    -2;   {transferencia erronea}
  ERR_STATUS    =    -3;   {sucedio algo malo al obtener status}
  ERR_RECALIBRATE =  -4;   {recalibrate no trabajo correctamente}
  ERR_WR_PROTECT=    -5;   {diskette protegido contra escritura}
  ERR_DRIVE     =    -6;   {algo malo en un drive}
```

```
{$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$}
{$ Miscelaneos                                                              $}
{$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$}
  MAX ERRORS     =      5;   {# de intentos de reset antes de abandonar}
  MAX_RESULTS    =      8;   {# maximo de bytes regresados por el FDC}
  NR_DRIVES      =      2;   {# maximo de drives}
  MAX_FDC_RETRY  =    100;   {# de intentos de sacar informacion al FDC}
  NR_BLOCKS      =    720;   {# de bloques en un disco de 9 sectores}
  OK             =      1;
  int_flag     = $E0;

{$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$}
{$ Estructura para guardar el estado de cada unidad de disco flexible  $}
{$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$}
TYPE
     floppy = ^floppy drive;
     floppy drive    =   record
        fl opcode    : byte       ;    {DISK READ O DISK WRITE}
        fl curcyl    : byte       ;    {cilindro actual}
        fl_drive     : byte       ;    {drive direccionado}
        fl cylinder  : byte       ;    {cilindro direccionado}
        fl_sector    : byte       ;    {sector direccionado}
        fl_head      : byte       ;    {cabeza direccionada}
        fl count     : integer    ;    {contador de bytes}
        fl address   :  byte      ;    {direccion virtual de usuario}
        fl_results   : array[0..MAX RESULTS] of byte; {el controlador puede dar
                                        errores}
        fl_calibration : boolean  ;    {CALIBRATED or UNCALIBRATED}
        end;

CONST

  UNCALIBRATED  =     false ;       {los drives necesitan ser recalibrados}
  CALIBRATED    =     true  ;       {no se requiere recalibracion}

 VAR

   floppys       : array[0..NR DRIVES]  of floppy_drive;
   prev motor    : byte;          {que motor se prendio al ultimo}
   need reset    : boolean;       {1 cuando hay que resetear controlador}
   initialized   : boolean;       {1 despues de primera transferencia}
   steps_per_cyl: byte;           {# pulses por cilindro al steper motor}
   disk driver : processref;

{$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$}
procedure real address(p:phys_bytes;var p2:phys bytes);
begin
inline($5?/ $C4/ $46/ $02/ $8C/ $C3/ $B1/ $E3/ $F000/ $33/ $d2/ $Ea/ $07/
       $D1/ $EA/ $D1/ $EA/ $D1/ $EA/ $D1/ $EA/ $0D/ $C1/ $81/ $E1/ $0FFF/
       $01/ $E1/ $D1/ $E1/ $01/ $E1/ $01/ $E1/ $F3/ $13/ $C1/ $83/ $D2/
       $00/ $C4/ $7E/ $04/ $26/ $89/ $5?/ $02/ $26/ $89/ $45/ $00/ $8B/
       $EC/);
end;
{$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$}
```

```
function phys_address(pchys_bytes):phys_bytes;
var apt_aux : phys_bytes;
begin
real_address(p,apt_aux);
phys_address := apt_aux;
end;

{=============================================================================}
{=                              FDC  OUT                                     =}
{=============================================================================}
procedure fdc_out(val:byte);
{Saca un byte hacia el controlador de floppy. Esto no es tan trivial, ya que
usted puede escribir unicamente  en el, cuando el decide 'escucharlo'. Si el
controlador se rehusa a escuchar, se manda un reset por hardware al controla-
dor}
var retries,r : integer;

begin
if not need_reset then
  begin
  retries := MAX_FDC_RETRY;
  {pueden necesitarse varios intentos para que el controlador acepte el comando}
  while retries > 0 do
     begin
     r := port[FDC_STATUS];
     r := r and (MASTER OR DIRECTION);     {solo examina bits 2 y 3}
     if r <> CTL_ACCEPTING then retries := retries -1   {FDC no esta escuchando}
     else
        begin
        port[FDC_DATA] := val;
        retries := -1;
        end;
     end;
  if retries > -1 then  need_reset := TRUE;
  end;
end;

{=============================================================================}
{=                              WAIT FOR INT                                 =}
{=============================================================================}

procedure wait_for_int;
begin
if seek_status and $80 = 0 then wait(ioi$0e,b);
SEEK_STATUS := SEEK_STATUS and $7f;
end;
{=============================================================================}
{=                              FDC  RESULTS                                 =}
{=============================================================================}
```

```
function fdc_results(fp:floppy) : integer;
{Extrae resultados del controlador despues de una operacion}
var i,j,status : byte;

{lazo extrayendo bytes del controlador hasta que no exista otro}
begin
i:=0;
while i < MAX_RESULTS do
  begin
  status := port[FDC_STATUS];
  if (status and MASTER = 0) then
     begin
     fdc_results := ERR_STATUS;
     i := MAX_RESULTS+1;
     end
  else
     begin
     status := port[FDC_STATUS];                    {lee de nuevo}
     if (status and DIRECTION) = 0 then
        begin
        fdc_results := ERR_STATUS;
        i := MAX_RESULTS+1;
        end
     else
        begin
        fp^.fl_results[i] := port[FDC_DATA];
        for j :=0 to 5 do;                          {gasta tiempo}
        status := port[FDC_STATUS];
        if (status and CTL_BUSY) = 0 then
           begin
           fdc_results := OK;
           i := MAX_RESULTS+1;
           end;
        end;
     end;
  i := succ(i);
  end; {end while}
if i = MAX_RESULTS then
  begin
  need_reset := true;
  fdc_results := ERR_STATUS;
  end;
end;
{=============================================================================}
{=                              RESETF                                       =}
{=============================================================================}
```

```
procedure reset;
{Manda un comando de reset al controlador. Esto es hecho despues de cualquier
catastrofe como rehusa a contestar}

var i,r,status : byte;
    fp : floppy;
begin
{Deshabilita interrupciones y prueba el bit de reset bajo}
need_reset := FALSE;
disableinterrupts;
motor_status := 0;
motor_goal := 0;
port[DOR] := 0;                         {strobe reset bit low}
port[DOR] := ENABLE_INT;                 {strobe it high again}
enableinterrupts;
wait_for_int;
fp := addr(floppys[0]);                  {usa floppy 0 para prueba}
fp^.fl_results[0] := 0;
fdc_out(FLC_SENSE);
r := fdc_results(fp);
if r <> OK then writeln ('FALLO RESET');
status := fp^.fl_results[0];
if status <> CHANGE then writeln(' FDC NO QUEDA LISTO DESPUES DEL RESET')
else
  begin
  fdc_out(FDC_SPECIFY);
  fdc_out(SPEC1);
  fdc_out(SPEC2);
  for i := 0 to NR_DRIVES - 1 do floppys[i].fl_calibration := UNCALIBRATED;
  end;
end;
```

```
{=============================================================================}
{=                              DMA_SETUP                                    =}
{=============================================================================}
procedure dma_setup(fc : floppy);
{ La PC puede realizar operaciones DMA usando un chip controlador de DMA.
Para usarlo es necesario cargar una direccion de  memoria de 20 bits para
leer o escribir, el numero de bytes a transferir  - 1 , y un codigo de lec-
tura o escritura. Esta rutina habilita el chip  de  DMA. Note que el chip
no es capaz de realizar transferencias mas  alla de 64 kb (por ejemplo no
puede leerse un block de 512 bytes empezando en la direccion fisica 65520)

var mode,low_addr,high_addr,top_addr,low_ct,high_ct:byte;
    user_phys : phys_bytes;
begin
if fp^.fl_opcode = DISK_WRITE then mode := DMA_WRITE
else
  mode := DMA_READ;
user_phys := phys_addresses(fp^.fl_address);
low_addr := lo(ofs(user_phys^));
high_addr := hi(ofs(user_phys^));
top_addr := lo(seg(user_phys^));
low_ct := lo(fp^.fl_count-1);
high_ct := hi(fp^.fl_count-1);
{Ahora inicializa los registros de DMA}
disableinterrupts;
port[DMA_M2] := mode;                   {limpia first_last fli-flop}
port[DMA_M1] := mode;                   {modo de transision sencilla}
port[DMA_TOP] := top_addr;              {4 bits mas altos de direccion}
port[DMA_ADDR] := low_addr;             {8 bits mas bajos de direccion}
port[DMA_ADDR] := high_addr;            {8 bits siguientes de direccion}
port[DMA_COUNT] := low_ct;              {contador de bytes a mover bajo}
port[DMA_COUNT] := high_ct;             {contador de bytes a mover alto}
enableinterrupts;
port[DMA_MSK] := $02;                   {limpia canal 2 de DMA}
end;
```

```
{=========================================================================}
{=                          START_MOTOR                                  =}
{=========================================================================}
procedure start_motor(fp : floppy);
var motor_bit,running:b;ss;
begin
disableinterrupts;
motor_count := 7&sec;                    {5 segundos maximo de tiempo}
motor_bit:= 1 shl (fp .fl_drive+4) ;
motor_goal := motor_bit or ENABLE_INT or fp .fl_drive;
if (motor_status and prev_motor) <> 0 then motor_goal := motor_goal or
prev_motor;
running := motor_status and motor_bit;  {diferente de cero si el motor esta
encendido}
port[$03f2] := motor_goal;
motor_status := motor_goal;
prev_motor := motor_bit;
enableinterrupts;
{Si el motor ya estaba prendido no esperes}
if running = 0 then waittime(5*tick);
end;
```

```
{=====================================================================}
{                          RECALIBRATE                                =}
{=====================================================================}
function recalibrate(fp:floppy):integer;
{ El controlador de floppy no tiene manera de determinar su direccion absoluta
(cilindro). En lugar de eso, el mueve la cabeza un cilindro por paso, llevando
esta cuenta por software. Sin embargo despues de un comando SEEK el hardware
lee la informacion desde el diskette diciendo en que lugar se encuentra en ese
momento. Si la cabeza esta en un lugar erroneo ,se hace una recalibracion for
zando a la cabeza a desplazarse al cilindro 0.}

var r : byte;
begin
start_motor(fp);                    {no se puede recalibrar con un motor apagado}
fdc_out(FDC_RECALIBRATE);           {avisa al controlador que se recalibra }
fdc_out(fp .fl_drive);              {Especifica drive}
if need_reset then recalibrate :=ERR_SEEK
else
  begin
  wait_for_int;
  {checa si la recalibracion se llevo a cabo}
  fdc_out(FDC_SENSE);
  r := fdc_results(fp);             {fuerza a SEEK la siguiente vez}
  fp .fl_cyl := -1;
  if (r <> OK) or ((fp .fl_results[ST0] and ST0_BITS) <> SEEK_ST0) or
     (fp .fl_results[RET_PCN] <> 0) then
     {la recalibracion fallo hay que resetear el FDC}
     begin
     need_reset := true;
     fp .fl_calibration := UNCALIBRATED;
     recalibrate := ERR_RECALIBRATE;
     end
  else
     begin
     {recalibracion exitosa}
     fp .fl_calibration := CALIBRATED;
     recalibrate := OK;
     end;
  end;
end;
```

```
{=============================================================================}
{=                                  SEEK                                     =}
{=============================================================================}
{Posiciona la cabeza del drive especificado en el cylindro deseado}
function seek(fp : floppyp : integer ;
var r :integer;
begin
{ Proporciona un comando de SEEK en el drive indicado a menos que la cabeza
este ya posicionada en el cilindro adecuado}
{ Estamos posicionados en el cilindro correcto?}
with fp  dc
  begin
  if fl_calibration = UNCALIBRATED then
     if recalibrate(fp) <> OK then r := err_seek;
  if (fl_curcyl <> fl_cylinder) and fl_calibration then
     begin
     {cilindro erroneo. Manda un seek y espera por la interrupcion.
     fdc_out(FDC_SEEK);           (Empieza mandando comando seek)
     fdc_out((fl_head shl 2) or fp .fl_drive);
     fdc_out(fl_cylinder * steps_per_cyl);
     if(need_reset) then r := ERR_SEEK
     else
        begin
        wait_for int;
        {se recibio interrupcion, checa estado del drive}
        fdc_out(FDC_SENSE);
        r := fdc_results(fp);
        if(fl_results[ST0] and ST0_BITS) <> SEEK_ST0 then
           r:= ERR_SEEK;
        if(fl_results[ST1] <> fl_cylinder * steps_per_cyl) then
           r:= ERR_SEEK;
        if r <> OK then if recalibrate(fp) <> ok then
           r := ERR_SEEK;
        end;
     end
  else {cabeza ya posicionada}
     r := OK;
  end;
seek := r;
end;
```

```
{=============================================================================}
{=                              TRANSFER                                     =}
{=============================================================================}
{Transfiere bloque de datos de disco a memoria o de memoria a disco}
function transf(fc : floppy) : integer;
{ El drive esta ahora en el cilindro correcto}
var r,s,cp : integer;
{Jamas tratara de transmitir si el drive esta descalibrado o el motor apagado}
Begin
with fc^ do
  begin
  if fl_calibration = UNCALIBRATED then r := ERR_TRANSFER
  else
     begin
     if (motor_status shr (fl_drive + 4)) and 1) = 0 then r := ERR_TRANSFER
     else
        begin
        {El comando es ejecutado enviando 9 bytes al controlador}
        if fl^.oprade = DISK_WRITE then cp := FDC_WRITE
        else
           cp := FDC_READ;
        fdc_out(cp);                    {envia comando de lectura o escritura}
        fdc_out(fl^.head shl 2 or fl^.drive);
        fdc_out(fl^.cylinder^);         {que cilindro}
        fdc_out(fl^.head^);             {que cabeza}
        fdc_out(fl^.sector^);           {que sector}
        fdc_out(2);                     {tamano de sector}
        fdc_out(NR_SECTORS);            {que tan grande es un trak}
        fdc_out(GAP);                   {que tan grande es el GAP}
        fdc_out(DTL);                   {longitud de datos}
        { Bloquearte esperando interrupcion del disco}
        if need_reset then r := ERR_TRANSFER
        else
           begin
           wait_for_int;
           {lee resultados y checa si hay errores}
           r := fdc_results(fp);
           if r <> OK then transf := r
           else
              begin
              if (fl^.results[ST1] and BAD_SECTOR)
              or (fl^.results[ST2] and BAD_CYL) <> 0 then
                 fl^.calibration := UNCALIBRATED;
              if (fl^.results[ST1] and WRITE_PROTECT) <> 0 then
                 begin
                 writeln('EL DISCO EN EL DRIVE ',fl^.drive,' ESTA PROTEGIDO');
                 r := ERR_WR_PROTECT;
                 end;
              if (fl^.results[ST0] and ST0_BITS) <> TRANS_ST0 then
                 r := ERR_TRANSFER
              else
                 begin
                 if (fl^.results[ST1] or fl^.results[ST2]) <> 0 then
                    r := ERR_TRANSFER
```

```
                {compara el numero de sectores transferidos con el esperado}
                else
                  begin
                  s := (fl.results[ST_CYL] - fl.cylinder)*NR_HEADS*NR_SECTORS;
                  s := s + (fl.results[ST_HEAD] - fl.head) * NR_SECTORS;
                  s := s + (fl.results[ST_SEC] - fl.sector);
                  if s*SECTOR_SIZE <> fl.count then r := ERR_TRANSFER
                  else
                    r := OK;
                  end;
                end;
              end;
            end;
          end;
        end;
      end;
    end;
  end;
trans^ := r;
end;
```

```
{=============================================================================}
{=                                do_rdwt                                    =}
{=============================================================================}
function do_rdwt(mess : message) : integer;
{ Lleva a cabo una peticion de lectura o escritura a disco}
var
  r,drive,errors : integer;
  fp :  ^floppy_drive;
{decodifica los parametros del mensaje}
begin
steps_per_cyl := 1;
with mess.m_ml do
  begin
  drive := m_device;
  {si drive valido adelante}
  if (drive >= 0) and (drive < NR_DRIVES) then
     begin
     fp := addr(floppys[drive]);      {fp apunta al record de esta drive.
     with fp^ do
        begin
        fl_drive := drive;            {salva explicitamente el numero de drive}
        fl_opcode := mess.m_type;     {DISK READ o DISK WRITE}
        fl_cylinder := m_cylinder;
        fl_sector := m_sector;
        fl_head := m_head;
        fl_count := m_count;
        fl_address := m_address;       {fl_address = ADDRESS}
        end;
     if fp^.fl_count = BLOCK_SIZE then
        begin
        errors := 0;
        r := 0;
        {Este loop permite que una operacion fallida se repita}
        while (errors <= MAX_ERRORS) and (r <> OK)  do
           begin
           errors := errors + 1;
           if initialized  and (errors = MAX_ERRORS) and (fp^.fl_cylinder > 0)
then
              begin
              if steps_per_cyl > 1 then writeln('diskette no leible')
              else
                 begin
                 steps_per_cyl := steps_per_cyl + 1;
                 errors := 0;
                 end;
              end;
           {Primero checa si se requiere un reset}
           if need_reset then reset;
           {Prepara chip de DMA}
           dma_setup(fp);
           {Ve si el motor esta prendido, si no lo prende y espera}
           start_motor(fp);
           {Si buscamos un nuevo cilindro posisionate en el}
           r := seek(fp);
```

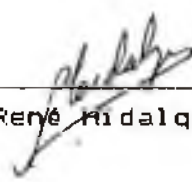
```
            {Realiza la transferencia}
            if r = OK then r:=transf(ip);
            if r = ERR_WR_PROTECT then errors := MAX_ERRORS + 1;
            end {endwhile}
          end {if fl_count}
        else
          do_rdwt := -22; {EINVAL}
        end {if drive}
    else
      do_rdwt := -5; {EIO}
    end; {with}
  if (r = OK) and (ip .fl_cylinder <> 0) then initialized :=true;
  if (r = OK) then do_rdwt := BLOCK_SIZE
  else
    do_rdwt := -5; {EIO}
  end;

{*****************************************************************************}
procedure floppy_task;
var r : integer;
    aes0 : message;
begin
need_reset := true;
while true do
  begin
  receive(any,aes0);
  case  aes0.m_type of
     DISK_READ  : r := do_rdwt(aes0);
     DISK_WRITE : r := do_rdwt(aes0);
     else         r := -22 {EINVAL};
     end;
  aes0.m_type := TASK_REPLY;
  aes0.m_m1.m_rep_status := r;
  send(aes0.m_source,aes0);
  end;
end;
{*****************************************************************************}
function lectrack(device,head,cylinder,sector : integer; buffer : phys_bytes) :
integer;
var aes : message;
begin
with aes.m_m1 do
  begin
  m_device := device;
  m_count := 1024;
  m_sector := sector;
  m_address := buffer;
  m_cylinder := cylinder;
  m_head := head;
  aes.m_type := DISK_READ;
  send_rec(disk_driver,aes);
  lectrack := m_rep_status;
  end;
end;
```

```
{****************************************************************************}
function dscribetrack(device,head,cylinder,sector : integer; buffer : phys_bytes)
: integer;
var mes : message;
begin
with mes.m_m1 do
  begin
  m_device := device;
  m_count := 1024;
  m_sector := sector;
  m_address := buffer;
  m_cylinder := cylinder;
  m_head := head;
  mes.m_type := DISK_WRITE;
  send_rec(disk_driver,mes);
  dscribetrack :=  m_rep_status;
  end;
end;
{****************************************************************************}
procedure initdiskdriver;
begin
disk_driver := createprocess(ord(floppy_task),2048);
end;
```

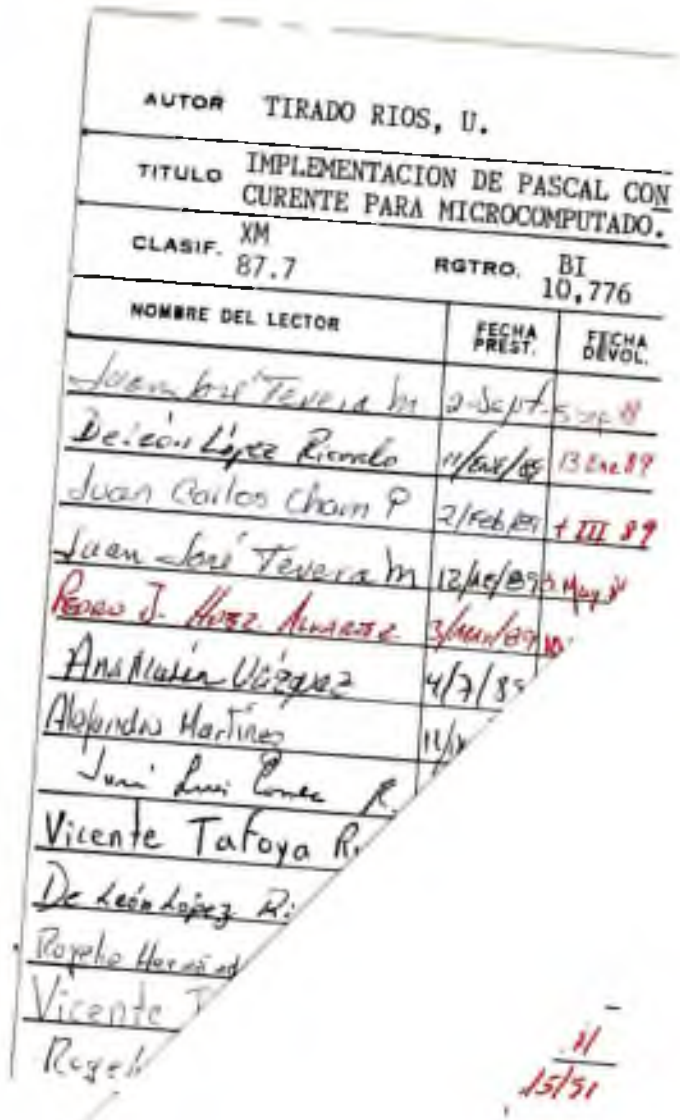
El jurado designado por la Sección de Computación del Departamento de Ingeniería Eléctrica del Centro de Investigación y de Estudios Avanzados del Instituto Politécnico Nacional, aprobó esta tesis el 6 de Noviembre de 1987.

Dr. Manuel E. Guzmán Renteria

Dr. Armando Maldonado Talamantes



Fis. René Hidalgo Salinas.



AUTOR TIRADO RIOS, U.

TITULO IMPLEMENTACION DE PASCAL CON CURENTE PARA MICROCOMPUTADO.

CLASIF. XM 87.7 RGTRO. BI 10,776

NOMBRE DEL LECTOR	FECHA PREST.	FECHA DEVOL.
Juan José Tevera M.	2-Sept-88	5 Sept 88
Delia López Ricardo	11/Ene/89	13 Ene 89
Juan Carlos Chan P.	2/Feb/89	1 III 89
Juan José Tevera M.	12/Mar/89	15 Mar 89
Pedro J. Hdez Alvarez	3/Marzo/89	10
Ana María Vázquez	4/7/89	
Alejandro Martínez	11/	
Juan Luis Cano R.		
Vicente Tafoya R.		
De León López R.		
Rogelio Hernández		
Vicente T		
Roge		

11
15/51