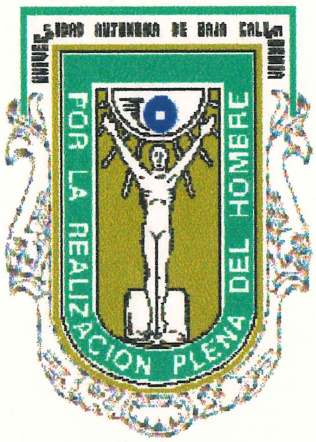
UNIVERSIDAD AUTONOMA DE BAJA CALIFORNIA

FACULTAD DE CIENCIAS



MEDICION DE MOVIMIENTO ARMONICO ASISTIDO POR COMPUTADORA

TESIS QUE PARA OBTENER EL TITULO DE LIC. EN CIENCIAS COMPUTACIONALES PRESENTA:

ATANACIO REYES VALENZUELA

Ensenada B. C. Agosto de 1999

UNIVERSIDAD AUTONOMA DE BAJA CALIFORNIA

FACULTAD DE CIENCIAS

MEDICION DEL MOVIMIENTO ARMONICO ASISTIDO POR COMPUTADORA

T E S I S P R O F E S I O N A L

QUE PRESENTA

ATANACIO REYES VALENZUELA

APROBADO POR:

Jesús R. Lerma A.

M. C. JESUS RAMON LERMA ARAGON

Presidente del jurado

M. C. JOSE IGNACIO ASCENCIO LOPEZ

Secretario

Fis. FRANCISCO JUAREZ GARCIA

1er. Vocal

AGRADECIMIENTOS

En primer lugar a mi director de tesis, Jesús Ramón Lerma Aragón, quien insistió en que este trabajo cumplía con las características para ser presentado como una tesis.

A Victor Arévalo y Luis Mario Lamadrid Moreno: Fueron dos personas que formaron parte de mi conciencia, y como piedra en el zapato estuvieron contínuamente insistiendo y argumentando que no era suficiente completar satisfactoriamente los cursos universitarios para dar por concluida una carrera, y que el título es la única constancia que demuestra nuestro paso por la Universidad.

Francisco Venegas, Javier Siller Leyva y Francisco Juárez García, les agradezco sus constantes recordatorios e insistencia para que cumpliera el trámite de titulación, y que a su vez me sugirieron que trabajara sobre el tema sin pretender que este fuera sobresaliente o único. Ya que en la búsqueda de esos objetivos puede pasar el tiempo en vano, llegando a la frustración y al desánimo.

A todos aquellos colegas y compañeros que me antecedieron en este trámite, gracias por servirme como fuente de inspiración.

También quiero resaltar la colaboración de mis sinodales: Ignacio Ascencio, Francisco Juárez, Jesús Lerma, ya que sus observaciones a esta tésis fueron fundamentales en la realización de la misma.

A mi memoria escapan los nombres de muchas personas que merecen ocupar un lugar en este escrito, a quienes solicito una disculpa y que a la vez agradezco su participación consciente o eventual en la formación de quien esto escribe.

A mis padres, hermanos y familiares quienes en la alborada de mi vida me otorgaron su dirección, consejo y ayuda, y que en etapas subsecuentes tuvieron la suficiente confianza como para otorgarme la libertad de dirigir mi existencia. Gracias a ellos hoy puedo sentir que soy lo que he querido y lo que he podido ser.

RESUMEN de la tesis de ATANACIO REYES VALENZUELA presentada como requisito parcial para la obtención de la Licenciatura en Ciencias Computacionales. Ensenada, Baja California, México. Agosto de 1999.

Medición del movimiento armónico asistido por computadora.

Jesús R. Lerma A

Aprobado por: M. C. JESUS RAMON LERMA ARAGON

En este trabajo se presentan los fundamentos necesarios para el desarrollo de interfaces gráficas de usuario. También se discute lo referente a procesamiento de datos en tiempo real, como lo es la captura de datos por encuesta, captura de datos por interrupción, a la vez de que se muestra el funcionamiento de algunos dispositivos que forman parte de la computadora personal IBM y compatibles, los cuales son necesarios en la implementación de los métodos de captura anteriormente enunciados. Tales dispositivos son: el contador de intervalos de tiempo, el reloj de tiempo real, y el controlador de interrupciones programable.

Se muestra la estructura interna y funcionamiento del puerto paralelo de la computadora con el propósito de que sea considerado como interface de comunicación con el exterior de la computadora.

Por último se detalla el funcionamiento del manejador y analizador de movimiento armónico; aparato que se utiliza para la realización de experimentos relacionados con el movimiento armónico. Se deduce el modelo matemático que describe su funcionamiento y se explica el desarrollo de un programa computacional de captura y procesamiento de datos en tiempo real que fue creado con el propósito de ser usado en conjunción con este aparato.

Contenido

Figuras

Tablas

1. Introducción

La experimentación como parte del método científico juega un papel importante dentro de la investigación y la docencia, por lo que se hace necesario contar con mecanismos, procedimientos e instrumentos que permitan medir cantidades físicas con una precisión confiable y adecuada.

La computadora es una herramienta que puede ser utilizada para la captura de eventos físicos justo en el momento en el que suceden, ya que no sólo captura datos vía teclado sino que también puede capturarlos mediante otros dispositivos periféricos (puerto serie, puerto paralelo, circuitería de adquisición de datos, etc.).

En el trabajo presente se muestra el desarrollo de un programa cuya función es capturar el movimiento producido por dicho generador y analizador de movimiento armónico, medir el periodo, la amplitud, la frecuencia y construír la función de tiempo producida por tal movimiento, en tiempo real y con la precisión necesaria para que los experimentos reflejen resultados útiles.

la importancia del desarrollo de este programa radica en que dota al generador y analizador de movimiento armónico de versatilidad, precisión y atracción hacia la experimentación, además de que los fundamentos teóricos que sirven de marco para la realización del trabajo, también son aplicables para la construcción de otros instrumentos útiles en la medición de magnitudes físicas, tales como aceleración de una masa bajo el estímulo de una fuerza, movimiento de un péndulo, caída libre etc...

Es necesario hacer notar también que debido a los cambios tecnológicos en el área computacional, resulta de vital importancia conocer a fondo los detalles de funcionamiento del generador y analizador de movimiento armónico ya que esto constituye una ventaja que permite realizar actualizaciones y mejoras evitando de esta forma la obsolescencia.

2. Antecedentes

El laboratorio de Física de la Facultad de Ciencias, cuenta con un Oscilador y Analizador de movimiento Armónico. Ese aparato produce oscilaciones que son sensadas por detectores ópticos. De esta forma procesa y obtiene el periodo y la amplitud de cada oscilación. Tales parámetros los presenta en un despliegue numérico.

La persona que esté experimentando con ese aparato, debe estar pendiente a las variaciones que se den tanto del periodo como de la amplitud, todo el tiempo que dure el experimento, con el fin de registrarlas. En algunas ocasiones detectar esas variaciones ocularmente y registrarlas con precisión resulta imposible, ya que además de detectarlas también es necesario registrar el tiempo (en fracciones de segundos) en el cual suceden, lo cual provoca que la realización de experimentos con ese aparato no sea atractiva ni precisa.

Cuando se adquirió el Oscilador y analizador de movimiento armónico, el fabricante de ese aparato ofrecía adicionalmente un paquete de hardware y software para análisis y adquisición de datos. Tal equipo estaba diseñado para funcionar en una computadora Apple II+ ó Apple IIc. Revisando los catálogos y lista de precios de esas fechas se deduce que para completar el equipo (incluyendo la computadora) era necesario desembolsar una suma aproximada de US$5,500. Si ahora se decidiera adquirir el equipo adicional se caería en el infortunio de que todo ese equipo, incluyendo la computadora están fuera del mercado.

Existen otros aparatos similares que hacen las mismas mediciones mediante otros principios, (obviamente más sofisticado) que pueden sustituír el actual, sin embargo el precio es un inconveniente. Por otro lado si el aparato actual se conecta a una computadora, y se desarrolla el

software apropiado, puede lograrse incrementar la versatilidad, precisión y atracción hacia la experimentación, además de que se adquieren los conocimientos necesarios para realizar las actualizaciones necesarias cuando sucedan cambios tecnológicos computacionales.

Los manuales y referencias técnicas del Oscilador y Analizador de movimiento Armónico no sólo muestran la forma de uso, mantenimiento y especificaciones técnicas del aparato, si no que además presentan el diagrama esquemático del circuito eléctrico donde se encuentran los detectores ópticos, y las señales que el aparato produce. Una vez comprendiendo el funcionamiento de ese circuito, se puede conectar a la computadora y procesarse las oscilaciones sensadas por los detectores ópticos de forma adecuada.

3. Fundamentación teórica

3.1. Interfaces gráficas

El propósito fundamental de todo programa computacional radica en hacer más fácil un proceso o en hacer posible un proceso que de otra forma no lo sería. El uso del programa debe ser atractivo, aprender a usarlo no debe representar un problema ya que el objetivo es resolver un problema existente, el programa debe adaptarse a las necesidades del usuario y no el usuario al programa.

Cuando es necesaria la visualización de gráficas, una buena opción que facilita el uso de un programa es el desarrollo de una interface gráfica de usuario, el desarrollo de la misma se facilita si se tienen los conocimientos necesarios sobre el funcionamiento de las tarjetas de video que se van a usar, la representación del color en la computadora, realización y uso de paletas de color y formato de archivos gráficos.

En el presente capítulo se mostrarán de forma resumida algunos de los conceptos necesarios para el desarrollo de interfaces gráficas, que a la vez son útiles para el desarrollo de aplicaciones que requieran despligue de fotografías, tipografía y cualquier tipo de visualización por computadora.

3.1.1. La tarjeta de video EGA y VGA

El estándar EGA/VGA es el factor más importante que ha contribuído al éxito de esas tarjetas gráficas. Este estándar ha contribuído a que los desarrolladores de software puedan alcanzar una gran audiencia. Las tarjetas EGA/VGA incluyen su BIOS propio, siendo este el principal factor que contribuye a la compatibilidad, ya que las rutinas residentes en el BIOS hacen invisible cualquier diferencia en implementaciones de hardware (Ferraro, 1994).

3.1.1.1. Registros de control

Todas las funciones gráficas de la tarjeta de video EGA/VGA son controladas a través de un conjunto de registros. El estándar EGA/VGA dicta el contenido forma de uso y función que desarrolla cada uno de esos registros los cuales son accesados por la computadora por medio de las instrucciones IN, OUT de lenguaje ensamblador (o su equivalente en algún lenguaje de más alto nivel de abstracción).

En la mayoría de los casos, los registros EGA y VGA son idénticos sin embargo el estándar VGA ha agregado muchas características no disponibles en EGA y muchos campos de algunos registros son especificados sólo para VGA, también puede ocurrir que algunos campos tienen una función para EGA y una función diferente para VGA.

Los registros están agrupados en cinco conjuntos básicos los cuales se listan a continuación:

Grupo de registros	Dirección de puerto de E/S (Hex)
Generales o Externos	3BA ó 3DA, 3CA, 3C2, y 3CC
Secuenciadores	3C4, 3C5
Controladores de CRT	3B4, 3B5, ó 3D4, 3D5
Gráficos	3CE, 3CF
Atributos	3C0, 3C1

Tabla III.I.I. Registros y direcciones de puerto para tarjetas EGA/VGA

Note que algunos registros existen en dos direcciones de puerto, esto con el propósito de que tanto el adaptador a color como el monocromo puedan coexistir en el mismo sistema, por ejemplo: las direcciones de puerto que se encuentran en 3Bx se refieren a que el adaptador está emulando un modo monocromo mientras que las direcciónes de puerto que se encuentran en 3Dx indican que el adaptador se encuentra emulando un modo a color.

En muchas aplicaciones es aconsejable usar las rutinas del BIOS para modificar los registros en la tarjeta EGA/VGA. Todas las funciones básicas pueden ser programadas usando el BIOS, sin embargo la velocidad de procesamiento gráfico puede ser incrementada usando algunas características no implementadas en las llamadas del BIOS lo cual a veces hace necesario accesar los registros directamente, en adición, muchas de las características que son programadas en el BIOS requieren un tiempo mayor al ser ejecutadas debido a la generalidad de su código.

EGA y VGA son compatibles a nivel de BIOS. El código escrito para EGA correrá directamente en VGA si está escrito usando sólo llamadas al BIOS, pero si controla los registros directamente no necesariamente se ejecutará correctamente en VGA.

Existen muchos casos en los cuales el programador debe usar las llamadas al BIOS, sobresaliendo aquellos en los cuales la llamada se usa una o dos veces en el programa o cuando los retardos ocasionados por el sobreflujo del BIOS no afectan el desempeño del programa. Usar las funciones del BIOS asegura la compatibilidad, el funcionamiento correcto de la tarjeta de video y del monitor (Ferraro, 1994).

3.1.1.2. Acceso a los registros

Los registros generales, también llamados registros externos, y el registro de selección de color que está dentro del grupo de registros de atributo, pueden ser accesados directamente, es decir

sólo invocando las instrucciones IN, OUT del lenguaje ensamblador. El resto de los registros son accesados de forma indirecta.

La técnica de direccionamiento indirecto usa dos localidades en el puerto de Entrada/Salida para accesar un grupo de registros, las dos localidades son llamadas registro de índice o dirección y registro de dato. Los registros de índice en un grupo tienen un valor de índice que empieza en cero y se incrementa hasta llegar al último registro del grupo. La forma de escribir un dato en cualquier registro de un grupo es escribiendo en el registro de índice el valor del registro deseado (0,1,...n) y en un segundo acceso se escribe el dato en el registro de dato. Por ejemplo: para apagar el cursor debe ponerse en cero el bit 5 del Cursor Start Register (registro 0Ah del grupo de Controladores de CRT 03D4h), para lo cual debe ejecutarse el siguiente código.

```
/* Operación para lectura del Cursor Start Register */
outportb( 0x3D4, 0x0A );      /* 0Ah en el registro indice */
cursor = inportb( 0x3D5 );    /* lectura al registro 0Ah */
cursor &= 0x0DF;

/*Operación para escritura del Registro y apagar el cursor */
outportb( 0x3D4, 0x0A );      /* 0Ah en el registro indice */
outportb( 0x3D5, cursor );    /* escritura al registro 0Ah */
```

3.1.1.3. Memoria de Video

El programador puede leer o escribir directamente en la memoria de video para dibujar líneas y formas sin tener que usar las llamadas al BIOS, lo que permite mover imágenes y formas gráficas entre la memoria de video y la memoria de la computadora.

La memoria de video puede ser organizada en una variedad de resoluciones. Dependiendo del modo gráfico puede puede usarse en formato empaquetado o formato de planos de bits.

3.1.1.4. Resolución del color

El número de colores simultáneos que puede desplegar la tarjeta VGA, depende del número de bits asociados con cada pixel en la memoria de video, de esta forma se determina que 1 bit por pixel despliega dos colores, 2 bits por pixel despliegan 4 colores, 4 bits por pixel despliegan 16 colores y así sucesivamente.

La resolución espacial, en conjunción con el número de bits por pixel, determina el monto de memoria de video necesaria. La memoria de video en VGA contiene un máximo de 256K con los cuales se pueden lograr diferentes modos estándares de resolución del color y resolución espacial. El número de bits por pixel dicta la cantidad de colores simultáneos que pueden desplegarse, esos bits direccionan cada registro de la paleta de color, dicho registro determina el color a desplegarse en el monitor. Cada uno de esos registros está segmentado en tres partes. Cada una de esas partes está asignada a representar la intensidad de cada uno de los tres colores primarios: rojo, verde o azul.

La longitud de cada registro de de la paleta de color determina el máximo de colores posibles: La longitud de los registros en EGA es de 6 bits, por lo que a cada color primario le corresponden 2 bits, mientras que en VGA dicha longitud es de 18 bits por lo que a cada color primario le corresponden 6 bits. El máximo número de colores diferentes que pueden lograrse en EGA es de 64, mientras que en VGA es 256 Kcolores (262,144 colores) (Ferraro, 1994).

Resolución	Bits por pixel	Monto de memoria (bytes)
320 por 200	1	8,000
320 por 200	2	16,000
320 por 200	8	64,000
640 por 200	2	32,000
640 por 350	1	28,000
640 por 350	2	56,000
640 por 350	4	112,000
640 por 480	1	38,400
640 por 480	4	153,600

Tabla III.I.II. Monto de memoria de video necesaria.

3.1.2. Representación del color

Los colores son medidos en una variedad de formas, los artistas han escogido los términos tintes, sombras y tonos para describir colores. Otros términos usados comunmente incluyen, matiz, color y saturación. Los usuarios del color usan la terminología que mejor se acomoda a sus necesidades.

Cuando los artistas agregan negro a los pigmentos para decrementar el brillo, usan el término sombra. Para cambiar la saturación del color, ellos agregan blanco al pigmento, el color resultante es llamado tinte, el color también puede ser modificado agregando ambos: blanco/negro, esto modifica el tono del color. La adición de blanco y/o negro a un pigmento no modifica el matiz del color.

Existen muchos modelos que se usan para representar el color, todos ellos emplean un espacio tridimensional. Es interesante hacer notar que el ojo humano tiene tres diferentes tipos de receptores de color en la retina. Entre los modelos conocidos se encuentran: el CIE, RGB, CMY, YIQ, HSV y HLS, de los cuales sólo describiremos algunos (Ferraro, 1994).

El modelo de color RGB está basado en un sistema cartesiano tridimensional. Los monitores a color y la mayoría de los sistemas gráficos usan este modelo.

El color rojo (Red) está representado en un eje, el azul (Blue) en un segundo eje y el verde (Green) en un tercer eje. El origen del cubo de color es rojo=0, verde=0, azul=0, corresponde al negro (ausencia de color o de luz). Si nos movemos en diagonal partiendo del origen, producimos la escala de grises, el color al final de la diagonal es rojo=máximo, verde=máximo, azul=máximo, corresponde al color blanco.

El modelo de color CMY usa los tres colores substractivos primarios Cyan, Magenta y Amarillo. Este modelo de color es usado en dispositivos que depositan tinta sobre papel blanco, de tal manera que en este modelo la ausencia de color es el blanco y la mezcla de los tres colores substractivos primarios produce negro.

Los modelos de color descritos previamente son efectivos en implementaciones de equipo de procesamiento de color, sin embargo están lejos de describir color en términos humanos.

Dos modelos usados con este propósito son el HSV y el HLS, los cuales usan los términos, matiz, saturación y valor (Hue, Saturation, Value); y matiz, iluminación y saturación (Hue, Lightness, Saturation) respectivamente. Existen también técnicas para convertir los modelos HSV y HLS en el modelo mas usado en computación gráfica, que es el RGB.

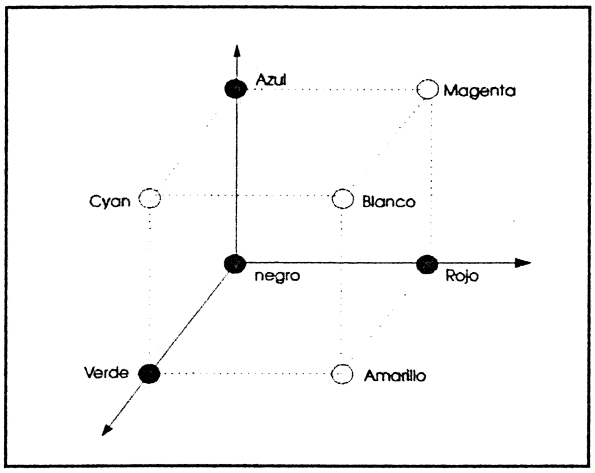


Fig. 3.1.1. El cubo de color.

3.1.3. Archivos en formato PCX

El desarrollo de interfaces gráficas directamente mediante la codificación puede ser un trabajo árduo. Normalmente la satisfacción de las necesidades críticas de un programa representa una mínima parte del mismo, mientras que la atención al usuario y el despliegue de información de una forma agradable, puede representar la mayor parte de los esfuerzos de programación. Este esfuerzo puede ser aligerado si usamos software comercial orientado a diseño gráfico. Con el apoyo de esas herramientas pueden diseñarse las pantallas necesarias para el programa con los colores y formas adecuados, las cuales se graban en archivos con formatos gráficos estándares, los cuales serán cargados por nuestro programa a tiempo de ejecución.

Cualquier modificación en el despliegue gráfico es más fácil y rápida si se realiza mediante software de diseño gráfico en los archivos de pantallas, que si se hace modificando el código del programa.

Los archivos gráficos en formato PCX son ampliamente conocidos y su codificación es relativamente fácil de entender, además de que casi todos los programas de dibujo comerciales pueden leer, y/o crear archivos con este formato, lo cual lo convierte en una alternativa viable para el desarrollo de programas con interfaz gráfica.

3.1.3.1. Paletas de color

Una paleta de color es una tabla de búsqueda que se usa para que un dato en memoria de video asociado con un pixel sea convertido en el color que le corresponde.

Una paleta de color permite al programador cambiar colores rápidamente. Por ejemplo, una imagen en EGA/VGA puede ocupar 256 Kbytes de memoria de video. En modo de 16 colores, sólo pueden desplegarse 16 colores simultáneamente. Si no existiera la paleta de color, sería necesario modificar los datos de los 256 Kbytes de memoria para poder cambiar los colores en el monitor. Con la paleta de color sólo es necesario modificar 16 registros de color.

Otra ventaja de usar la paleta de color es que la cantidad de colores posibles puede ser más grande que el número de colores que pueden ser desplegados simultáneamente. En EGA por ejemplo, pueden seleccionarse 16 colores de entre 64 para desplegarse simultáneamente, sin embargo en VGA, la ventaja es mayor, ya que los colores pueden seleccionarse entre un conjunto de 256 Kcolores.

La paleta de color consiste de 16 registros de paleta. En EGA cada registro es de 6 bits, mientras que en VGA es de 8 bits.

En EGA, Los datos contenidos en los planos de bits, direccionan a los registros de la paleta de color, de tal forma que el registro seleccionado, produce un color en el monitor.

En VGA los datos contenidos en los planos de bits seleccionan registros en de la paleta de color. Cada registro en la paleta de color produce otra dirección, no un código de color. esta dirección sirve para seleccionar uno de los 256 registros de color el cual contiene el código de color que es desplegado en el monitor (Ferraro, 1994).

3.1.3.2. Run-length Encoding

Las imágenes son archivos de bits que ocupan mucho espacio, por esta razón es necesario implementar técnicas de compresión de datos que minimicen tanto el espacio de almacenamiento necesario como tiempo de recuperación y manipulación.

Run-length Encoding es una técnica unidimensional de compresión de imágenes que toma ventaja de los pixeles vecinos horizontales que tienen el mismo valor. Esta técnica de compresión selecciona la orientación horizontal, porque normalmente la memoria de video es orientada horizontalmente.

Run-length Encoding almacena secuencias de valores repetidos. El número de repeticiones de un valor particular se almacena junto con tal valor. Si un grupo está formado de 100 pixeles, todos ellos con valor 10, en vez de almacenar 100 valores consecutivos de 10, con esta técnica es posible almacenar sólo dos valores. Estos valores son la longitud del grupo, el cual es 100, y el valor del pixel el cual es 10.

Si los pixeles adyacentes de una línea horizontal son todos diferentes. esta técnica es desventajosa, ya que en vez de almacenar 100 pixeles con su correspondiente valor, se almacenarán 200 pares (1, valor).

Esta técnica de compresión tiene la característica de que la imagen original una vez comprimida puede ser recuperada sin degradarse (Ferraro, 1994).

3.1.3.3. Codificación de archivos PCX

Los archivos PCX están divididos en tres partes: el encabezado, la imagen y la paleta de color. El encabezado contiene la información necesaria para describir el tipo de archivo en cuestión, la cantidad de colores que contiene, la resolución espacial y de color, las dimensiones de la imagen etc. El encabezado consta de lo siguiente:

fabricante	1 byte.
version	1 byte.
codificacion	1 byte.
bits_por_pixel	1 byte.
xmin,ymin	2 bytes cada uno.
xmax,ymax	2 bytes cada uno.
res_hor	2 bytes.
res_ver	2 bytes.
paleta	48 bytes
reservado	1 byte.
planos_color	1 byte.
bytes_por_linea	2 bytes.
tipo_de_paleta	2 bytes.
relleno	58 bytes.

La paleta que se describe en el encabezado sólo es válida cuando la imagen tiene una resolución de color de 4 bits por pixel, o sea cuando se despliegan 16 colores simultáneos para lo cual sólo se necesita una paleta de 16 colores con registros de paleta de 3 bytes por registro.

Cuando la resolución de color es mayor a 4 bits por pixel la paleta se encuentra en la parte final del archivo. Por ejemplo, si la resolución de color es de 8 bits por pixel, se requiere entonces una paleta de 256 colores, siendo la longitud de cada registro de paleta de 3 bytes, lo que nos da un total de 768 bytes. Entonces en un archivo con esta resolución de color, la paleta estará contenida en los últimos 768 bytes del archivo.

La imagen se encuentra después del encabezado y está codificada con una variación de la técnica Run-Lenght Encoding. Tal variación consiste en lo siguiente:

Se usan 6 bits para representar cada pixel (en vez de 8). cuando los dos bits más significativos (6, 7) están en alto, los 6 bits restantes se usan para indicar la cantidad de repeticiones del siguiente byte. Cuando los dos bits más significativos (6, 7) están en bajo, los 6 bits restantes representan un pixel.

3.2. Procesamiento de datos en tiempo real

Procesar datos en tiempo real, significa interactuar directamente con el ambiente externo de la computadora. Para lograrlo, es necesario que el tiempo de procesamiento sea lo suficientemente rápido para sujetarse a las restricciones del ambiente externo sin que suceda falla alguna.

En aplicaciones de tiempo real, donde es necesario colectar datos, el ambiente dicta la rapidez de la computadora. De igual forma cuando la computadora tiene que transferir datos a dispositivos externos (Motores, válvulas, relevadores etc.), los requerimientos de tales dispositivos al igual que sus características, además del desempeño deseado, deben tomarse en cuenta para determinar la velocidad de procesamiento de la computadora.

Existen sistemas de tiempo real unidireccionales, o sea, aquellos que colectan, o que generan datos, como también existen sistemas bidireccionales, los más típicos de este grupo, son los sistemas de control de lazo cerrado, que contínuamente están sensando el estado del ambiente al que le están transmitiendo señales para controlarlo.

Los sistemas de tiempo real se caracterizan principalmente por tener dispositivos de entrada salida que actúan como sensores del sistema. Las salidas suceden contínuamente al igual que las entradas, y en la mayoría de los casos estos eventos son aleatorios. Usualmente manipulan múltiples entradas y salidas, las cuales en ciertas ocasiones deben atenderse en el mismo intervalo de tiempo.

Las restricciones de tiempo para los sistemas de tiempo real normalmente son muy exigentes por lo que debe tomarse en cuenta la velocidad de la computadora, el sistema operativo, y en ocasiones no se usa sistema operativo sino un programa dedicado grabado en ROM que levanta al sistema al encendido y da atención al ambiente.

Si se considera usar un sistema operativo conocido en aplicaciones de tiempo real, deben preferirse aquellos que no son multitareas (DR-DOS, MS-DOS etc.), con el propósito de que todo o la mayoría del tiempo de procesamiento se lo dedique a la aplicación de tiempo real. En caso de que las condiciones o los requerimientos de la aplicación exijan el uso de un sistema operativo multitareas, debe elegirse un sistema preemtive, en lugar de un sistema cooperativo, y deben estudiarse las políticas empleadas por el calendarizador del sistema operativo con el fin de que la aplicación de tiempo real interrumpa cualquier otro proceso y se apropie del procesador en el momento que así lo requiera. Debe considerarse también la latencia de las interrupciones y del calendarizador del sistema operativo con el fin de que esta sea menor a las exigencias de tiempo de la aplicación de tiempo real en cuestión. Cualquier edición del sistema UNIX cuenta con las características descritas.

Entre las principales aplicaciones de tiempo real se encuentran:

- Control de procesos: laboratorios, equipos industriales, procesos que incluyen reacciones químicas, control de herramientas.
- Telecomunicaciones: commutadores, redes de computadoras, muxs, modems.
- Instrumentación: Instrumentación médica, multímetros, osciloscopios, equipos de medición.
- Productos de consumo: Hornos de microondas, máquinas de lavado, televisores, juguetes, copiadoras, automóviles.
- Periféricos: Terminales, impresoras, discos, drives.

3.2.1. Captura de datos por encuesta (Polling)

Este es el método más simple de captura de datos en tiempo real. Consiste en encuestar periódicamente cada puerto de entrada con el fin de capturar datos, o para sensar el estado de algún dispositivo externo.

La implementación del método puede llevarse a cabo de dos formas: mediante un ciclo en el cual se muestrean los eventos o puertos, o haciendo uso de la interrupción del contador de intervalos de tiempo. Cuando se realiza la captura de datos por polling mediante un ciclo de muestreo, el tiempo en el cual se consulta cada puerto o evento no es controlado, dicho tiempo de muestreo depende de la velocidad de la computadora, y de la cantidad de dispositivos a encuestar. Si se necesita muestrear los eventos, o puertos a intervalos de tiempo iguales o conocidos, entonces puede usarse la interrupción que genera el contador de intervalos de tiempo. Para hacer lo anterior, es necesario insertar en el vector de interrupción del contador intervalos de tiempo, la dirección de la rutina de muestreo, y una vez que la rutina termine, debe ejecutarse la rutina original del contador. Esta forma de implementación tiene el inconveniente de que dificulta la depuración del programa debido a que el flujo del mismo se complica.

Las principales desventajas del método de captura de datos por Polling son:

- El microprocesador debe estar dedicado siempre a encuestar todos los puertos o eventos.
- Cuando la cantidad de dispositivos a encuestar es grande, el ciclo de espera también es grande.
- Se le da la misma atención a puertos con mucha actividad y a los puertos inactivos o poco activos, por lo que se afecta la eficiencia del programa.
- Si suceden entradas a determinado puerto, cuando el microprocesador está ocupado o encuestando otro puerto, dicho dato puede perderse.

La facilidad de implementación de este método lo convierten en buen candidato de uso, sobre todo cuando la exactitud de la captura de los datos no es crítica, o cuando las entradas suceden a velocidades tan altas, que no es posible capturarlas sin pérdida. En este caso los datos perdidos pueden estimarse mediante métodos estadísticos, o por interpolación.

3.2.1.1. Contador de intervalos de tiempo

Las computadoras personales IBM y compatibles incluyen entre su hardware un contador de intervalos tiempo que interrumpe al microprocesador cada vez que se cumple un intervalo de tiempo previamente programado en dicho contador.

Este contador, conocido comúnmente como timer se encuentra en la dirección 40 hex, pero se accesa mediante un registro de control que se encuentra en la dirección 43 hex (Foster et al, 1998), (NEC, 1987) El número que debe escribirse en este registro tiene el significado siguiente.

Bits 7, 6	Selecciona el contador, 0, 1, 2.
Bits 5, 4	Latch, LSB, MSB, LSB-MSB.
Bits 3, 2, 1	Modo
0	Interrupt on count.
1	One-shot
2	Rate generator
3	Square Wave generator
4	Triggered strobe (software)

5 Triggered strobe (hardware)

Bit 0 Conteo en Binario/BCD.

El contador cero es el contador de intervalos de tiempo, se encuentra en la dirección 40h. El contador uno es el de refresco de memoria, se encuentra en la dirección 41h, y el contador dos es el oscilador de la bocina, se encuentra en la dirección 42h, y se usa para generar sonidos que pueden ser programados a la frecuencia deseada. El contador que estamos analizando es el cero y debe seleccionarse y configurarse como LSB-MSB, en modo Square Wave generator y en conteo binario.

Debido a que el contador es de 16 bits y que el registro 40 hex por donde se le escribe o se le lee es de 8 bits, se necesitan dos accesos para poder escribir o leer el contenido completo del contador. De acuerdo a como está configurado, el primer acceso lee o escribe el byte menos significativo (LSB) y el segundo acceso lee o escribe el byte más significativo (MSB).

Para determinar o definir cada intervalo se debe tomar en cuenta lo siguiente:

1. Existe un generador de pulsos (oscilador) conectado al contador que gobierna cada cuenta. Este oscilador genera pulsos con una frecuencia de 1.19318Mhz.

2. Para programar al contador se le debe escribir un número de 16 bits. Cuando no se le escribe toma por omisión el 0FFFF hex. Tal número se decrementa en uno por cada pulso recibido del oscilador.

3. Cuando el contador llega a cero, interrumpe al microprocesador a través del vector IRQ0, y se reinicializa con el número previamente programado, para empezar nuevamente el conteo en forma regresiva.

Las rutinas del BIOS referentes a tiempo, asumen que el contador está programado con el valor de omisión (0FFFh). Tomando en cuenta este número y la frecuencia del oscilador. El tiempo

que le toma al contador llegar a cero será de 54.924 milisegundos. Si dividimos 1000/54.924 obtenemos que el contador interrumpe al microprocesador 18.2 veces por segundo.

3.2.1.2. Reloj de tiempo real

Además del contador de intervalos de tiempo, existe dentro de la computadora un reloj de tiempo real el cual provee las funciones siguientes:

- Medición de minutos segundos y horas de un día.
- Mide días de la semana, fecha, mes y año.
- Reconocimiento automático de final de mes.
- Actualización automática para horario de verano (DayLight Saving Time).
- Provee 64 bytes de RAM (14 para medición de tiempo y 50 de propósito general).
- Generador de de onda cuadrada.
- Genera tres tipos de interrupciones:
    - Alarma. Desde una vez por segundo hasta una vez por día.
    - Contador de intervalos. Desde cada 30.517 microsegundos hasta 500 milisegundos.
    - Fin de ciclo de actualización de reloj.

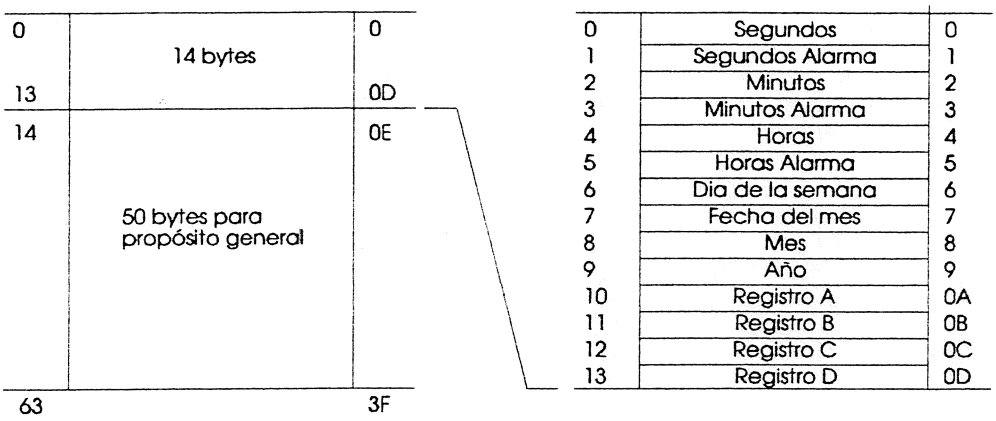


Fig. 3.2.1. Mapa de memoria del reloj de tiempo real.

Tal como se observa en la figura 3.2.1. La memoria proveida por el reloj de tiempo real consiste de 50 bytes de propósito general mas 10 localidades que normalmente cuentan tiempo, calendario y alarma y 4 bytes que sirven como registros de control y estado. Todos los 64 bytes son de lectura y escritura excepto por lo siguiente (Motorola, 1983):

1) Los registros C y D son de lectura solamente.

2) El bit 7 del registro A es de lectura solamente.

3) El bit de más alto orden del contador de segundos es de lectura solamente.

Los tres bytes dedicados a la alarma pueden ser usados en dos formas. si se inserta un valor en las localidades apropiadas de hora, minutos y segundos, y además se habilita la interrupción de la alarma, sucederá una interrupción de alarma cada intervalo de tiempo especificado. La segunda forma de uso, es colocar un código "don't care" en uno o más de los bytes de la alarma. Un código "don't care" es cualquier byte hexadecimal entre C0 y FF (1 en cualquiera de los dos bits más significativos). Si un código "don't care" se pone en el byte de horas, la interrupción de alarma sucedará cada hora. Similarmente si se coloca un código "don't care" en el byte de minutos sucederá

una interrupción de alarma cada minuto. Si los tres bytes de alarma tienen código "don't care", sucederá una interrupción cada segundo.

Los 50 bytes de propósito general los usa el BIOS para almacenar información relacionada con la configuración de la computadora y periféricos: Puertos serie, Puertos paralelos, tipo de disco duro, tipo y capacidad de los drives para floppy, RAM instalada etc.

Otra de las características de este reloj de tiempo real es que puede generar una frecuencia de salida de onda cuadrada la cual puede servir como sintetizador de frecuencia o para generar tonos de audio bajo el control de un programa, esta función puede habilitarse o deshabilitarse con el bit 3 del registro B (SQWE).

Además de servir como reloj este disposivo es capaz de generar interrupciones a intervalos de tiempo programados en el registro A mediante los bits 0-3. Las interrupciones a intervalos de tiempo conocido pueden ser usadas en sistemas de tiempo real como lo son: Sistemas de multitareas, generar intervalos de muestreo iguales, crear intervalos de salida etc. Esta interrupción puede habilitarse o deshabilitarse mediante el bit 6 (PIE) del registro B. La tabla III.II.I. muestra como se programa el periodo de las interrupciones, así como la frecuencia de salida de onda cuadrada. Note que la onda cuadrada de salida y el periodo de la interrupción son iguales, y que por cada incremento en los bits de selección del registro A el periodo de la interrupción se duplica.

BITS 0-3 DEL REGISTRO A				BASE DE TIEMPO DE 4.194304 ó 1.048576 MHz	
RS3	RS2	RS1	RS0	INT.PERIODICA	FREC. DE SALIDA
0	0	0	0	Nada	Nada
0	0	0	1	30.517 µs	3.90625 KHz
0	0	1	0	61.035 µs	16.384 KHz
0	0	1	1	122.070 µs	8.192 KHz
0	1	0	0	244.141 µs	4.096 KHz
0	1	0	1	488.281 µs	2.048 KHz
0	1	1	0	976.562 µs	1.024 KHz
0	1	1	1	1.953125 ms	512 Hz
1	0	0	0	3.90625 ms	256 Hz
1	0	0	1	7.8125 ms	128 Hz
1	0	1	0	15.625 ms	64 Hz
1	0	1	1	31.25 ms	32 Hz
1	1	0	0	62.5 ms	16 Hz
1	1	0	1	125 ms	8 Hz
1	1	1	0	250 ms	4 Hz
1	1	1	1	500 ms	2 Hz

Tabla III.II.I. Selección de frecuencia y periodo para salida de onda cuadrada e interrupción periódica.

Existe un intervalo de tiempo en el cual el reloj de tiempo real actualiza los segundos, verifica sobreflujo para incrementar los minutos y de la misma forma procede con las horas, las semanas, los meses y los años. Este intervalo además verifica si se cumplen las condiciones para generar una interrupción de alarma. A este lapso de tiempo se le llama ciclo de actualización, cualquier acceso a los bytes de tiempo y alarma durante este ciclo es bloqueado con el fin de proteger al usuario de leer datos transitorios. durante este ciclo el bit 7 del registro A (UIP-Update In Progress) se encuentra en 1.

Debe evitarse leer o actualizar los registros de tiempo y alarma durante el ciclo de actualización, lo cual puede lograrse de dos formas:

La primera consiste en usar la interrupción de fin de ciclo de actualización de reloj. Si es habilitada, provoca que suceda una interrupción después de cada ciclo, lo cual indica que durante los 999 ms siguientes a la interrupción puede leerse información válida.

La segunda forma consiste en usar el bit 7 del registro A (UIP) para verificar si existe una actualización en progreso o no. Después de que el bit UIP pasa a alto, el ciclo de actualización inicia 244 µs después.

Los registros de control y estado del reloj de tiempo real se explican a continuación:

Registro A:

bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0
UIP	DV2	DV1	DV0	RS3	RS2	RS1	RS0

UIP: (Update in progress). Es una bandera que puede ser monitoreada por el programa para verificar si existe la condición de ciclo de actualización . Cuando este bit está en 1 el ciclo de actualización está en progreso, y los registros de tiempo y alarma no están disponibles. Cuando está en 0 los registros de tiempo y alarma pueden ser accedidos.

DV2, DV1, DV0: Estos tres bits se usan para seleccionar cual de las tres frecuencias van a ser usadas como base de tiempo (4.194304 Mhz, 1.048576 Mhz y 32.768 Khz).

RS3, RS2, RS1, RS0: Estos cuatro bits se usan para seleccionar la frecuencia de salida de onda cuadrada y el periodo de la interrupción generadora de intervalos de tiempo (ver tabla III.II.I).

Registro B:

bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0
SET	PIE	AIE	UIE	SQWE	DM	24/12	DSE

SET: Cuando este bit está en "0", el ciclo de actualización funciona normalmente, de lo contrario cuando está en "1" cualquier ciclo de actualización en progreso es abortado y el programa puede inicializar los registros de tiempo y alarma sin ser interrumpido.

PIE: (Periodic Interrupt Enable). Este bit habilita la interrupción periódica. Un programa debe poner este bit en "1" para recibir interrupciones con un periodo programado en los bits RS3, RS2, RS1, RS0 del registro A.

AIE: (Alarm Interrupt Enable). Este bit debe estar en "1" si se desea permitir que sucedan interrupciones de alarma.

UIE: (Update-ended Interrupt Enable). si este bit está en "1" el reloj de tiempo real generará una interrupción después de finalizar cada ciclo de actualización, para indicar que durante los próximos 999ms puede leerse información de tiempo válida.

SQWE: (Square-Wave Enable). Este bit habilita la salida de onda cuadrada a la frecuencia programada en los bits RS3, RS2, RS1, RS0 del registro A.

DM: (Data Mode). Indica si los datos de tiempo y alarma están en formato BCD o binario. un "1" en este bit indica datos en binario, mientras que "0" significa datos en BCD.

24/12: Establace el formato de los bytes de hora. "1" indica formato de 24 horas mientras "0" significa formato de 12 horas.

DSE: (Daylight Saving Enable). Bit de horario de verano. Este bit permite al programa dos actualizaciones especiales durante el año. si este bit se programa a "1", el último Domingo de Abril el reloj se se incrementará automáticamente desde la 1:59:59 AM a las 3:00:00 AM, de igual forma el último Domingo de Octubre cuando el reloj marca la 1:59:59 AM, el siguiente cambio será 1:00:00 AM. Estas actualizaciones especiales no ocurrirán cuando el bit DSE está en "0".

Registro C:

bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0
IRQF	PF	AF	UF	0	0	0	0

IRQF: (Interrupt Request Flag). Es una bandera que se pone en "1" cuando sucede alguna interrupción que esté habilitada. Esta bandera cambia a "0" una vez que el registro C ha sido leido.

PF: (Periodic Interrupt Flag). Esta bandera se pone en "1" cuando se genera la interrupción periódica, independientemente de que si está habilitada o no en el bit PIE del registro B. Esta bandera cambia al estado "0" cuando el registro C es leido.

AF: (Alarm Interrupt Flag). Un "1" en este bit indica que el tiempo actual concuerda con el programado en los bytes de alarma. Si la interrupción de alarma (AIE) está habilitado se generará una interrupción. Al leer el registro C este bit cambia a "0".

UF: (Update-ended Interrupt Flag). Esta bandera se pone en "1" cada vez que finaliza un ciclo de actualización. Cambia a "0" cuando se lee el registro C.

Los bits 3 al 0 no se usan, son sólo de lectura y no pueden ser modificados.

Registro D:

De este registro sólo se usa el bit 7 (VRT), el cual es una bandera que indica si los datos de tiempo, alarma y RAM son válidos. Los valores de tiempo, alarma y RAM, pueden ser incorrectos cuando el reloj de tiempo real sensa voltaje insuficiente o sin voltaje. Esto puede suceder cuando la pila que alimenta al reloj de tiempo real se baja o deja de funcionar.

3.2.2. Captura de datos por Interrupción

Este método es el más recomendado cuando no se desea perder ningún dato de entrada y se desea exactitud en la captura. Otra causa por la cual se prefiere usar este método en vez de polling, se da cuando no es posible fijar la frecuencia con la cual deben muestrearse los eventos externos, o que fijar tal frecuencia vuelve ineficiente al programa debido que los eventos suceden a tiempos aleatorios.

El método consiste en que el dispositivo encargado de recibir los datos genere una señal eléctrica que interrumpa el flujo del programa que el microprocesador está ejecutando, para que este atienda la entrada. Una vez que el microprocesador ha dado servicio a la captura de los datos, continúa con la tarea que estaba en ejecución antes de ser interrumpido.

Este mecanismo permite que los eventos sean atendidos en el momento en que suceden y no se pierda tiempo en estar periódicamente verificando la existencia o no exeistencia de tal o cual evento.

Antes de entrar en detalles es preciso mencionar que el flujo de un programa puede ser interrumpido por tres razones:

- por una interrupción invocada en el mismo programa, a la cual se le denomina interrupción por software.
- Por una interrupción invocada desde un dispositivo externo al procesador o la memoria RAM, a la cual se le llama interrupción por hardware.
- Por una llamada a una subrutina la cual es invocada dentro del mismo programa.

En el último de los casos descritos anteriormente, el código ejecutado por la llamada se le llama código reentrante debido a que se encuentra dentro del mismo programa. (el procesador sale de ejecutar instrucciones del programa y entra a ejecutar instrucciones del mismo programa). En los primeros dos casos el código ejecutado después de que el programa sea interrumpido se llama código no reentrante debido a que este generalmente se encuentra fuera del programa (Intel, 1992).

En este trabajo nos referiremos sólo a las interrupciones por software y por hardware.

3.2.2.1. Interrupciones por software

Antes de definir una interrupción por software iniciaremos por decir que cuando el microprocesador está corriendo en modo real, los primeros 1024 bytes de la memoria RAM contienen direcciones que apuntan a rutinas que dan servicio a interrupciones. O sea, que cuando sucede una interrupción, se ejecuta alguna de las rutinas apuntadas por estas direcciones. Cada dirección está formada por cuatro bytes (dos palabras), dos de estos bytes indican el segmento y los otros dos indican el desplazamiento (offset) dentro del mismo segmento donde se encuentra la rutina. Un simple cálculo aritmético nos permite descubrir estos 1024 bytes forman 256 direcciones que apuntan a igual cantidad de rutinas (Interrupt Service Rutine).

Cada una de estas 256 direcciones puede ser referenciada por un número de 8 bits al cual se le llama vector de interrupción.

Es preciso mencionar también que si el procesador corre en modo protegido cada uno de los 256 vectores de interrupción no hacen referencia a los primeros 1024 bytes de memoria sino que en vez de esto, dicho vector selecciona un descriptor de interrupción contenido en una tabla de descriptores de interrupciones la cual es inicializada y mantenida por el sistema operativo que corra en modo protegido. Un descriptor de interrupción es capaz de direccionar hasta 4 Gbytes de memoria, además de proveer al procesador de información necesaria para otorgar privilegios de acceso e instrucciones permitidas en dicha región de memoria, mientras que una dirección en modo real sólo tiene acceso a 1 Mbyte de memoria y no otorga privilegios especiales de acceso ni prohibe la ejecución de tales o cuales instrucciones. La figura 3.2.2. muestra un descriptor de interrupción y una dirección referenciada en modo real.

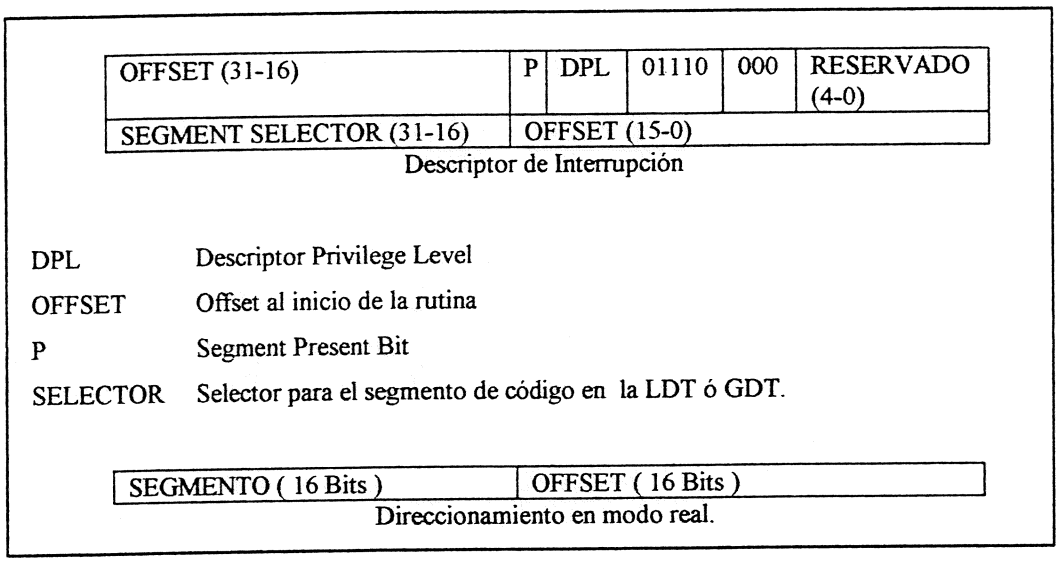


Fig. 3.2.2. Descriptor de interrupción y direccionamiento en modo real.

En este trabajo no trataremos la forma del manejo de interrupciones en modo protegido, debido a que el tema es demasiado amplio, además de que está relacionado con la administración de la memoria, lo cual lo hace dependiente del sistema operativo en el cual se esté trabajando. Se hace mención del modo protegido nada más como referencia y para mostrar algunas de las diferencias existentes con el modo real. También se considera conveniente agregar que los procesadores Intel capaces de funcionar en modo protegido de 32 bits son i386, i486 y Pentium, mismos que también pueden funcionar en modo real para compatibilidad con los procesadores anteriores.

Existe en los procesadores Intel la instrucción:

INT *n*, (En Exadecimal CD *n*. En binario 11001101*n*)

que provoca que el procesador interrumpa el flujo de programa y ejecute una rutina direccionada por el vector de interrupción (*n* representa al vector de interrupción).

Cuando el apuntador de programa (IP) encuentra la instrucción INT, salvaguarda sus banderas, algunos segmentos como los son CS, IP etc. y busca en los primeros 1024 bytes de memoria la dirección indicada por el vector de interrupción (multiplica *n* por 4), carga los valores encontrados en CS:IP, para ejecutar la rutina apuntada por esa dirección (Intel, 1996). La rutina (ISR) debe responsabilizarse de salvaguardar en pila los registros que use durante su ejecución y una vez que termine de ejecutarse debe sacarlos de la pila para devolverlos a su valor original y ejecutar la instrucción IRET (Interrupt Return) la cual causa que los valores que salvaguardó automáticamente el procesador en la pila sean devueltos a CS:IP y de esta forma el programa regrese a ejecutar las instrucciones subsecuentes que estaba ejecutando antes de ser interrumpido.

3.2.2.2. Interrupciones por hardware

Los dispositivos externos al procesador generalmente son más lentos que él, de lo contrario el poder atenderlos sería imposible, por lo que mientras se espera el arribo de algún dato, lo ideal es que el procesador aproveche el tiempo haciendo algo, en vez de estar contínuamente verificando el arribo o no arribo de datos. Además de que existe la posibilidad de que los datos lleguen en el tiempo en el cual el procesador no esté verificando la entrada, lo que provoca que el dato se pierda.

Para ejemplificar lo anterior consideremos un programa que atienda el teclado. supongamos que cada cierto tiempo el programa verifica el puerto conectado al teclado para ver si el usuario ha presionado alguna tecla, puede darse el caso de que por cada 100 verificaciones una de ellas resulte en éxito, o puede que ninguna, lo que resulta una pérdida de tiempo. Puede suceder también que después de que el programa revisó el puerto de entrada el usuario presionó la tecla "A" y después, antes de la siguiente revisión, el usuario presionó la tecla "T" la cual se sobreescribe en el mismo puerto de entrada, cuando el programa realiza la siguiente revisión encuentra la letra "T" la cual captura en su buffer interno y la letra "A" se habrá perdido.

Para evitar lo anterior, el procesador tiene una línea llamada INT (No confundir con la instrucción INT *n*) por la cual puede ser interrumpido. Es por medio de esta línea como un dato le dice al procesador "ya estoy aquí", corresponde ahora a un programa ejecutado por el procesador la responsabilidad de atender o no a esa petición. Una vez que la linea INT cambia de nivel (0 a 1), el procesador contesta al dispositivo por otra línea llamada INTA (Interrupt Acknowledge) para indicarle que su petición ha sido recibida.

Como puede verse, debido a que sólo existe una línea INT sólo un dispositivo puede interrumpir al procesador -obviamente el dispositivo que tenga la suerte de haber sido conectado en esta línea. Para evitar esta restricción existe el Controlador de interrupciones programable (PIC por sus iniciales en inglés). Este dispositivo se conecta a la línea INT y a él pueden conectarse 8 dispositivos adicionales.

3.2.2.3. El controlador de interrupciones (8259)

Las computadoras Personales conocidas como PC normalmente están equipadas con dos controladores de interrupciones a los cuales pueden ser conectados 8 dispositivos por controlador, con lo cual pueden controlar interrupciones provenientes de 15 dispositivos diferentes (esto porque uno de los dispositivos conectados al primer PIC es el segundo PIC). La figura 3.2.3. muestra como están conectados estos dispositivos al procesador (Tanenbaum et al, 1997).

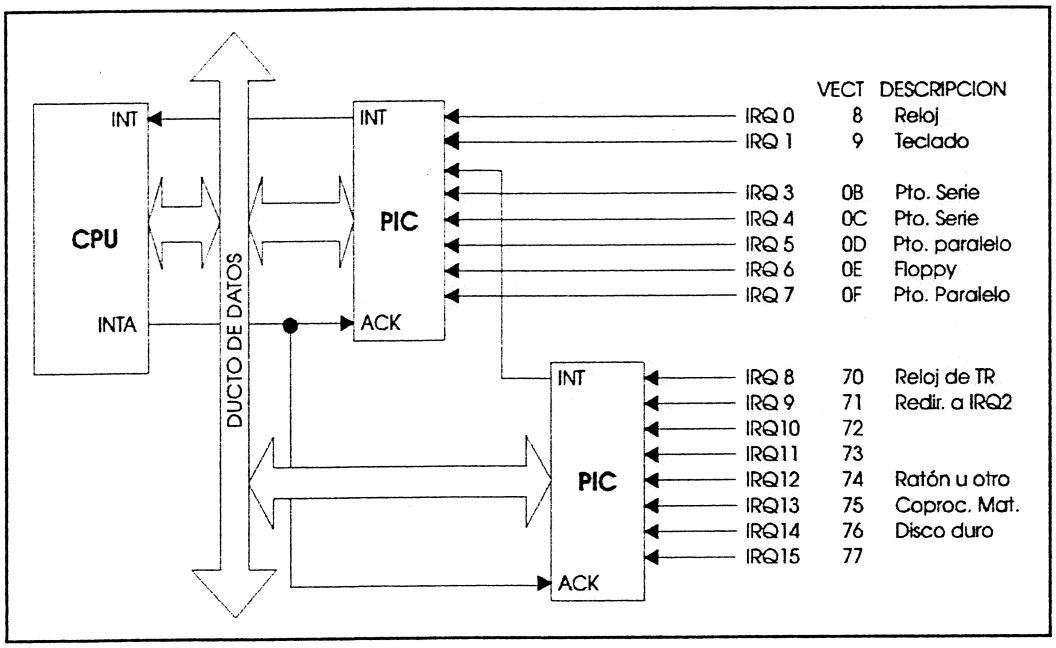


Fig. 3.2.3.Conexión de los Controladores de Interrupción Programables al procesador de la PC

El Controlador de interrupciones programable es inicializado por el BIOS de la computadora al encendido. Es en este punto donde se le programa cada una de las prioridades de los niveles de interrupción, los vectores de interrupción asociados a cada IRQ (Interrupt Request), la cantidad de PIC's conectados al procesador y su modo de operación.

La secuencia mediante la cual es atendida una interrupción solicitada por algún dispositivo externo conectado al PIC se enumera a continuación:

1. Una transición positiva en cualquiera de las líneas IRQ del PIC provoca que la salida INT del controlador de interrupciones cambie a un nivel lógico alto.

2. Este cambio de nivel es sensado por la línea INT del procesador, el cual pulsa un INTA. El INTA es sensado por la línea ACK del PIC, lo que provoca que si existen varias solicitudes de interrupción pendientes, en el controlador de interrupciones resuelva a cual se le dará

servicio, de acuerdo al algoritmo previamente programado. Si la solicitud de interrupción a la cual el PIC decide darle servicio proviene de algún PIC esclavo, se le concede el control para que él continue con los pasos pendientes.

3. El procesador pulsa un segundo INTA el cual al ser detectado por el PIC provoca que el vector de interrupción *n*, correspondiente a la IRQ que se le dará servicio sea colocado en el ducto de datos del procesador. Si existen solicitudes de interrupción pendientes de ser servidas, el PIC pulsará la línea INT para repetir el procedimiento anterior sólo si recibe el código EOI (End of Interrupt).

4. Con el vector de interrupción proporcionado por el PIC, el procesador forma la instrucción INT *n*, y el procedimiento siguiente se lleva a cabo de la misma forma que con las interrupciones por software (Intel, 1996).

La rutina responsable de dar servicio a la interrupción solicitada, además de tomar las precauciones enunciadas anteriormente para las interrupciones por software, debe enviar el código EOI (End of Interrupt) al PIC, para permitir que se le de servicio a otras solicitudes de interrupción de dispositivos conectados al PIC.

Si se desea que la ISR sea atómica, el código EOI debe colocarse hasta finalizar la ISR (antes de la instrucción IRET por supuesto). Esto evitará que la ISR sea interrumpida, dando la apariencia de una instrucción indivisible. Si el hecho de que la ISR sea interrumpida o no por alguna solicitud de interrupción a otra ISR o inclusive a la misma, no es importante, el código EOI puede colocarse al principio de la ISR (Después de salvaguardar registros en la pila).

La programación del PIC se realiza primero mediante una secuencia de inicialización, en la cual recibe un máximo de cuatro mandos a los que se les llama Inicializations Command Words (ICW's). Una vez que la Inicialización ha sido programada se le pueden programar tres diferentes

mandos de operación a los cuales se les llama Operational Command Words (OCW's). Los OCW's pueden ser programados y cambiados en cualquier momento durante la ejecución de un programa.

En la PC los ICW's y OCW's son recibidos por el PIC a través del puerto 20h y 21h en el PIC maestro, y por el puerto A0h y A1h en el PIC esclavo (Foster et al, 1998). El modo de inicialización del PIC maestro puede ser activado en cualquier momento poniendo en 1 el bit 4 del puerto 20h. Este primer acceso al puerto 20h corresponde a ICW1 y con él se le indica al PIC que los siguientes datos recibidos por el puerto 21h corresponden a ICW2 y posiblemente ICW3 e ICW4.

Existen solo 3 OCW's. OCW1 se pone en el puerto 21h, con él se le indica al PIC las IRQ's que deben enmascararse (deshabilitarse). OCW2 se escribe en el puerto 20h. Para indicar que se trata de OCW2 los bits 3 y 4 de este puerto deben estar en 0, el resto de los bits forman el mando en el cual se le indica al PIC, el algoritmo de priorización a usar, el código EOI (Non-specific End Of Interrupt) y el código SEOI (Specific End Of Interrupt). OCW3 se escribe en el puerto 20h. Para indicar que se trata de OCW3 el bit 4 de este puerto debe estar en 0 y el bit 3 en 1, el resto de los bit indican al PIC algunos mandos que programan la forma en que debe manipular datos en sus registros internos, así como algunas formas de enmascaramiento (Intel, 1996).

No se recomienda programar una secuencia de inicialización ni tampoco programar mandos de operación ya que esto afecta el modo de operación del hardware de la computadora. Es necesario señalar que el BIOS de la computadora realiza la secuencia de inicialización al encendido y además programa los mandos de operación (OCW's). Los dispositivos conectados al procesador tales como controlador de disco, floppy, puertos y el timer toman en cuenta los parámetros programados en el PIC por el BIOS para su funcionamiento, cualquier alteración a esa programación trastornará el modo de operar de la computadora.

Los únicos mandos que pueden usarse con el cuidado apropiado son OCW1 y OCW2. OCW1 se usa para desenmascarar (habilitar) cualquier IRQ que se desee usar, para hacerlo sólo se

pone en 0 el bit correspondiente a la IRQ. Por ejemplo si se desea habilitar la IRQ del primer puerto paralelo (IRQ5) se debe ejecutar la instrucción en lenguaje C:

```
outb( 0x21, inpb(0x21) & 0xEF );
```

con lo cual se pone el bit 5 del puerto 21h en 0 y los bits restantes se dejan sin cambio. OCW2 se usa para programar algunos mandos como lo son:

- El código EOI el cual consiste en poner el bit 5 del puerto 20h en 1, con lo cual se le indica al PIC que reinicialice en su registro interno de IRQ's el estado de la IRQ de más alta prioridad (La IRQ que se le está dando servicio).

- El código SEOI, consiste en poner en 1 el bit 6 del puerto 20h, con lo cual se le ordena al PIC que reinicialice alguna IRQ especificada con los bits 0, 1 y 2 de este puerto.

- Selección del algoritmo de asignación de prioridades. con el bit 7 del puerto 20h puesto en 1 se selecciona prioridad rotativa. esto quiere decir que a la IRQ que se le ha dado servicio se le asigna la más baja prioridad. Si el bit se pone en 0, (tal como está programado por el BIOS) la IRQ0 siempre tendrá la más alta prioridad seguida por IRQ1, IRQ2,...,IRQ7.

De acuerdo a lo anterior, para que la ISR le envíe al PIC el código EOI, sin modificar los parámetros programados previamente por él, debe formar el OCW2 con la siguiente instrucción en lenguaje C:

```
outb( 0x20, 0x20 );
```

La secuencia de inicialización tiene los siguientes pasos:

1. escribir ICW1.

2. escribir ICW2.

3. Si tiene PIC esclavo conectado, escribir ICW3.

4. Si en ICW1 se indicó que ICW4 era necesario, escribir ICW4.

La figura 3.2.4. muestra los parámetros que pueden programarse en cada uno de los ICW's. El programa siguiente realiza la secuencia de inicialización programada en el BIOS de la computadora (Foster, 1993).

```
/* Se inicializa el PIC maestro */
outb( 0x20, 0x11 );     /* ICW4 necesario */
outb( 0x21, 0x08 );     /* vector base (IRQ0) */
outb( 0x21, 0x04 );     /* en IRQ2 está conectado un PIC esclavo */
outb( 0x21, 0x01 );     /* ICW4 modo ix86 */
outb( 0x21, 0xFF );     /* Todas las IRQ's enmascaradas */
/* Se inicializa el PIC esclavo */
outb( 0xA0, 0x11 );     /* ICW4 necesario */
outb( 0xA1, 0x70 );     /* vector base (IRQ8) */
outb( 0xA1, 02 );       /* Redireccionado a IRQ2 */
outb( 0xA1, 0x01 );     /* ICW4 modo ix86 */
outb( 0xA1, 0xFF );     /* Todas las IRQ's enmascaradas */
```

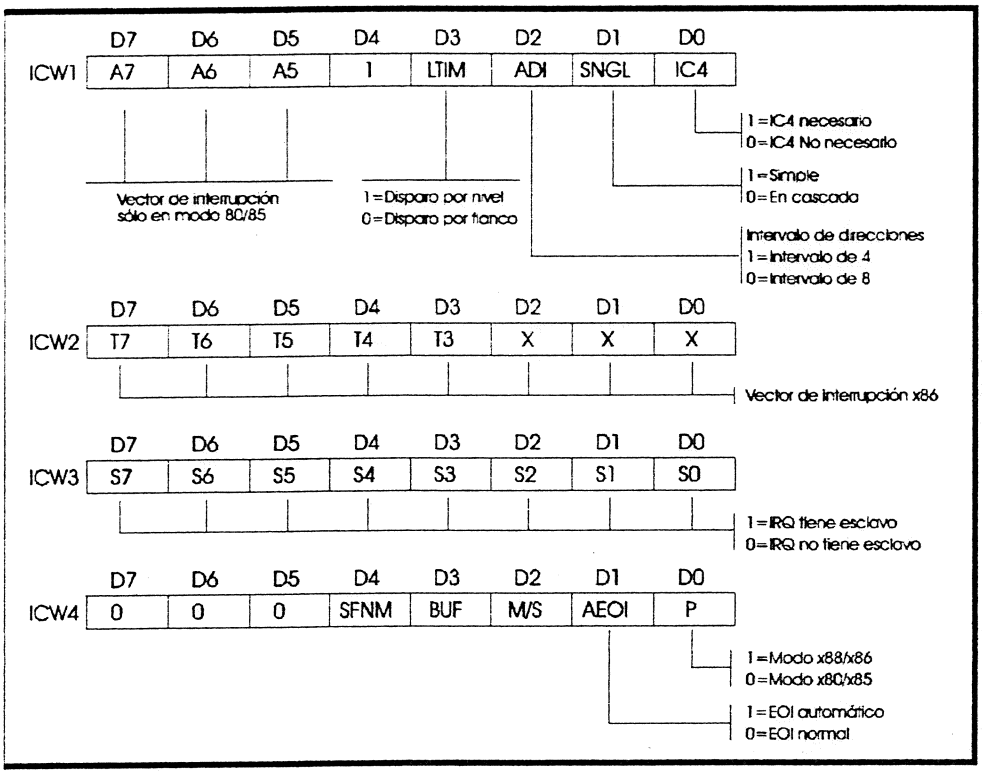


Fig. 3.2.4. Estructura de los ICW's.

3.3. El puerto paralelo

El puerto paralelo es un dispositivo diseñado principalmente para unir la computadora con una impresora. No existe ningún protocolo de comunicación asociado al estándar eléctrico del puerto, lo que hace posible que pueda ser conectado a él, cualquier dispositivo que acepte o transmita datos en paralelo a niveles TTL.

Las reglas que definen la transmisión de datos entre el puerto y cualquier dispositivo conectado a él, pueden ser definidas por el programa que se ejecuta en la computadora, tomando en cuenta para ello las características del dispositivo. Esta libertad permite el desarrollo de diferentes aplicaciones, ya sea de comunicación, de adquisición de datos y de control.

La conexión física a este puerto se realiza mediante un conector de 25 terminales (DB-25) que puede ser encontrado en la parte posterior de la computadora. La Fig. 3.3.1. muestra dicho conector, en el cual, los nombres que se le han etiquetado a cada terminal corresponden a la conexión con una impresora en paralelo. Sin embargo se debe tomar en cuenta que no sólo pueden conectarse impresoras a dicho conector como se dijo anteriormente. Las etiquetas asignadas a las terminales del conector se usarán en lo sucesivo para indicar cada bit en el registro al cual acceden dentro del puerto paralelo.

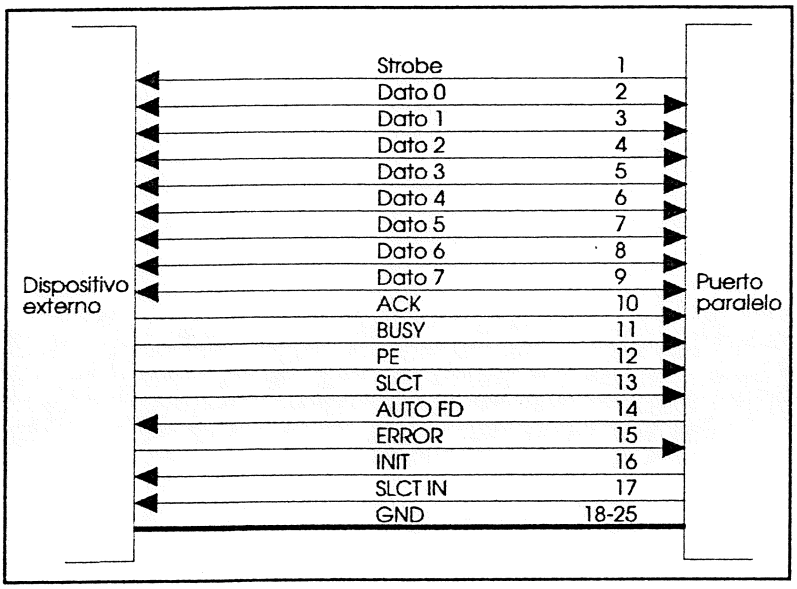


Fig. 3.3.1. Conector DB-25 del puerto paralelo.

El puerto paralelo cuenta con tres registros de 8 bits cada uno, de los cuales uno es bidireccional y se denominará en lo sucesivo como registro de datos, los otros dos, uno es entrada y el otro es de salida. Un simple cálculo aritmético nos daría como resultado que son 24 bits con los que cuenta el puerto paralelo para conectarse al exterior, sin embargo, del registro de salida sólo los 4 bits menos significativos (0-3) están unidos al conector DB-25, y del registro de entrada sólo están unidos los 5 bits más significativos (3-7). Esto da como resultado que solo 17 bits estén disponibles de los 24 con los que se cuenta.

El mapa de memoria de entrada/salida de la computadora designa 3 grupos de direcciones para igual cantidad de puertos paralelos, sin embargo sólo existen dos IRQ's designadas a este tipo de puertos, por lo que sólo dos de ellos pueden coexistir simultáneamente en una computadora. Los grupos de direcciones asignadas a los puertos son (Foster et al, 1998):

- Para el puerto 0. A partir de la 3BCh hasta la 3BFh.
- Para el puerto 1. A partir de la 378h hasta la 37Fh.
- Para el puerto 2. A partir de la 278h hasta la 27Fh.

La letra h al final de la dirección indica "hexadecimal". Note que el grupo de direcciones asignado al puerto cero contiene cuatro bytes, mientras que los otros dos contienen ocho. Debido a que el puerto paralelo contiene sólo tres registros de un byte cada uno, el resto de los bytes sobrantes en el grupo de las direcciones que se le asignan en la computadora quedan sin uso.

A continuación se detallará el uso de cada bit en los tres registros del puerto paralelo.

REGISTRO DE DATOS (Dirección base): Este es un registro bidireccional, sus ocho bits se usan para transmitir o recibir bytes en paralelo (todos los bits a la vez).

REGISTRO DE ENTRADA (Dirección base + 1): Este registro se usa generalmente para reportar el estado del dispositivo al cual está conectado el puerto paralelo. Sólo 5 bits tienen contacto con el

exterior del puerto ya que son los únicos que están unidos al conector DB-25 entre los cuales se encuentran:

BIT 7	BIT 6	BIT 5	BIT 4	BIT 3	BIT 2	BIT 1	BIT 0
BUSY	ACK	PE	SLCT	ERROR	NU	NU	NU

Con la abreviación NU se indica bit no usado. El estado lógico del bit ACK es por omisión '1'. Cuando este bit cambia al estado lógico '0', y se mantiene ese estado por cuando menos 5 microsegundos, si la IRQ de este puerto está habilitada, provocará que suceda una interrupción en la entrada del Controlador de interrupciones correspondiente a este puerto paralelo.

PUERTO DE SALIDA (Dirección base+2): Este registro se usa generalmente para enviar mandos de control y/o configuración al dispositivo conectado al puerto paralelo. Sólo 4 bits de este registro están unidos al conector DB-25 los cuales son:

BIT 7	BIT 6	BIT 5	BIT 4	BIT 3	BIT 2	BIT 1	BIT 0
NU	NU	NU	IRQ	SLCT IN	INIT	AUTO FD	STROBE

El bit IRQ se programa en '1' para permitir que el puerto paralelo pueda generar una interrupción cuando la entrada el bit de ACK cambie de '1' a '0'. A este bit se le escribe '0' para evitar que el puerto interrumpa a la computadora, sobre todo cuando se va a leer la entrada a puerto usando el método de captura de datos por encuesta.

4. Desarrollo de la programación

4.1. El generador y analizador de movimiento armónico

El generador y analizador de movimiento armónico es un aparato mecánico con el que cuenta el Laboratorio de Física de la Facultad de Ciencias para realizar experimentos relacionados con movimiento armónico. Este aparato produce oscilaciones mecánicas perfectamente visibles, las cuales son sensadas por detectores ópticos. De esta forma procesa y obtiene el periodo y la amplitud instantánea de cada oscilación. Tales parámetros los presenta en un despliegue numérico. Es necesario resaltar que el aparato no tiene capacidad de registrar el movimiento que mide, por lo que es necesaria la intervención humana para que supervise los despliegues numéricos y registre los cambios instantáneos de los parámetros que el aparato muestra. La figura 4.1.1. presenta al manejador y analizador de movimiento armónico al cual se le desarrolló la programación que se describe en este trabajo.

La mecánica y la electrónica contenida en el generador y analizador de movimiento armónico permite controlar y medir la mayoría de los parámetros que forman parte del movimiento armónico en cualquiera de sus formas. Los parámetros que pueden ser controlados son (Pasco, 1990):

1. La constante del resorte.
2. La masa.
3. La frecuencia motriz.
4. La amplitud motriz.
5. El amortiguamiento.

Las oscilaciones que se producen en el sistema dependen de como sean fijados estos parámetros. Cada oscilación tiene amplitud y tiene periodo los cuales pueden ser medidos por el aparato.

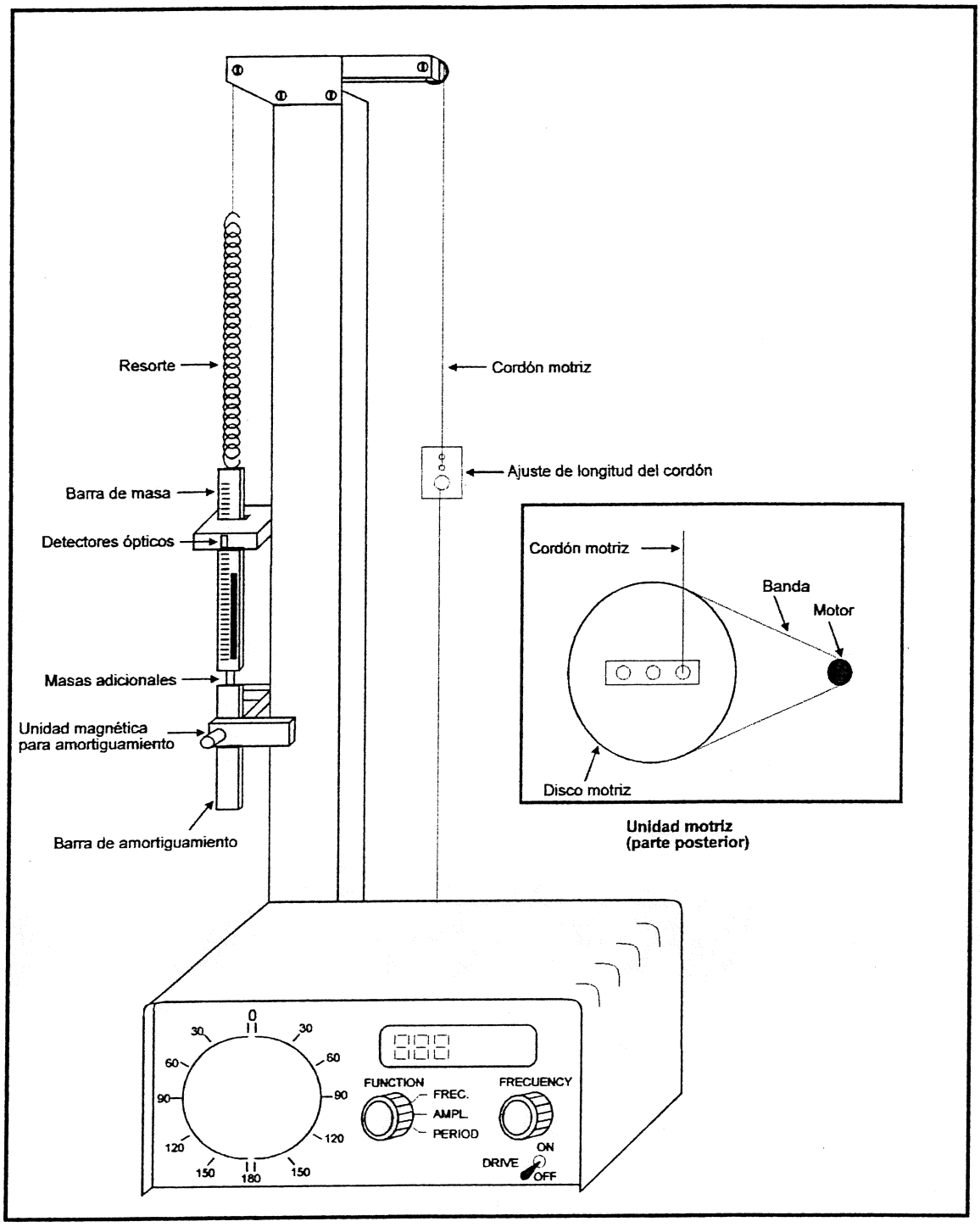


Fig.4.1.1. Generador y analizador de movimiento armónico.

En la parte posterior del aparato se encuentra un conector de salida DB-9 el cual provee señales TTL. Este conector permite que el manejador y analizador de movimiento armónico sea conectado a una computadora con el propósito de capturar los datos en forma automática para procesamiento, registro, y análisis.

4.1.1. Parámetros del sistema

Los párrafos siguientes describen los parámetros que pueden ser controlados por el experimentador los cuales fueron enumerados en la sección previa.

1. La constante del resorte.

El sistema cuenta con cuatro resortes, cada uno con una constante diferente. Estos resortes pueden ser intercambiados con el fin de modificar este parámetro en el sistema.

2. La masa.

La barra de masa, la barra de amortiguamiento y el resorte, forman una masa aproximada de 50 gramos. En adición a lo anterior, pueden adicionarse al sistema una ó dos masas adicionales proveídas en el equipo. Cada una de esas masas adicionales equivale a 50 gramos. Estas masas se montan entre la barra de masa y la barra de amortiguamiento, como lo muestra la Figura 4.1.1.

3. La frecuencia motriz.

La frecuencia motriz es ajustada girando la perilla FRECUENCY que se encuentra en la parte frontal del aparato. Para medir la frecuencia motriz, se debe seleccionar FREC. en la perilla de FUNCTION y el despliegue numérico mostrará la frecuencia con la cual gira el disco motriz.

4. La amplitud motriz.

En la parte posterior del aparato se localiza el disco motriz, el cual cuenta con una barra en la cual está sostenido el cordón motriz. Para ajustar la amplitud motriz se deben aflojar los tornillos que sujetan la barra, después se desliza la barra con el propósito de que el cordón motriz quede sujetado a una distancia r medida a partir del centro del disco. La amplitud motriz está determinada por el doble de r. Una vez que se ha ajustado la amplitud motriz se aprietan nuevamente los tornillos de la barra para fijar dicha amplitud.

5. El amortiguamiento.

La unidad magnética para amortiguamiento la componen la barra de amortiguamiento y dos magnetos. Los magnetos están unidos cada uno a un tornillo, con el propósito de que al ser girado pueda alejar o acercar los magnetos de la barra de amortiguamiento dependiendo del sentido del giro . El amortiguamiento puede variarse acercando o alejando los magnetos de la barra de amortiguamiento. Otra forma de producir amortiguamiento se logra provocando que la barra de amortiguamiento se mueva sumergida dentro de un líquido. El amortiguamiento logrado mediante este método depende de la densidad del liquido utilizado.

6. Amplitud y periodo del sistema formado por el resorte y la masa.

El generador y analizador de movimiento armónico puede medir la amplitud y el periodo instantáneos del sistema formado por el resorte y la masa, para medir la amplitud, se gira la perilla de FUNCTION de tal forma que seleccione AMPL. Los despliegues numéricos mostrarán entonces la amplitud de cada oscilación producida por el sistema. Para medir el periodo se procede de la forma antes descrita sólo que ahora se selecciona PERIOD con la perilla de FUNCTION.

Debido a que el aparato no tiene capacidad de registro de los parámetros que es capaz de medir, se requiere que el experimentador esté atento a las variaciones de los despliegues numéricos con el fin de que sea él quien registre los resultados. Por otro lado no es posible que el aparato despliegue amplitud y

periodo simultáneamente por lo que en un solo experimento no pueden obtenerse ambos parámetros. Las dos causas anteriores muestran la necesidad de poder registrar instantáneamente y sin la intervención humana los parámetros que pueden ser medidos por el aparato, para que puedan ser analizados y procesados de tal forma que se logren experimentos más reales y apegados a la precisión que es capaz de lograr el manejador y analizador de movimiento armónico.

4.1.2. Los detectores ópticos

El generador y analizador de movimiento armónico cuenta con cuatro fotodetectores, dos de ellos monitorean la posición de la barra de masa y dos monitorean la posición del disco motriz.

Cada fotodetector consiste de un emisor de luz (L.E.D) y un fototransistor. En la figura 4.1.1. se puede apreciar la localización de los dos fotodetectores que monitorean la posición de la barra de masa.

El sistema óptico de monitoreo de posición del movimiento inducido por la unidad motriz, está formado por la barra de masa y por dos detectores ópticos. La barra de masa es una barra transparente con bandas opacas marcadas perpendicularmente y equiespaciadas dos milímetros, la longitud de cada banda es aproximadamente un tercio del ancho de la barra, a estas bandas las denominaremos en lo sucesivo "bandas de posición". La barra tambien está marcada con una banda opaca longitudinal que cubre la mitad de la barra, a esta banda le llamaremos en lo sucesivo "banda de signo". Los fotodetectores se encuentran fijos, situados uno en la trayectoria de las bandas de posición y el otro en la trayectoria de la banda de signo.

La barra de masa se mueve debido a la acción de la unidad motriz. La luz emitida por el L.E.D. cruza a través de la barra y es detectada por el fototransistor el cual genera un voltaje positivo que es interpretado como un estado lógico "1". Si la luz es interrumpida por una banda opaca, el fototransistor no puede detectarla. Este efecto provoca que el fototransistor no genere un voltaje, lo cual es interpretado como un estado lógico "0". En el apéndice 1, se incluye el diagrama eléctrico del manejador y analizador de movimiento armónico, en el cual se pueden apreciar las señales de los fotodetectores que están conectadas al conector DB-9. La leyenda "mass bar phase out" en nuestra notación equivale a la banda de

signo, mientras que la leyenda "mass bar amplitude out" equivale a las bandas de posición. Estas dos señales son las únicas que usa el programa para obtener los parámetros periodo y amplitud, al mismo tiempo que sirven para construir la función del movimiento producido por el sistema.

4.1.3. La ecuación de movimiento

La figura 4.1.2. muestra el diagrama del generador y analizador de movimiento armónico con algunas de las variables involucradas en el análisis matemático. En el diagrama, la barra de masa está por debajo de su posición de equilibrio a una distancia x, moviéndose hacia arriba con una velocidad V.

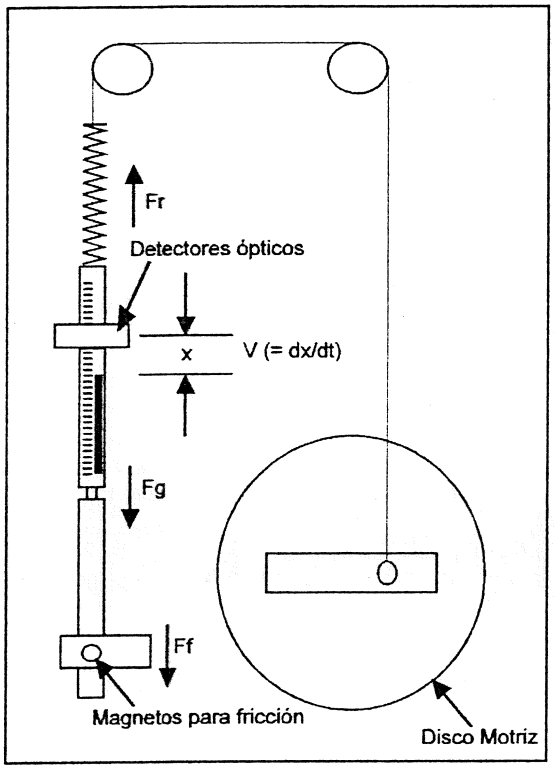


Fig. 4.1.2. Diagrama del generador y analizador de movimiento armónico.

La ecuación de movimiento para la masa del vástago puede ser determinada usando la segunda ley de Newton

Para hacer esto, deben determinarse las fuerzas que actúan en el vástago cuando este es desplazado desde su posición de equilibrio. Estas fuerzas, tal como lo muestra el diagrama son:

- gravedad (Fg).
- La fuerza de restauración del resorte (Fr).
- La fricción que pueda detener el movimiento del sistema (Ff).
- La fuerza motriz (Fm). Producida cuando el cordón en la parte superior del resorte se desplaza. Esta fuerza es tomada en cuenta cuando se determina la fuerza de restauración del resorte Fr.

La fuerza de gravedad.

$$Fg = m.g$$

donde :

m es la masa de la barra graduada.

g es la aceleración de la gravedad.

La fuerza de restauración del resorte.

De acuerdo a la ley de Hook, la fuerza ocasionada al estirar un resorte, es proporcional a la distancia debajo de su longitud natural a la cual el resorte es estirado. Tal relación puede ser expresada como

$$Fr = -k.x;$$

donde:

k es la constante de estiramiento del resorte y,

x es la diferencia entre la longitud natural del resorte y la longitud del estiramiento experimentado.

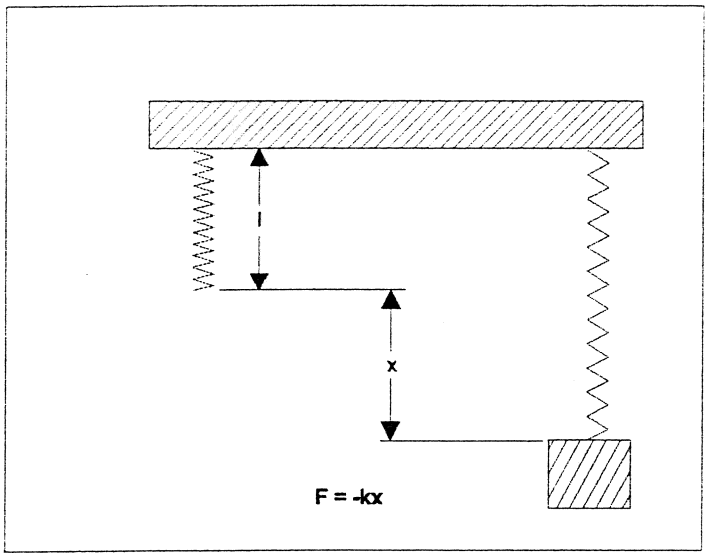


Fig. 4.1.3. Ley de hooke

Para el generador y analizador de movimiento armónico la relación anterior quedará como sigue.

$$Fr = -k[\,(x - x1) - L)\,] \qquad (1)$$

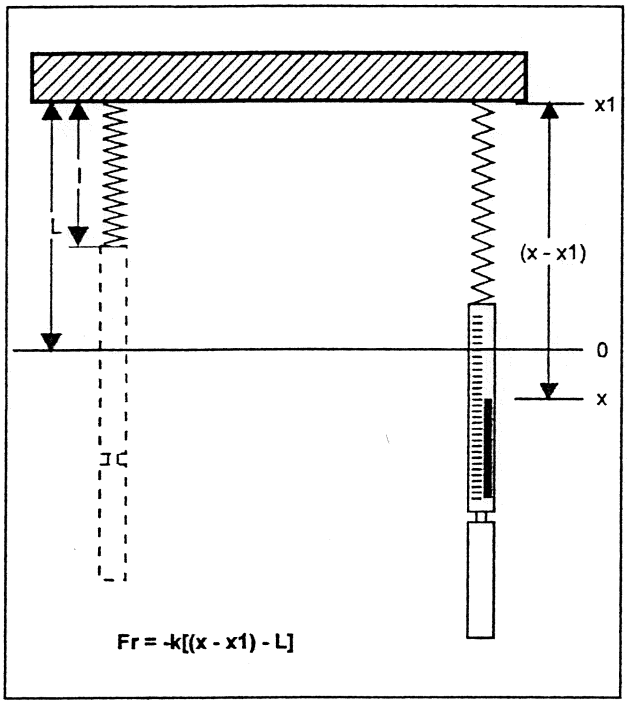


Fig. 4.1.4. Ley de hooke generalizada.

La fuerza de fricción

Para determinar la fuerza de fricción Ff, se asume que Ff es proporcional a la velocidad de la masa de la barra. Por consiguiente,

$$Ff = B\frac{d}{dt}x$$

donde:

B es llamada constante de amortiguamiento o constante de proporcionalidad. el signo negativo indica que la fricción actúa en dirección opuesta a la velocidad de la barra.

Al insertar las fuerzas anteriores a la Ley de Newton tenemos:

$$F = Fg + Fr + Ff$$

$$m\frac{d^2}{dt}x = mg - k(x - x1 - L) - B\frac{d}{dt}x$$

$$m\frac{d^2}{dt^2}x + B\frac{d}{dt}x + kx = kx1 + kL + mg$$

La ecuación anterior puede ser simplificada mediante una sustitución. L fue definida como la longitud natural del resorte mas la mitad de la longitud de la barra graduada (Figura 4.1.4). Considerando que el estiramiento del resorte desde su longitud natural es debida a la fuerza de gravedad nos queda que:

$$Fr = -k\,(L0 - L) = -m.g$$

De donde despejando para L nos queda:

$$L = L0 - mg / k.$$

sustituyendo esta expresión en la ecuación de movimiento nos queda:

$$m\frac{d^2}{dt^2}x + B\frac{d}{dt}x + kx = k(x1 + L0) \qquad (2)$$

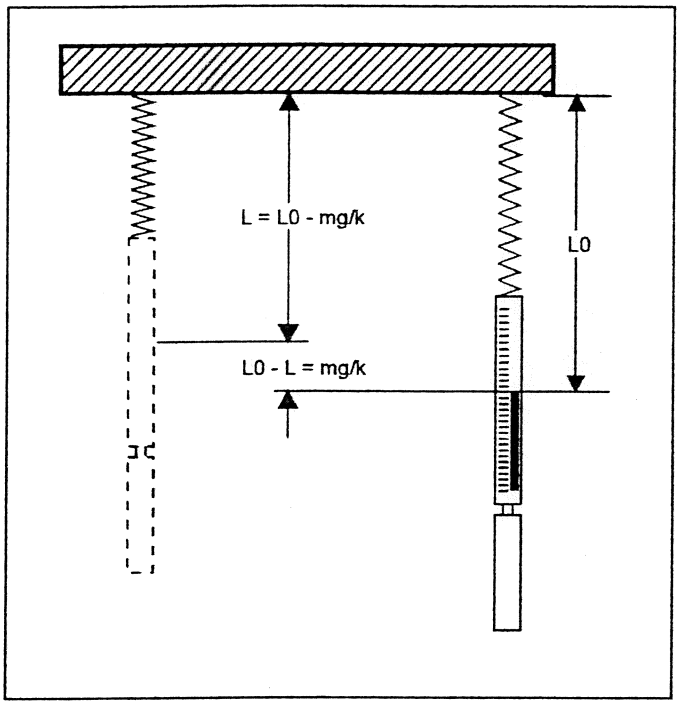


Fig.4.1.5. Definición de L0

La variable x1 está determinada por la posición de la rueda motriz y viene a ser:

$$x1 = -L0 + A0.sen(wt)$$

Sustituyendo para x1 en la ecuación (2)

$$m\frac{d^2}{dt^2}x + B\frac{d}{dt}x + kx = kAsen(wt) \qquad (3)$$

Esta es la ecuación general del movimiento armónico. Incluye varios parámetros que pueden ser ajustados de acuerdo a un sistema en particular. Los parámetros son:

- m: la masa.
- B: el coeficiente de amortiguamiento (fricción).
- k: la constante del resorte.

- A: la amplitud motriz.
- ω: la frecuencia motriz.

4.1.4. Solución a la ecuación de movimiento

La ecuación 3 es una ecuación diferencial lineal de segundo orden no homogenea. Existen métodos estándares para resolver este tipo de ecuaciones los cuales pueden ser encontrados en cualquier texto introductorio de ecuaciones diferenciales. A manera de ejemplo se deducirá la solución a la ecuación de movimiento, correspondiente a oscilaciones libres, y se popondrá la solución a la ecuación general.

La ecuación correspondiente a oscilaciones libres queda:

$$My'' + By' + ky = 0 \qquad (1)$$

Esta es una ecuación diferncial lineal de segundo orden homogenea, cuya solución general (Piskunov, 1983) está dada por:

$$y = C_1 e^{k1t} + C_2 e^{k2t}$$

K1 y *k2* son las raíces de la ecuación característica respecto a la ecuación (1) la cual es:

$$mq^2 + Bq + k = 0$$

Partiendo de esta ecuación se analizarán los casos correspondientes a oscilaciones libres.

Caso 1: Oscilaciones libres, no amortiguadas (B = 0)

La ecuación diferencial y su ecuación característica correspondiente son:

$$My'' + ky = 0 \qquad (2)$$

$$mq^2 + k = 0$$

Sus raíces son complejas y conjugadas:

$$q = \pm\sqrt{\frac{k}{m}}i$$

Se inserta la variable:

$$w = \sqrt{\frac{k}{m}}$$

Quedando la solución:

$$y = C_1\cos(wt) + C_2 sen(wt) \qquad (3)$$

Se sustituye en la ecuación (3) las constantes C1 y C2 por otras A y θ ligadas con C1 y C2 por las relaciones $C1 = Asen(\theta)$ y $C2 = Acos(\theta)$ las cuales se determinan asi:

$$A = \sqrt{C_1^2 + C_2^2} \quad y \quad \theta = \tan^{-1}\frac{C_1}{C_2} \qquad (4)$$

Introduciendo estos nuevos valores a la ecuación (3) queda:

$$y = Asen(\theta)cos(wt) + Acos(\theta)sen(wt)$$

Ó

$$y = Asen(wt + \theta) \qquad (5)$$

A las oscilaciones de esta función se les llama armónicas. En este caso el periodo es:

$$2\pi/w$$

Y la amplitud de las oscilaciones es A.

Para encontrar el valor de la constante A consideremos las condiciones iniciales y = y0, V = y' = V0 cuando t = t0 = 0 en la ecuación (3).

$$y = C1\cos(wt) + C2\,sen(wt)$$

$$y' = -C1w\,sen(wt) + C2w\cos(wt)$$

de donde:

$$y0 = C1\cos(wt) = C1$$

$$V0 = C2w\cos(wt)$$

$$C2 = V0 / w$$

De acuerdo a las consideraciones hechas en (4) nos queda:

$$A = \sqrt{y_0^2 + \left(\frac{V_0}{w}\right)^2} \quad y \quad \theta = \tan^{-1}\frac{y_0}{\left(\frac{V_0}{w}\right)} = \tan^{-1}\left(\frac{wy_0}{V_0}\right)$$

La solución particular a la ecuación (2) es entonces:

$$y = \sqrt{y_0^2 + \left(\frac{V_0}{w}\right)^2}\, sen(wt + \theta)$$

$$w = \sqrt{k/m}$$

$$\theta = \tan^{-1}\left(\frac{wy_0}{V_0}\right)$$

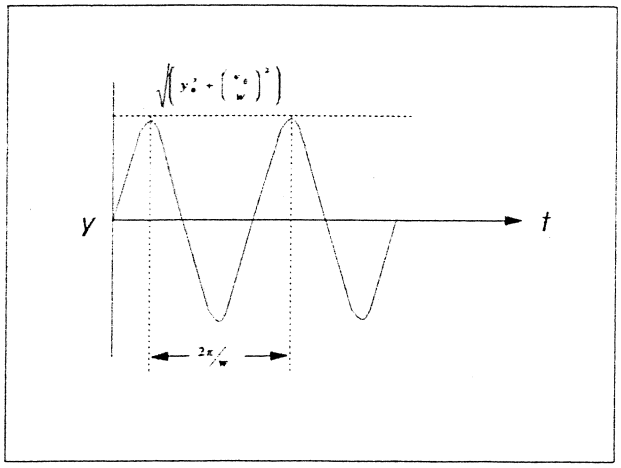


Fig. 4.1.6. Oscilaciones libres, no amortiguadas

Caso 2: Oscilaciones libres, amortiguadas (B <> 0)

La ecuación diferencial y la ecuación característica correspondiente a este caso son:

$$My'' + By + ky = 0 \qquad (1)$$

$$mq^2 + Bq + k = 0$$

Las raíces correspondientes a esta ecuación son:

$$q_{1,2} = -\frac{B}{2m} \pm \sqrt{\left(\frac{B}{2m}\right)^2 - \frac{k}{m}} \qquad (2)$$

De acuerdo a estas raices podemos tener soluciones de tres tipos diferentes las cuales dependen del coeficiente de amortiguamiento B. El valor de B determinará que las raices de la ecuación caracteristicas sean.

i) Reales e iguales.

ii) Reales y diferentes.

iii) Imaginarias

Solución de tipo i)

Las raíces son reales e iguales cuando

$$B = 2m\sqrt{\frac{k}{m}}$$

Las raices son entonces:

$$q_1 = -\frac{B}{2m} \qquad q_2 = -\frac{B}{2m}$$

Por lo que la solucion a la ecuación (1) queda:

$$y = (C_1 + C_2 t)e^{-\left(\frac{B}{2m}\right)t} \qquad (3)$$

$$V = y' = \frac{C_2 - (C_1 + C_2 t)e^{\left(\frac{B}{2m}\right)t}\left(\frac{B}{2m}\right)}{e^{\left(\frac{B}{2m}\right)t}}$$

Para encontrar los valores C_1 y C_2 insertamos las condiciones iniciales de
V = V_0 y $y = y_0$ cuando $t = t_0 = 0$

$y_0 = C_1$

$$V_0 = C_2 - C_1\frac{B}{2m} = C_2 - y_0\frac{B}{2m}$$

de donde:

$$C_2 = V_0 + y_0\left(\frac{B}{2m}\right)$$

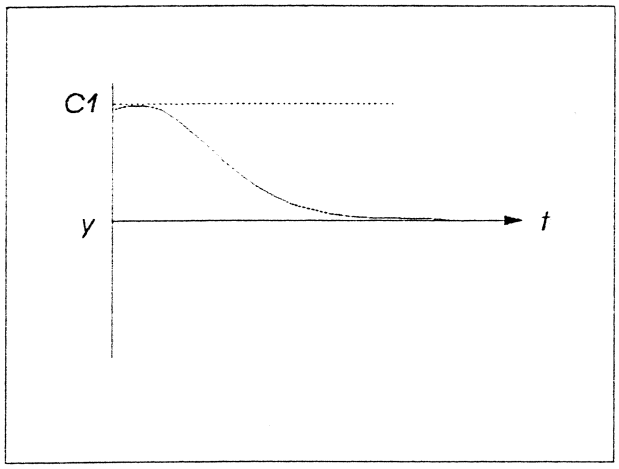


Fig. 4.1.7. Oscilaciones libres y críticamente amortiguadas

Solución de tipo ii)

Las raíces de la ecuación característica son reales y diferentes cuando

$$B > 2m\sqrt{\frac{k}{m}}$$

Las raices son las que se muestran en (2) por lo que la solución a la ecuacion (1) es:

$$y = C_1 e^{\beta_1 t} + C_2 e^{\beta_2 t} \qquad V = y' = C_1\beta_1 e^{\beta_1 t} + C_2\beta_2 e^{\beta_2 t}$$

$$\beta_1 = -\frac{B}{2m} + w$$

$$\beta_2 = -\frac{B}{2m} - w$$

$$w = \sqrt{\left(\frac{B}{2m}\right)^2 - \frac{k}{m}}$$

Para encontrar C1 y C2 consideramos las condiciones iniciales y = y0, V = V0 y t = t0 =0.

$$C_1 = \frac{V_0 - y_0\beta_2}{\beta_1 - \beta_2}$$

$$C_2 = \frac{V_0 - y_0\beta_1}{\beta_2 - \beta_1}$$

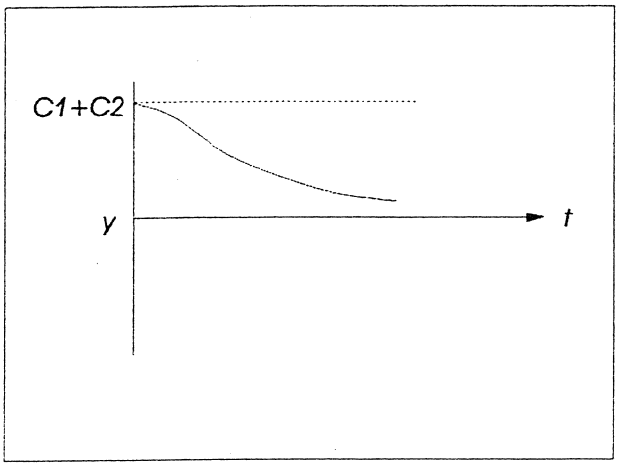


Fig. 4.1.8. Oscilaciones libres y sobreaomortiguadas

Solución de tipo iii)

Para que las raíces de la ecuación característica sean imaginarias es necesario que

$$B < 2m\sqrt{\frac{k}{m}}$$

Las raices de la ecuación característica son complejas y conjugadas

$$q_1 = -\frac{B}{2m} + \sqrt{\frac{k}{m} - \left(\frac{B}{2m}\right)^2}\, i \qquad q_1 = -\frac{B}{2m} - \sqrt{\frac{k}{m} - \left(\frac{B}{2m}\right)^2}\, i$$

Se inserta la variable:

$$w = \sqrt{\frac{k}{m} - \left(\frac{B}{2m}\right)^2}$$

Quedando la solución:

$$y = e^{-\left(\frac{B}{2m}\right)t}\left(C_1 \cos(wt) + C_2 sen(wt)\right) \qquad (4)$$

Se procede como en el caso 1 correspondiente a oscilaciones libres no amortiguadas para deducir la solución.

$$y = Ae^{-(B/2m)t} sen(wt + \theta)$$

$$w = \sqrt{\frac{k}{m} - \left(\frac{B}{2m}\right)^2}$$

$$\theta = \tan^{-1}\left(\frac{wy_0}{V_0 + \frac{By_0}{2m}}\right)$$

$$A = \sqrt{y_0^2 + \left(\frac{V_0}{w} + \frac{By_0}{2mw}\right)^2}$$

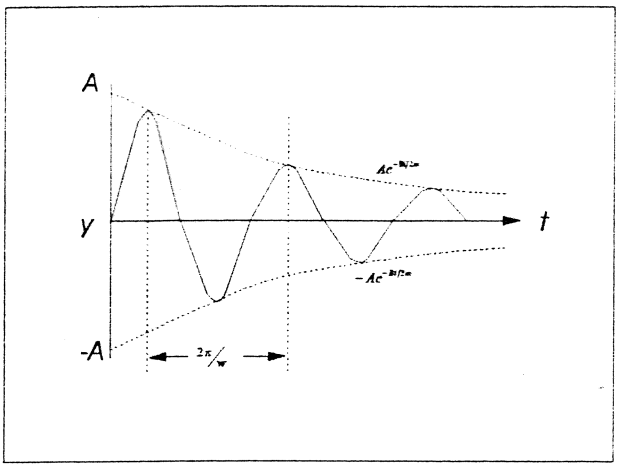


Fig. 4.1.9. Oscilaciones libres y amortiguadas

Caso 3: Oscilaciones forzadas

En este caso se considera que la amplitud del disco motriz está dada por

$$F = A_0 sen(wdt).$$

La solución a la ecuación del generador de movimiento armónico que se prèsenta a continuación incluye dos tèrminos: el primero corresponde a la solución del oscilador libre apropiado (Piskunov, 1983) (como se mostró en los casos anteriores). El segundo término es dependiente de la amplitud y frecuencia de la fuerza motriz.

$$y = (\text{solución a la ecuación homogenea}) + \frac{kA_0 sen(w_d t + \theta)}{\sqrt{m^2(w^2 - w_d{}^2) + B^2 w_d{}^2}}$$

$$\theta = \tan^{-1}\left(\frac{Bw_d}{m(w^2 - w_d{}^2)}\right)$$

$$w = \sqrt{\frac{k}{m}}$$

4.1.5. La captura de datos por interrupción

Tal como se describió en la sección 4.1.2. para obtener los parámetros del movimiento producido por el sistema experimental, el manejador y analizador de movimiento armónico cuenta con 4 fotodetectores de los cuales en el presente trabajo sólo se usan dos de ellos -los encargados de monitorear la posición de la barra de masa.

Para desarrollar el programa encargado de capturar y medir los parámetros del movimiento fue necesario hacer un cable. El cual está formado por un conector DB-9 en uno de sus extremos, el cual se conecta al manejador y analizador de movimiento armónico, y un conector DB-25 en el otro extremo el cual se conecta al puerto paralelo de la computadora. De acuerdo al diagrama eléctrico del manejador y analizador de movimiento armónico (Apéndice 1). Son seis las señales que el circuito es capaz de generar o detectar, de las cuales sólo cuatro están unidas al conector de salida DB-9, mismas que a su vez se conectaron al puerto paralelo como se muestran en la tabla IV.I.I

CONECTOR DB-9 (Manej. y An. de Mov. Arm.)	CONECTOR DB-25 (Puerto paralelo)
Disk Start/Stop Out	PE
Mass Bar Phase Out (Signo)	BUSY
Mass Bar Amplitude Out (Posición)	ACK
Ground Out	GND

Tabla IV.I.I. Conexiones entre el generador y analizador de movimiento armónico y el puerto paralelo.

De las conexiones entre el DB-9 y el DB-25 puede deducirse fácilmente que cada vez que la señal detectora de las bandas de posición pase del estado lógico '0' al estado lógico '1' provoca que el puerto paralelo solicite al controlador de interrupciones que interrumpa al procesador y ejecute una rutina de servicio a la interrupción solicitada. En el sistema experimental real, lo anterior significa que el detector óptico de la banda de posición determinó que la posición de la barra de masa se ha incrementado en dos milímetros, para determinar el sentido del movimiento, se tiene el conocimiento previo de que el movimiento de la barra de masa es periódico, o sea, que partiendo de una posición cualquiera, se mueve en un sentido

hasta una posición *p*, luego cambia de sentido, pasa por la posición *cero*, y continúa hasta la posición *-p*, para cambiar de sentido y repetir su paso por la posición cero, y continuar sucesivamente dicho movimiento. Muestreando en cada interrupción el detector de la banda de signo, es posible conocer el instante en el cual la barra de masa cambia de signo, lo cual significa que pasa por la posición cero y se mueve con el sentido del signo al cual cambió.

De acuerdo al análisis anterior del sistema experimental real, puede deducirse que lo que debe hacerse por cada interrupción, es:

1. muestrear y registrar el signo del movimiento.
2. si el signo de la interrupción actual es diferente al signo registrado en la interrupción anterior, el contador de posición debe ponerse en cero y registrarse. En caso contrario el contador de posición debe incrementarse en 1 y registrarse.
3. Debe obtenerse el tiempo en el cual sucedió la interrupción y registrarse junto con los parámetros anteriores.

El procedimiento anterior es ejecutado por la rutina de servicio de interrupción (ISR). El código se presenta en lenguaje de programación C en el apéndice 2 (líneas 207-217). Los parámetros que se registran por medio de este procedimiento se almacenan en el arreglo Func[MAX_PTS] (Linea 115), en el que cada elemento del arreglo es una estructura la cual está definida entre las líneas 76-80.

La gráfica obtenida por el procedimiento anterior se presenta en la figura 4.1.10. Debe apreciarse que aunque la gráfica muestra una función armónica, no es esta la que representa el movimiento producido en el sistema experimental que se analiza, sin embargo provee los elementos necesarios para obtener tanto los parámetros del movimiento armónico (amplitud y periodo) que se busca como la función que representa dicho movimiento.

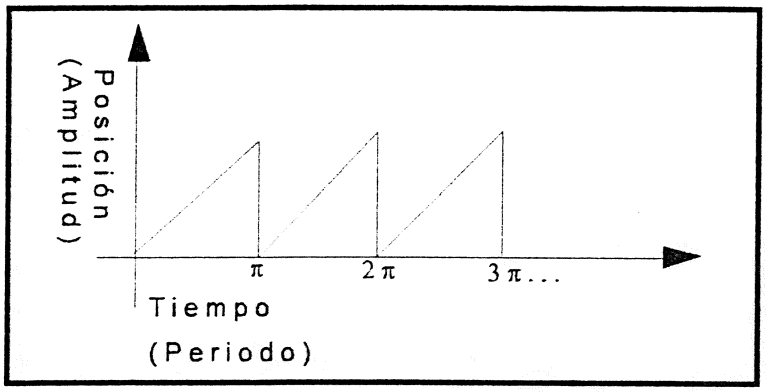


Fig. 4.1.10. Gráfica obtenida por la ISR del puerto paralelo.

Debe considerarse que antes de que pueda ejecutarse la ISR cuando menos por primera vez, es necesario que el puerto paralelo solicite una interrupción al controlador de interrupciones, y que este a su vez esté programado para responder a esa petición. Para que lo anterior pueda ser posible es necesario realizar lo siguiente:

1. Programar al puerto paralelo para que este sea capaz de generar una petición de interrupción cuando su bit ACK cambie de '0' a '1'. Lo anterior se hace escribiendo un '1' en el bit IRQ ENABLE del registro de entrada/salida del puerto paralelo.

2. escribir un OCW1 en el controlador de interrupciones mediante el cual se habilite la interrupción del puerto paralelo.

3. Salvaguardar el contenido del vector de interrupción correspondiente a la IRQ del puerto paralelo y escribir en ese vector la dirección de la ISR que se desea se ejecute.

El procedimiento anterior (líneas 512-523) prepara al puerto paralelo para capturar los datos del manejador y analizador de movimiento armónico por medio de interrupciones.

Para medir el tiempo con una resolución de 5 milisegundos (por omisión se mide a 53 mlisegundos), con el fin de lograr mayor precisión en los experimentos, se provoca que el contador de intervalos de tiempo interrumpa al procesador 200 veces por segundo. Por cada interrupción se ejecuta una ISR que incrementa un contador (Tiempo) que al inicio del experimento es igual a cero. El procedimiento

para lograr lo anterior es semejante al realizado para preparar al puerto paralelo para interrumpir al controlador de interrupciones. El código necesario para modificar la resolución del contador de intervalos se muestra en las líneas 524 a la 532. El código para habilitar su interrupción y que se ejucute la ISR correspondiente se encuentra en las líneas 505 a la 511 y la correspondiente ISR en las líneas 500 a la 504.

En resumen, son dos las ISR que se están ejecutando a lo largo de cada experimento, una para la captura de los datos y la otra para la medición del tiempo. Debe cuidarse de que los dispositivos que se están usando (contador de intervalos y puerto paralelo) una vez terminado el experimento queden configurados de la misma forma que antes de ser usados, de igual forma debe procederse con el programador de interrupciones y los vectores de interrupción. El código que realiza estas tareas se describe en las líneas 283 a la 293 para el puerto paralelo y 250, 251 y 252 para el contador de intervalos.

4.1.6. Obtención de los parámetros del movimiento

Como se presento en la sección anterior, la ejecución de la ISR en repetidas ocasiones obtiene como resultado una función de de pares ordenados (y, t) la cual se muestra en la figura 4.1.10. y cuyos valores se almacenan en el arreglo Func[ ]. La función que se encuentra en este arreglo no representa el movimiento producido por el manejador y analizador de movimiento armónico, pero es útil para obtener los parámetros del movimiento así como la función que representa el movimiento real.

Los parámetros del movimiento armónico son amplitud y periodo. La función que se encuentra en el arreglo Func es armónica, es decir, Func[i] = Func[i+p]. Para obtener los parámetros de esta función debe observarse que:

$$Func[\,i\,].y = 0,\ Func[\,i+1\,].y = 1,\ Func[\,i+2\,].y = 2,\ \ldots\ ,Func[\,i+p\,].y = 0.$$

La función inicia con un valor $y = 0$, se incrementa hasta un máximo y regresa a $y = 0$ completando un ciclo. El valor máximo de ese ciclo es su amplitud. El periodo de ese ciclo está determinado por:

$$C[\,j\,].periodo = Func[\,i+p\,].t - Func[\,i\,].t$$

Se conoce que cada semiciclo del movimiento armónico simple producido por el manejador y analizador de movimiento armónico está descrito por la función:

$$Fsc = Func[i].signo * C[j].amplitud * sen(\theta_i) \quad para \quad \theta_i = 0.0,\ldots,\pi \quad (1)$$

En las ecuaciones anteriores así como en las sucesivas relacionadas con los parámetros del movimiento armónico y su función de movimiento, se considera a *i* como la i-ésima interrupción provocada por el puerto paralelo (i-ésimo punto de Func) y a *j* como el j-ésimo ciclo de Func. Después de relacionar la función producida por la ejecución de la ISR del puerto paralelo en repetidas ocasiones, con la función producida por el generador y analizador de movimiento armónico (*F*) se puede deducir que un ciclo de Func corresponde a medio ciclo (un semiciclo) de *F*, por lo que la amplitud de un ciclo de la funcion Func corresponde a la amplitud de un semiciclo de la función *F*, lo mismo que el periodo (*Fsc* es la Función de un SemiCiclo del generador y analizador de movimiento armónico). El signo del semiciclo de la función *Fsc* corresponde al parámetro Func[i].signo, el cual a su vez es un parámetro de la función que resulta de ejecutar repetidas veces la ISR del puerto paralelo.

Las consideraciónes anteriores, las cuales son útiles para determinar los parámetros de interés en el movimiento armónico, se encuentran implementadas en el código del programa en las líneas 294 a la 313 en el apéndice 2. En ese código se obtienen los parámetros periodo y amplitud de todos los semiciclos capturados con los cuales se puede armar la función *F* (Función del generador y analizador de movimiento armónico) para un rango de valores determinado.

4.1.7. Construcción de la función del movimiento

En la sección anterior se indica que cada semiciclo (medio ciclo) de la función *F* está determinado por:

$$Fsc = Func[i].signo * C[j].amplitud * sen(\theta_i) \quad para \quad \theta_i = 0.0,\ldots,\pi \quad (1)$$

También se han determinado los parámetros amplitud y periodo de cada semiciclo de la función *F* del movimiento registrado en un experimento dado. Con esos parámetros y conociendo la función *Fsc* sólo nos resta conocer el valor de θ_i.

Se tiene el conocimiento que θ_i es un parámetro de *Fsc*, sus unidades están expresadas en radianes y acotadas entre 0.0 y π, mientras que las unidades que deben hacerse corresponder a este dominio en la función Func son unidades de tiempo y están expresadas en segundos. La correspondencia entre estos dos sistemas de unidades puede establecerse determinando el tiempo t1, al inicio del j-ésimo ciclo, de tal forma que si consideramos la siguiente serie de puntos:

Func[i].y = 0, Func[i+1].y = 1, Func[i+2].y = 2, ... ,Func[i+p].y = 0.

t1 = Func[i].t

y cualquier θ_i puede ser determinada mediante la siguiente relación:

$$\theta_i = (Func[\, i+n \,].t - t1) * \pi / C[\, j \,].periodo \qquad i <= n < p$$

Con todo lo anterior es obvio concluir que la función *F* está formada por la concatenación de un determinado número de funciones *Fsc*, cada una con los parámetros amplitud y periodo posiblemente diferentes los cuales pueden ser extraídos a partir de la función Func.

La construcción de la función de movimiento está codificada entre las líneas 316 a la 336 en el apéndice 2, quedando almacenada en el arreglo *f* el cual se declara en la línea 161 cuya estructura se determina en las líneas 81 a la 85.

4.1.8. Despliegue de gráficas

El programa de captura y análisis de movimiento armónico, divide la pantalla en tres partes mediante las cuales interactúa con el usuario. Una de ellas la usa para mostrar las funciones que se están

realizando de acuerdo a los mandos que el usuario emite. Otra se usa exclusivamente para para el despliegue de gráficas y tabulaciones y la última para solicitudes de entrada por teclado o para despliegue de mensajes.

El area de interés en esta sección es particularmente la de despliegue de gráficas, la cual está acotada, por lo que cualquier cosa que sea necesario mostrar en esta área, debe ajustarse dentro de los límites de dichas acotaciones. Los puntos que acotan el área útil para graficar son: CGRAFX1, CGRAFY1, CGRAFX2 y CGRAFY2, y se definen en el programa en las líneas 30 a la 33.

Son cuatro cosas diferentes las que se grafican de forma alternada en esta área: amplitud de cada ciclo, periodo de cada ciclo, frecuencia de cada ciclo y la función de movimiento. Adicionalmente también se presenta una tabulación de amplitud, periodo y frecuencia de cada ciclo con respecto al tiempo.

Antes de mostrar cada gráfica debe prepararse el área útil, en la cual se dibuja una cuadrícula la cual muestra el plano coordenado. Para las gráficas de amplitud, periodo y frecuencia, el plano coordenado es el mismo, ya que a estos parámetros se le asignan valores absolutos, por lo que sólo se requiere el primer cuadrante (cuadrante positivo) del plano cartesiano. Para la gráfica de movimiento además del primer cuadrante, se usa también el cuarto, debido a que la función de movimiento contiene tanto valores positivos como negativos, para lo cual es necesario trazar un eje horizontal que divida el área útil en dos partes iguales, representando cada una un cuadrante. Todo lo anterior se encuentra implementado en el programa en las líneas 353 a la 411 del apéndice 2.

En todas las gráficas que se muestran, el eje coordenado horizontal representa el tiempo, en el cual el primer punto del eje es igual a cero y el último punto del eje es el tiempo total que duró el experimento en cuestión. El eje coordenado vertical representa el parámetro que se grafica: en el caso de periodo, representa tiempo en segundos, en el caso de frecuencia, representa ciclos por segundo, en el caso de amplitud representa milímetros y en el caso de la función de movimiento representa tambien milímetros. En todas las gráficas que se muestren, todos los pixeles que la representen deben caer dentro de los límites establecidos, para hacerlo es necesario determinar un factor de conversión de cada punto de cada gráfica a pixeles, dicho factor se determina dividiendo el máximo valor del área útil de la pantalla (pixeles) entre el

máximo valor de la gráfica (en sus unidades correspondientes). Estos factores (*factx* y *facty*) son determinados en las líneas 456 a la 459, previo a este cálculo en las líneas 424 a la 450 se encuentran los valores máximos de cada parámetro que se grafica (*maxx* y *maxy*). Cada valor de la gráfica debe multiplicarse por su factor correspondiente con el fin de que dicho pixel quede en en lugar apropiado dentro del los límites, de igual forma estos factores se le pasan a la función encargada de preparar el área de gráficado (líneas 353-411) para que etiquete las las partes en las que está dividido cada eje coordenado.

Por cada vez que se ejecuta la función Graficar( ) (líneas 412-499) se mostrará una grafica que puede corresponder a: amplitud, periodo, frecuencia o función de movimiento, tal decisión dependerá del valor que en ese instante tenga la variable global Pant.graf, cuya estructura se define en las líneas 107 a la 112 y se le pueden asignar los valores definidos en la línea 64.

4.2. Desarrollo de la interfaz de usuario

Debido a que entre los requerimentos del programa se incluye la necesidad de visualizar los datos mediante gráficas, la decisión de que la interfaz para el usuario sea en forma gráfica es una consecuencia obvia. Con el fin de que además de que sea funcional y amigable presente algún atractivo visible, se optó por diseñar la pantalla, con el auxilio de un programa especial para diseño gráfico. En la pantalla con la cual el usuario interactúa pueden notarse tres secciones: la sección de mandos, la sección de gráficas y la sección de mensajes.

Esta pantalla se salvaguarda en un archivo con formato gráfico PCX, el cual es dividido en cuatro archivos de igual magnitud, mediante un programa desarrollado especialmente para este propósito. Cada uno de estos cuatro archivos almacena un plano de bits, con el propósito de que posteriormente sea escrito directamente a la interfaz para gráficas VGA.

4.2.1. Diseño de la pantalla y archivos gráficos

Tal como acaba de mencionarse, son cuatro los archivos que almacenan la pantalla principal con la cual el usuario interactúa. La decisión de hacerlo de esta forma es porque la resolución geométrica deseable y además necesaria para el programa es 640x480 pixeles. Esta resolución nos permite también poder usar los tableros de caracteres gráficos definidos en la biblioteca de gráficas del compilador de TurboC. Por otro lado el sistema operativo MS-DOS, sobre el cual está desarrollado este programa funciona con el procesador en modo real, esto significa que la longitud de sus segmentos de memoria son fijos y acotados a 64Kbytes y la resolución tanto gráfica como de color empleada en el programa requiere un monto de memoria de 153600 bytes (640 * 480 / 2), por lo que no es posible almacenar este monto de memoria en un solo segmento, para posteriormente transferirse a memoria de video, además de que la memoria de video está dividida en páginas con longitud máxima de 64 Kbytes.

Una forma alternativa de transferir una imagen de memoria RAM a memoria de video, se logra cargando la imagen en varios segmentos de 64 Kbytes y transfiriendo el primer segmento a la página 0 de la memoria de video, conmutar la memoria de video a la página 2 y transferir el siguiente segmento y así sucesivamente hasta transferir la imágen completa. Sin embargo en el programa realizado se prefiere transferir cada plano de bits, desde el archivo directamente al plano de bits correspondiente en memoria de video, por la simplicidad y facilidad que representa hacerlo de esta forma.

Los archivos que forman los cuatro planos de bits son AMA.0 AMA.1, AMA.2 y AMA.3. El archivo con formato PCX (AMA.PCX) que se abre en el programa, se usa sólo para obtener la paleta que se carga en memoria de video y de donde se obtienen los colores que se usan a lo largo del programa. La función principal que inicializa el modo gráfico en video, forma la paleta y muestra la pantalla principal del programa, se llama InicializaGraficas( ) (línea 737 a 752). Como su nombre lo indica, esta función se encarga de poner el modo de video con una resolución geométrica y de color de 640x480 pixeles y 16 colores respectivamente, tiene como funciones utilitarias a LeePaleta( ), y LoadScreen16( ).

La función LeePaleta( ) (líneas 856 a la 866) abre el archivo AMA.PCX y del encabezado definido por la estructura PCXHEAD (Líneas 99 a la 166) obtiene la paleta la cual almacena en un arreglo el cual es

tomado como parámetro por la función PaletaPcx16( ) (líneas 892 a la 904), esta función forma cada uno delos 16 colores a partir de darle valores a tres variables que representan la cantidad de rojo, verde y azul. Estas 3 variables se le pasan como parámetro a la función ColorRgb( ) (líneas 709 a la 716) la cual con el uso de la función 10h de la interrupción 10h forma la paleta que usa el programa.

La función LoadScreen16( )(líneas 867 a la 885), Lee dirrectamente de cada uno de los archivos AMA.0, AMA.1, AMA.2 y AMA.3 la información que escribe directamente a memoria VGA (dirección absoluta A0000000h) en los planos 0, 1, 2 y 3. Para hacerlo, por cada plano que se va a escribir a memoria se indica en el grupo de registros secuenciadores de la tarjeta VGA (registro indice, dirección 3c4h) que de ese grupo se va accesar el segundo registro de datos por medio del cual se habilita o desabilita el plano de bits que se va a usar (enmascarando con '1' el bit que corresponde a la posición del plano). Una vez que el plano de bits correspondiente está habilitado se transfiere directamente desde archivo a la dirección A0000000h la información correspondiente a ese plano de bits.

4.2.2. Entrada de datos por teclado

Debido a que las funciónes scanf, getchar y anexas que son funciones de la biblioteca estándar del lenguaje C hacen eco en la terminal de salida estándar la cual funciona en modo de texto, no es posible usarlas en un programa que requiere que la visualización en pantalla sea en modo gráfico. Por este motivo fue necesario desarrollar funciones que obtuvieran la entrada de teclado, que validaran la entrada recibida de acuerdo al tipo de dato solicitado al usuario y desplegaran cada caracter recibido en modo gráfico.

Para detectar cada caracter tecleado por el usuario se usa la función GetKey( ) (líneas 717 a la 729). Esta función devuelve el caracter encontrado en el buffer de teclado sin hacer eco en la pantalla. La función GetIfKey( ) (líneas 730 a la 736) verifica si se ha presionado alguna tecla y devuelve su valor, de lo contrario regresa con '0'. Para discriminar los teclas pulsadas que corresponden a caracteres ASCII de los que corresponden a teclas especiales y de funciones se usa la función TeclaEsp( ) (líneas 905 a la 914).

Las funciones anteriormente descritas solamente transfieren un caracter desde el bufer de teclado a la variable que lo solicite, sin asignarle algun significado o función a la tecla encontrada. Estas funciones

sirven de soporte a la función LeeCadena( ) (líneas 763 a la 855). Esta función recibe como parámetros: la posición (x, y) donde se despliega cada caracter que el usuario teclee, un apuntador a la estructura reg y el color que se usa de fondo en la región donde se despliegan los caracteres tecleados. La estructura reg está definida en las líneas 93 a la 98 y en ella se depositan los valores capturados por la función LeeCadena convertidos a int, float, o char* según sea el caso. Los datos contenidos en la estructura REG tienen el siguiente significado:

- ent. Es un dato lógico. '0' significa que el dato capturado no es de tipo entero y '1' significa que si lo es.
- flot. Es un dato lógico. '0' significa que el dato capturado no es de tipo float y '1' significa que si lo es.
- alfa. Es un dato lógico. '0' significa que el dato capturado no es de tipo alfanumérico y '1' significa que si lo es.
- real. En caso de que flot = 1, en esta variable se almacena el número flotante capturado.
- entero. En caso de que ent = 1, en esta variable se almacena el número entero capturado
- c. En esta variable se almacena la cadena completa capturada.

Como puede observarse, la función LeeCadena( ) captura un conjunto de caracteres por teclado y en forma implícita provee la verificación automática de tipo de dato capturado. Para verificar el tipo de dato la función se auxilia del autómata de estado finito (Denning et al, 1978) definido por la tabla IV.II.I.

	número	letra	punto	+ | -	enter
0	2, número	3, letra	4, punto	1, + | -	6,W*
1	2, número	3, letra	4, punto	3, + | -	6,W
2	2, número	3, letra	4, punto	3, + | -	6,W
3	3, número	3, letra	3, punto	3, + | -	6,W
4	5, número	3, letra	3, punto	3, + | -	6,W-1**
5	5, número	3, letra	3, punto	3, + | -	6,W
6	0	0	0	0	0

Tabla IV.II.I. Autómata de transición asignada para la lectura de teclado.

Los asteriscos que se encuentran en la tabla IV.II.I. no forman parte del autómata, se usan como referencia a las consideraciones que a continuación se toman, y que son útiles para el descifrado de la cadena leida y su conversión al tipo de dato correspondiente.

1. El estado inicial es el estado 0 y el estado final es 6.
2. W = Entero, flotante o cadena. Representa el dato que regresa la función.
3. Si el autómata llegó al estado final procedente del estado 2 o del estado 4***, W = Entero.
4. Si el autómata llego al estado final procedente del estado 5, W = Flotante.
5. Si el autómata llegó al estado final procedente del estado 1 o del estado 3, W = cadena.
6. En el caso de leer BACKSPACE el autómata pasa al estado anterior y borra el último caracter leido. Para la implementación de este mecanismo, es necesario que el autómata tenga memoria de los estados (una pila LIFO).
7. * La función devuelve ENTER.
8. ** Le quita el último caracter a la cadena leida y devuelve W = Entero.
9. *** Si ha leido más de un caracter, en caso de haber leido dos el primero no debe ser + | -, De lo contrario es cadena.

La tabla IV.II.I. está codificada en las líneas 770 y 771 del apéndice 1. El autómata descrito en esta tabla empieza a funcionar en la línea 778. En esta línea el programa entra en un ciclo infinito (while(1)) en el cual cada tecla leida forza una iteración provocando que el autómata pase al estado determinado por la tabla IV.II.I. El estado en que se encuentra el autómata en una iteración cualquiera está determinado por la variable *i*, la historia de los estados en los cuales ha estado el autómata se registra en el arreglo *edo[ ]*, y la variable *cont* es un contador que indexa tanto el último caracter leído como también el último estado ocupado por el autómata.

El autómata está cambiando sus estados por cada tecla leida. Por cada iteración se verifica si el autómata se encuentra en su estado final (estado 6), una vez que la verificación anterior (línea 818) es

afirmativa, se procede a determinar la procedencia del estado que forzó al autómata a llegar a su estado final, con el propósito de convertir la cadena leída al tipo de dato que ella representa. Cabe hacer mención que el autómata garantiza que la cadena leída representa un tipo de dato sintáctica y semánticamente correcto.

Una vez que la cadena se ha convertido al tipo de dato correspondiente y se ha almacenado en el campo que le corresponde dentro de la estructura REG, el ciclo infinito se rompe mediante la instrucción return, lo cual a su vez provoca que la función LeeCadena( ) termine y regrese con un valor.

La función para lectura de teclado más completa, que a su vez hace uso de la función LeeCadena, es la función CapturaDato( ) (líneas 674 a la 708). A esta función se le pide que lea un dato de un tipo y de una longitud máxima determinada (int, float o cadena), y la función regresa el dato del tipo solicitado correctamente y además sin sobrepasar la longitud pedida. Esta función cumple con su petición con tal exactitud, que bien puede leer varias cadenas (cada cadena termina con un ENTER) y no terminar hasta que el tipo y la longitud de dato leído se ajusta al tipo de dato solicitado.

Emplear un autómata como se ha descrito anteriormente es un método que puede ser usado en otro tipo de problemas. Pueden usarse para control de fujo de un programa, para automatización de procesos, para verificación sintáctica etc.

5. Uso del programa

5.1. Descripción general y operación del programa

En este capítulo se describen las funciones del programa AMA (Analizador de Movimiento Armónico) y la forma como debe usarse. Aunque el programa por si solo es autoexplicativo, se incluyen estas instrucciones sólo para referencia ya que el usuario puede si así lo considera omitir leerlo.

El programa está compuesto por un archivo ejecutable y cinco archivos auxiliares, sin ellos el programa no funcionará adeccuadamente. El programa ejecutable se llama AMA_EXE y los archivos auxiliares son: AMA_0, AMA_1, AMA_2, AMA_3 y AMA_PCX. Todos estos archivos deben residir en el mismo directorio. Para usar el programa se requiere que la computadora esté conectada al generador y analizador de movimiento armónico, por medio del puerto paralelo mediante el cable que se proporciona para dicho propósito.

Considérese que los archivos anteriormente descritos se encuentran en la unidad C: en el directorio AMA. una vez posicionado en esa trayectoria, para dar inicio a la ejecución del programa debe darse el mando:

C:\AMA > AMA

Como se explicó en el capítulo anterior, la pantalla que a continuación muestra el programa está en modo gráfico, la cual se divide para propósitos de explicación en tres secciones:

1. Sección de mandos.

2. Sección de gráficas.

3. Sección de mensajes.

La figura 5.1. Muestra el área que la corresponde a cada una de estas tres secciones.

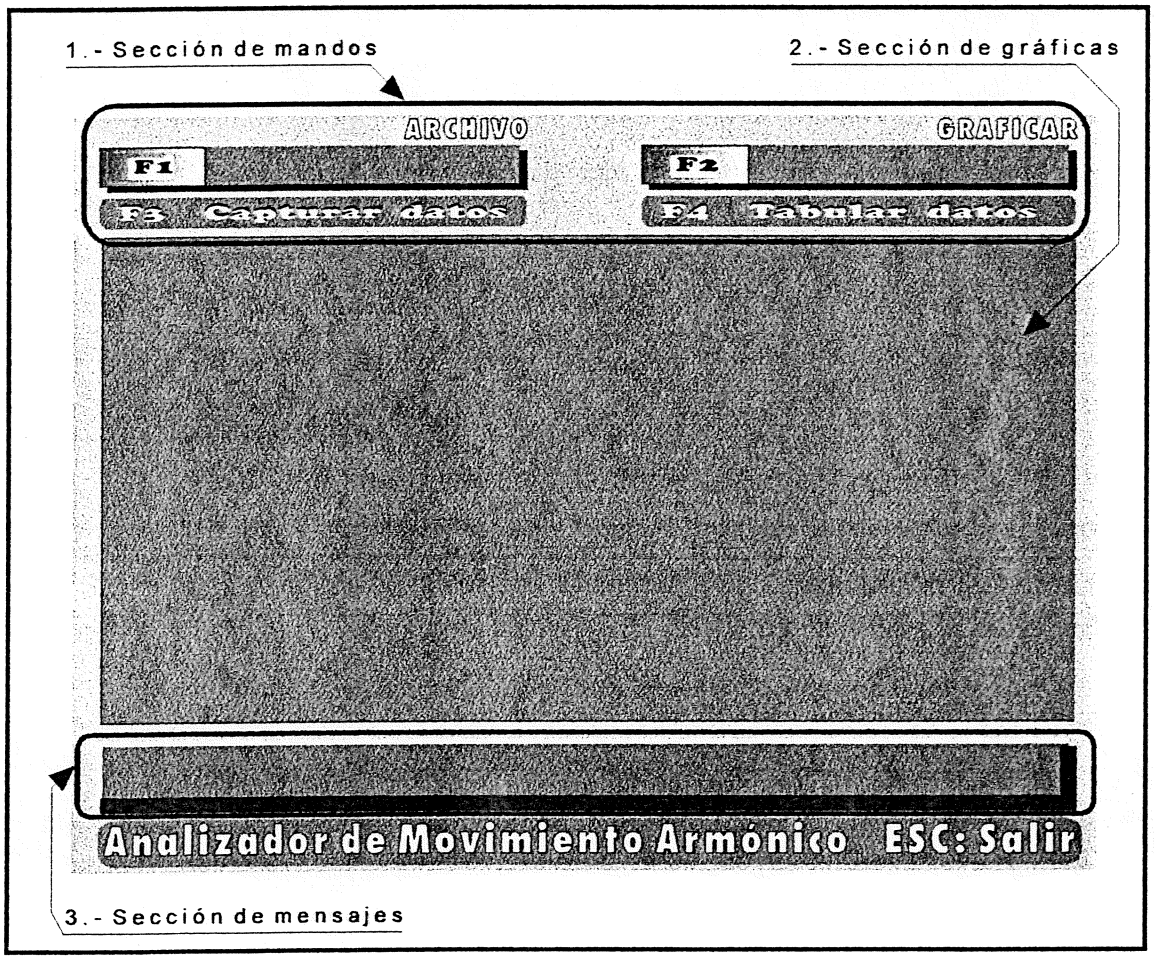


Fig. 5.1. Secciones de la pantalla del programa AMA.

Después de ejecutado el mando anterior aparece en el área de mensajes *"Dirección del puerto? [1:2:3]"* . Las posibles respuestas a esta pregunta son 1, 2 o 3, si se presiona <ENTER> por omisión se ofrece el 2 que normalmente es el más común en las computadoras.

En este punto, el programa está listo para recibir mandos por parte del usuario. En la sección de mandos de la pantalla se indica la forma y los posibles mandos que acepta el programa. Con la tecla de función *F1* se emiten mandos que tienen que ver con manejo de archivos. Con la tecla *F2* se dan mandos referentes a gráficas, las cuales aparecerán en la sección de gráficas. Con *F3* se capturan los datos que emite el generador y analizador de movimiento armónico, para lo cual se requiere que este aparato esté conectado y funcionando. Con la tecla *F4* se muestra una tabla con los parámetros que han sido previamente capturados por el programa (con *F3*), provenientes del generadorr y analizador de movimiento armónico.

5.1.1. Manejo de Archivos

Una vez que se han capturado los datos correspondientes a un experimento (con *F3*), es posible que sea necesario salvaguardarlos en un archivo ya sea para visualizar los resultados posteriormente o para ser analizados con alguna utilería de análisis matemático. Con ese propósito los datos capturados se usan para construír una función cuyo dominio es el tiempo (t) y el rango es la posición de la barra de masa del Manejador y Analizador de Movimiento Armónico. Esta función puede salvaguardarse en un archivo en formato ASCII.

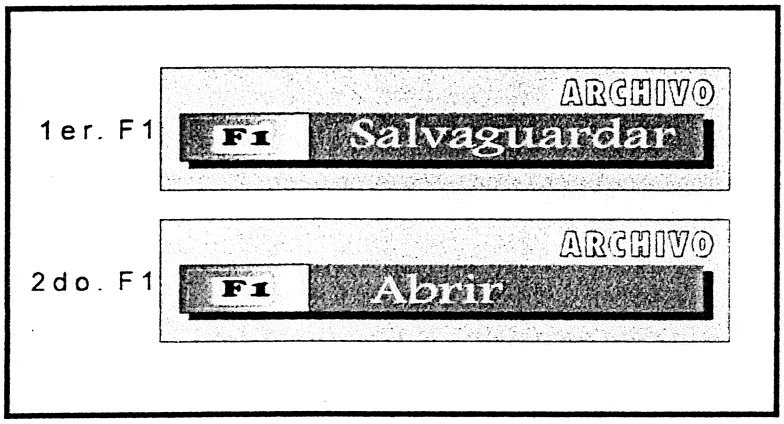


Fig. 5.1.1. Secuencia de mandos para manejo de archivos.

En la sección de mandos aparece un apartado con el encabezado "ARCHIVO" (Fig. 5.1.1). Los mandos necesarios para el manejo de archivos, pueden ser emitidos usando la tecla de funcion *F1*, tal como se indica en este apartado. Una vez que se presiona la tecla *F1* por primera vez, aparecerá en este mismo apartado la palabra "Salvaguardar"; si se presiona *F1* por segunda vez aparecerá en este apartado la palabra "Abrir", si se presiona *F1* otra vez aparecerá en este apartado la palabra "Salvaguardar" y así sucesivamente. Cada una de estas palabras que aparecen en este apartado indican la acción que se ejecutará una vez que se presione la tecla *<ENTER>*. En la ventana de mensajes aparecerán los errores o las peticiones que el programa necesite para ejecutar adecuadamente el mando emitido, tal como "Nombre de archivo?", "Archivo no existe" etc. La figura 5.1.1 muestra la secuencia de mandos por cada *F1* presionado.

5.1.2. Despliegue de gráficas

El programa provee la capacidad de graficar los parámetros característicos del movimiento armónico como lo son: Periodo, Amplitud y Frecuencia, además de que también puede graficar la función de movimiento. Estos parámetros pueden ser obtenidos por este mismo programa del

generador y analizador de movimiento armónico (con la tecla de función *F3*), o bien pueden obtenerse de un archivo que haya sido creado con este programa con anterioridad.

En la sección de mandos aparece un apartado con el encabezado "GRAFICAR" (Fig. 5.1.2). Los mandos necesarios para el despliegue de gráficas, pueden ser emitidos usando la tecla de funcion *F2*, tal como se indica en este apartado. Una vez que se presiona la tecla *F2* por primera vez, aparecerá en este mismo apartado la palabra "Periodo"; si se presiona *F2* por segunda vez aparecerá en este apartado la palabra "Frecuencia", si se presiona *F2* otra vez aparecerá en este apartado la palabra "Amplitud"... La secuencia completa de mandos que pueden emitirse se muestra en la figura 5.1.2.

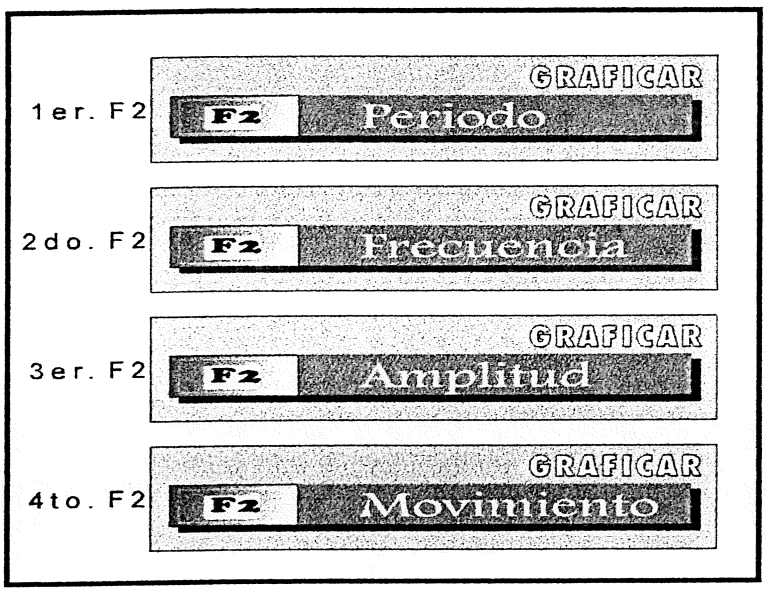


Fig.5.1.2. Secuencia de mandos para el despliegue de gráficas.

Cada una de estas palabras que aparecen en este apartado indican la gráfica que se desplegará una vez que se presione la tecla *<ENTER>*. En todas las gráficas se despliega el dato seleccionado con respecto al tiempo.

6. Sugerencias e ideas para el desarrollo de otras aplicaciones

El Desarrollo del presente trabajo no sólo arroja resultados útiles en la experimentación del movimiento armónico, si no también sirve como base para el desarrollo de otras aplicaciones que tienen que ver con fenómenos físicos en el campo de la mecánica. Todos los fenómenos físicos que involucran movimiento pueden ser medidos con mucha precisión usando los elementos que se involucran en este trabajo de tésis. Pueden construirse módulos de experimentación para comprobar las leyes de Newton, el movimiento de un péndulo; puede medirse la aceleración de un objeto en caída libre (aceleración producida por la fuerza gravitacional), El efecto de la energía cinética y muchos otros experimentos.

Un péndulo construido con un arco de plástico transparente, se le pueden dibujar barras delgadas semejantes a las que presenta la barra de masa del Manejador y analizador de movimiento armónico, y al igual que este aparato, se le pueden colocar fotodetectores, y el circuito electrónico usado en este trabajo y el mismo programa, tal vez con modificaciones muy ligeras al código, puede usarse para realizar experimentos que también involucran movimiento armónico.

Otro módulo de experimentación que basándose en el mismo principio y con los mismos elementos se presenta en la figura 6.1.

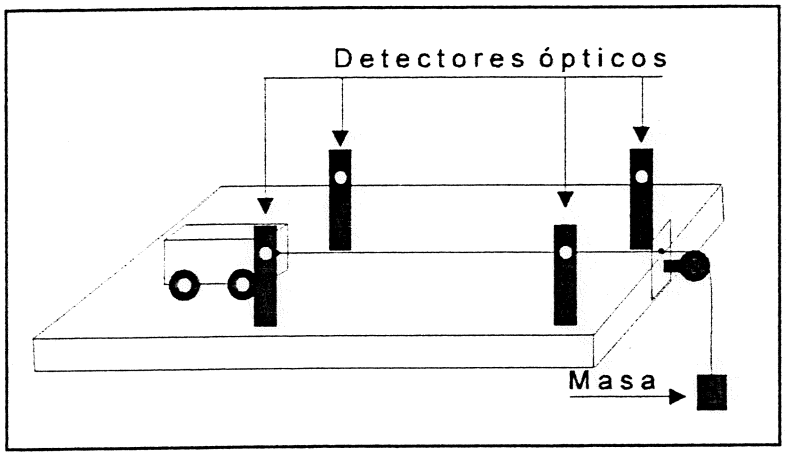


Fig. 6.1. módulo de experimentación para movimiento y fuerza.

El módulo de experimentación que se presenta en la figura 6.1., está formado por un móvil, el cual forzozamente debe ser opaco, un par de fotodetectores (cada uno formado por un diodo emisor de luz y un fototransistor), una cuerda delgada, y masa intercambiable.

El modelo de experimentación funciona de la siguiente forma:

Una vez que el módulo es energizado, ambos emisores de luz están encendidos y están siendo detectados por los fototransistores. El móvil opaco inicia su movimiento y al pasar a través del primer fotodetector provoca la luz del emisor no sea detectada por el fototransistor, lo cual genera una interrupción en la computadora. Esta interrupción indica a la computadora el inicio del conteo del tiempo.

El segundo emisor continúa encendido y detectado por el fototransistor, pero una vez que el móvil pase a través de él interrumpirá la detección de luz, y en consecuencia también a la computadora provocando el final de conteo del tiempo.

Conociendo la masa, la distancia recorrida por el móvil y el tiempo empleado en recorrer esa distancia puede determinarse la aceleración alcanzada. Este modelo sirve para determinar la

aceleración en forma indirecta (con el uso de las fórmulas $F = ma$, $F = mg$). Pero si lo que se desea es medir la aceleración experimentada, puede colocarse una barra semejante a la barra de masa del manejador y analizador de movimiento armónico en vez del móvil opaco.

El módulo descrito anteriormente con unas modificaciones a su estructura puede también ser usado para experimentos de caída libre, conteo de objetos que pasan a través de una banda transportadora, medir la frecuencia con la cual pasan esos objetos y realizar estadísticas útiles en la planeación de la producción.

Usando un poco de ingenio e imaginación pueden construirse diferentes aplicaciones basadas todas en el principio motivo de este trabajo. Tales aplicaciones no están amarradas a la experimentación científica, sino que también tienen que ver con automatización de procesos industriales, mediciones, retroalimentación en sistemas de control y en todo lo que tenga que ver con movimiento y medición del tiempo.

6. Conclusiones

1. En el procesamiento de datos en tiempo real, es necesario que tanto el software como el hardware involucrado en la aplicación, garanticen que los datos de entrada van a ser capturados sin pérdida, o cuando menos que los datos que no puedan ser capturados puedan ser estimados mediante métodos estadísticos. Existen dos métodos de captura de datos en tiempo real: Polling y por interrupción. La elección de cualquiera de estos métodos depende de los requerimientos de la aplicación.

2. Cuando la aplicación de tiempo real involucra salida de datos, ya sea para controlar aparatos externos a la computadora, el tiempo con el cual la computadora entrega estos datos también es crítico, ya que de ello depende que los aparatos que se controlan funcionen adecuadamente.

3. En el desarrollo de aplicaciones de tiempo real y aplicaciones que tienen que ver con entrada y salida de datos, es necesario conocer en detalle la arquitectura de la computadora que se va a usar. Los elementos que se involucran en la arquitectura de una computadora que deben conocerse son: memoria, procesador, dispositivos de acceso directo a memoria (DMA), contadores de tiempo de la computadora, y los dispositivos de entrada salida que se van a usar. El propósito de conocer estos elementos es para saber programarlos y buscar las alternativas de empleo de ellos que aporten mayor provecho a la aplicación que se va a desarrollar.

4. En el desarrollo de aplicaciones en tiempo real debe evitarse usar sistemas operativos que hacen uso intenso del procesador provocando sobrecargo del mismo. en este tipo de

aplicaciones lo que se desea es que el procesador este casi totalmente dedicado al procesamiento de los datos en tiempo real y no tan dedicado a las operaciones propias del sistema operativo. En algunas aplicaciones es tan fuerte este requerimiento que se evita por completo el uso del sistema operativo y se desarrolla la programación considerando al sistema como de propósito específico.

5. Los elementos involucrados en el modelo físico que sirvió de base para el desarrollo de este trabajo, tanto de software como de harware, pueden servir de base para el desarrollo de otras aplicaciones útiles en la experimentación científica, como también en la automatización de procesos industriales, ya que los principios físicos que usa el manejador y analizador de movimiento armónico para medir el movimiento, pueden ser utilizados mediante ligeras modificaciones, en todo tipo de aplicaciones que se relacionan con medición y registro de tiempo y movimiento, parámetros que son abundantes en el mundo real.

Bibliografía

1. Denning J. Peter, Dennis B. Jack, Qualits E. Joseph. Machines, Languages, and Computation. Prentice Hall Englewood Cliffs, New Jersey 1978. 92-136 pp.

2. Ferraro F. Richard. Programers Guide to the EGA, VGA and SuperVGA Cards. Adison-Wesley. 2nd. Ed. Wlmington, Delaware, U.S.A. 1994. 25, 160-165, 269, 270, 328 pp.

3. Foster O. John. BIOSMAKER. Anabooks. U.S.A. 269 pp.

4. Foster O. John, Choisser P. John. The XT-AT Handbook For Engineers, Programers and other Serious PC/XT and PC/AT Users. 6th Ed. Anabooks. U.S.A. 48, 64 pp.

5. INTEL Corporation. Intel486, Microprocessor Family. Programmer's Reference Manual. U.S.A. 1992. 7-1, 9-1 – 9-27 pp.

6. INTEL Corporation. Peripheral Components. U.S.A. 1996. 5-378 – 5-394 pp.

7. MOTOROLA Inc. MOTOROLA. 8-bits Microprocessor & Peripheral Data. U.S.A. 1983. 3-697 – 3-718 pp.

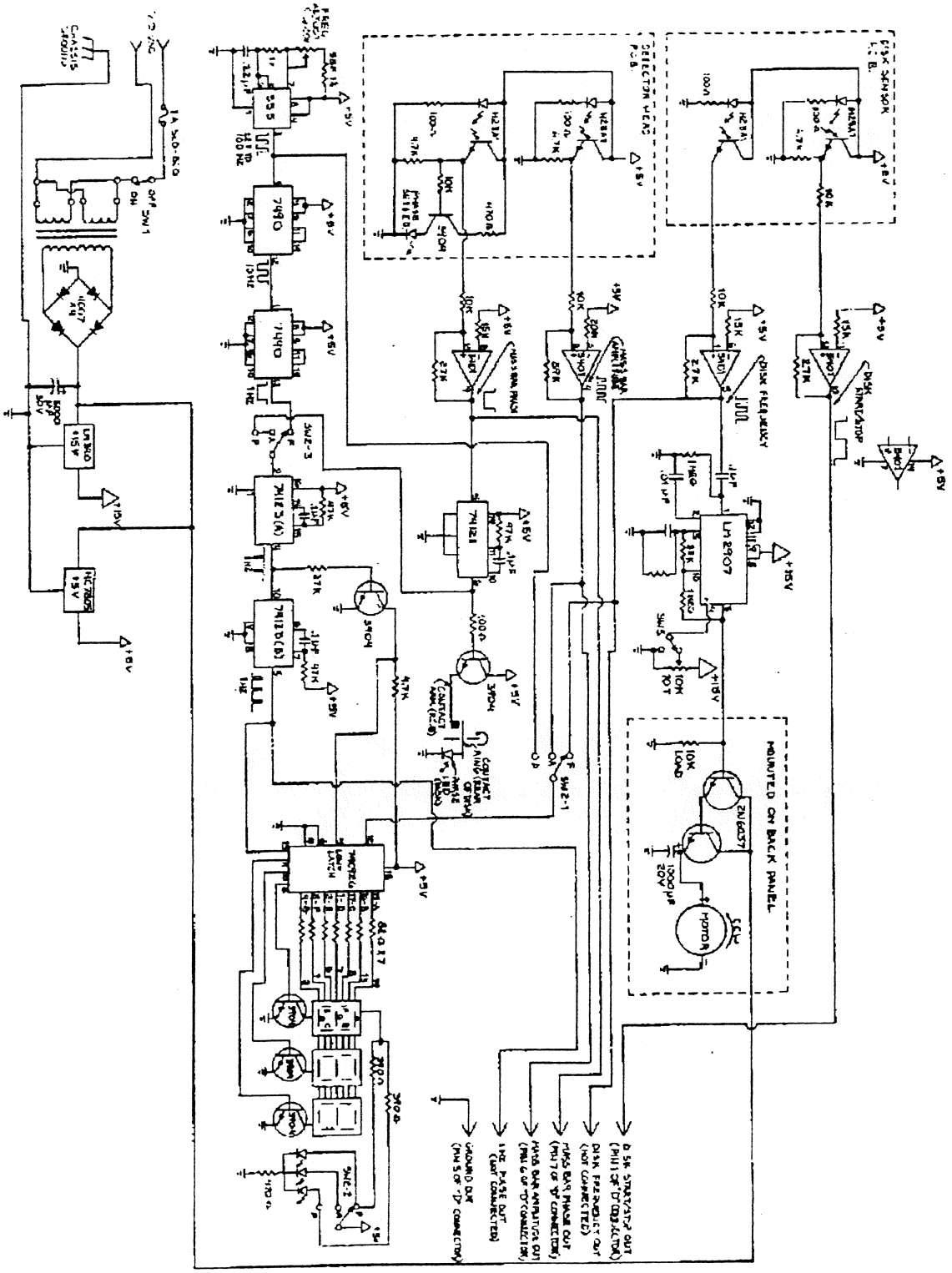
8. NEC Electronics Inc. NEC. Microcomputer Products. Microprocessors, Periferals, & DSP Products. U.S.A. 1987. 8-57 – 8-68 pp.

9. PASCO Model 9210A Driven Harmonic Motion Analizer. Operation Manual. 2-10, 27, 28 pp.

10. Piskunov, Nicolai S. Cálculo diferencial e integral. Ed. Mir, Moscú, 6th Ed. 1983. pp 76-106.

11. Tanenbaum S. Andrew, Woodhull S. Albert. Operating Systems, Design and Implementation. Ed. Prentice Hall , 2nd. Edition 1997.

Apéndice 1



Esquemático del generador y analizador de movimiento armónico

Apéndice 2

```
1. #include <graphics.h>
2. #include <stdio.h>
3. #include <dos.h>
4. #include <bios.h>
5. #include <math.h>
6. #include <stdlib.h>
7. #include <string.h>

8. #define CANT_CICLOS  30
9. #define MAXPTS       1000
10.#define PI_X_2       2.0*M_PI

11.#define fase(A)    ((A) & 0x80)? 1 : -1
12.#define disco(A)   ((A) & 0x20)? 1 : 0  /* Este macro no lo uso en
   esta versión */
13.#define mayor(x, y)  (x) > (y)? (x) : (y)

14.#define ARCHIVO_PCX  "ama.pcx"
15.#define ARCHIVO_VGA  "ama"

16.#define MAXCAR     20
17.#define LONG_PLANO   38400L
18.#define ABIERTO    1
19.#define CERRADO    0
20.#define ARCHIVO    1
21.#define GRAFICA    2

22.#define ARCH_X1    95
23.#define ARCH_Y1    25
24.#define ARCH_X2    280
25.#define ARCH_Y2    47

26.#define GRAF_X1    430
```

```
 27.#define GRAF_Y1    25
28.#define GRAF_X2    610
29.#define GRAF_Y2    47

30.#define CGRAFX1    30
31.#define CGRAFY1    81
32.#define CGRAFX2    618
33.#define CGRAFY2    379

34.#define MENS_X1    35
35.#define MENS_Y1    392
36.#define MENS_X2    618
37.#define MENS_Y2    430

38.#define CTR_C    0x03
39.#define BKSPCE   0x08
40.#define TAB      0x09
41.#define ENTER    0x0d
42.#define ESC   0x1b
43.#define F1    0x13b
44.#define F2    0x13c
45.#define F3    0x13d
46.#define F4    0x13e
47.#define F5    0x13f
48.#define F6    0x140
49.#define F7    0x141
50.#define F8    0x142
51.#define F9    0x143
52.#define F10   0x144
53.#define HOME  0x147
54.#define PGUP  0x149
55.#define PGDN  0x151
56.#define INS   0x152
57.#define DEL   0x153
58.#define END   0x14f
59.#define UP    0x148
60.#define DOWN  0x150
```

```
61.#define IZQ   0x14b
62.#define DER   0x14d

63.enum{ ABRE, SALVA };
64.enum{ MOV, PERIOD,  FREC, AMP };
65.enum{ NEGRO,    ROJO_OSC, VERDE_OSC,  CAFE,   AZUL_OSC,
66.     VIOLETA_OSC,  AZUL_GRIS,  GRIS_OSC,   GRIS, ROJO,
67.     VERDE, AMARILLO, AZUL_MAR,   VIOLETA,AZUL, BLANCO };

68.int COLOR16[16]={0,1,2,3,4,5,20,7,56,57,58,59,60,61,62,63} ;
69.char far *MemSVGA = (char far *) 0xA0000000L ;

70.typedef struct
71.{
72.     int    amplitud;
73.     float  periodo;
74.     float  t;
75.}AMA;           /* analizador de movimiento armónico */

76.typedef struct
77.{
78.     int  y,signo;   /* signo de la ordenada y */
79.     long t;
80.}PAR_ORD;       /* pares ordenados (t,y) */

81.typedef struct
82.{
83.     float  y;
84.     float  t;
85.}FUNCION;

86.typedef struct
87.{
88.     int cant_pts;
89.     int cant_ciclos;
90.}CANTS;
```

```
 91.typedef unsigned int WORD;
92.typedef unsigned char UCHAR;
93.typedef struct
94.{
95.     int     ent, flot, alfa, entero;
96.     float   real;
97.     char    c[MAXCAR];
98.}REG;

99.typedef struct
100.{
101.    char    manufacturer, version, encoding, bits_per_pixel;
102.    int     xmin, ymin, xmax, ymax, hres, vres;
103.    char    palette[48], reserved, colour_planes;
104.    int     bytes_per_line, palette_type;
105.    char    filler[58];
106.}PCXHEAD;

107.typedef struct
108.{
109.    char arch;
110.    char graf;
111.    char nom_arch[30];
112.} PANTALLA;

113./* variables Analizador de Movimiento Armónico (AMA) */
114.unsigned int Puerto;
115.PAR_ORD Func[MAXPTS];
116.CANTS   Cant;
117.long    Tiempo;

118.void interrupt (*anterior1C)(void);
119.void interrupt (*anteriorParalelo)(void);

120./* Variables para manejo de graficas */
121.PANTALLA  Pant;
122.char      Edo_Arch = CERRADO;
```

```
123./* funciones del Analizador de Movimiento Armónico (AMA) */
124.void  interrupt CapParalelo(void);
125.void  Capturar( AMA *ciclo, FUNCION *fdet );
126.void  ConfPuerto( void );
127.void  DesabilitaParalelo(void);
128.void  FormaCiclos( AMA *ciclo );
129.void  FormaFuncion( AMA *ciclo, FUNCION *f );
130.void  FormaSierra( FUNCION *f );
131.void  GrafCuadricula( int factx, int facty );
132.void  Graficar( AMA *ciclo, FUNCION *f );
133.void  interrupt Int1C(void);
134.void  IntTimer(void);
135.void  Paralelo(void);
136.void  Tabular( AMA *ciclo );
137.void  Tiemponts(unsigned char hi, unsigned char lo);

138./* Funciones utilitarias y de graficas */
139.int   AbrirArch( FUNCION *f );
140.void  ActualizaPantalla( char que );
141.WORD  CapturaDato( int x1, int y1,char *dato, char tipo,
142.         int tam, REG *reg );
143.void  ColorRgb(int re,int gr,int bl,int reg ) ;
144.WORD  GetKey( void );
145.WORD  GetIfKey( void );
146.int   InicializaGraficas(void) ;
147.void  InicializaParametros( void );
148.WORD  LeeCadena( int ctx1, int cty1, REG *reg, int bkcolor );
149.int   LeePaleta(char *archivo_pcx) ;
150.void  LoadScrn16( char* ) ;
151.WORD  May( WORD );
152.void  MensajeQ( char *men ) ;
153.void  PaletaPcx16( char *p ) ;
154.WORD  TeclaEsp( WORD c );
155.void  Salvaguarda( FUNCION *f );

156.void main(void)
```

```
157.{
158. int       i, que;
159. WORD    tecla;
160. AMA       ama[CANT_CICLOS];
161. FUNCION*fdet;

162. InicializaGraficas();
163. InicializaParametros();
164. ConfPuerto();
165. if( (fdet = (FUNCION * ) malloc( sizeof(FUNCION)*MAXPTS ) ) == NULL
          )
166. {
167.    closegraph();
168.    printf( "No me alcanzó la memoria" );
169. }
170. do{
171.    tecla = GetKey();
172.    switch( tecla )
173.    {
174.      case F1: ActualizaPantalla( que = ARCHIVO ); break;

175.      case F2: ActualizaPantalla( que = GRAFICA ); break;

176.      case F3: Capturar( ama, fdet ); break;

177.      case F4: Tabular( ama ); break;

178.      case ENTER:
179.        if( que == ARCHIVO )
180.        {
181.          if( Pant.arch == ABRE )
182.          {
183.            if( AbrirArch( fdet ) );
184.               FormaSierra( fdet );
185.            FormaCiclos( ama );
186.          }
187.          if( Pant.arch == SALVA )
```

```
188.              Salvaguarda( fdet );
189.         }
190.         if( que == GRAFICA )
191.           Graficar( ama, fdet );
192.         break;
193.    }
194. }while( tecla != ESC );
195. free( fdet );
196. for( i=0; i<16; i++)
197. {
198.    setfillstyle( SOLID_FILL, i);
199.    bar(10*(i+1),200, 10*(i+2), 220 );
200. }
201. tecla = GetKey();
202. closegraph() ;
203.}

204./*-----------------------------------------------------------------
205.Funciones del Analizador de Movimiento Armónico
206.-----------------------------------------------------------------*/
207.void interrupt CapParalelo(void)
208.{
209. Func[Cant.cant_pts].signo = fase( inportb(Puerto+1) );
210. if( Func[Cant.cant_pts].signo != Func[Cant.cant_pts-1].signo )
211.    Func[Cant.cant_pts].y = 0;
212. else
213.    Func[Cant.cant_pts].y = Func[Cant.cant_pts-1].y + 1;
214. Func[Cant.cant_pts].t = Tiempo;
215. Cant.cant_pts++;

216. outportb(0x20,0x20);
217.}

218.void Capturar( AMA *ciclo, FUNCION *fdet )
219.{
220. char valor;
```

```
221. Tiempo = 0;
222. Cant.cant_pts = 1;
223. Cant.cant_ciclos = 0;
224. Func[0].y = 0;
225. Func[0].t = 0;
226. Func[0].signo = 0;

227. /* espera para leer los datos del Analizador de Movimiento Armónico
     */
228. bar(MENS_X1, MENS_Y1, MENS_X2, MENS_Y2 );
229. outtextxy( MENS_X1+10, MENS_Y1+15, "Capturando . . ." );
230. Tiemponts( 0x17, 0x4D ) ; /* 200 ticks por seg */
231. IntTimer();
232. Paralelo();
233. while ( Cant.cant_pts < MAXPTS && !GetIfKey() )
234.    if(Tiempo > 600 && Cant.cant_pts<10 )
235.    {
236.      bar(MENS_X1, MENS_Y1, MENS_X2, MENS_Y2 );
237.      settextstyle( TRIPLEX_FONT, HORIZ_DIR, 1 );
238.      outtextxy( MENS_X1+10, MENS_Y1,
239.      "Analizador de movimiento armonico fuera de linea" );
240.      outtextxy( MENS_X1+10, MENS_Y1+18,
241.      "o el puerto paralelo que se indica esta desactivado" );
242.      GetKey();
243.      closegraph();
244.      DesabilitaParalelo();
245.      setvect(0x1c, anterior1C);
246.      outportb(0x43, valor) ;
247.      exit(1);
248.    }
249. DesabilitaParalelo();
250. Tiemponts( 0xFF, 0xFF ) ; /* 19.2 ticks por seg */
251. setvect(0x1c, anterior1C);
252. outportb(0x43, valor) ;

253. bar(MENS_X1, MENS_Y1, MENS_X2, MENS_Y2 );
254. FormaCiclos( ciclo );
```

```
 255. FormaFuncion( ciclo, fdet );
256. settextstyle( TRIPLEX_FONT, HORIZ_DIR, 2 );
257.}

258.void ConfPuerto( void )
259.{
260. REG     dev;

261. Puerto = 0;

262. while(1)
263. {
264.    bar(MENS_X1, MENS_Y1, MENS_X2, MENS_Y2 );
265.    outtextxy( MENS_X1+10, MENS_Y1+10, "Direccion del puerto?
   [1;2;3]" );
266.    while( Puerto < 1 || Puerto > 3 )
267.    {
268.       CapturaDato( MENS_X1+300, MENS_Y1+10, "2", 1, 1, &dev );
269.       if( dev.entero )
270.          Puerto = dev.entero;
271.       else
272.          Puerto = 2;
273.    }
274.    switch(Puerto){
275.       case 1: Puerto = 0x0278; break;
276.       case 2: Puerto = 0x0378; break;
277.       case 3: Puerto = 0x03bc; break;
278.    }
279.    break;
280. }
281. bar(MENS_X1, MENS_Y1, MENS_X2, MENS_Y2 );
282.}

283.void DesabilitaParalelo(void)
284.{
285. /*desabilitar interrupción en el puerto paralelo */
286. outportb(Puerto+2, inportb(Puerto+2) & 0xef);
```

```
287. /*desabilitar la máscara de interrupción en el 8259 */
288. outportb(0x21, inportb(0x21) | 0x80);

289. /*desabilitar interrupciones */
290. disable();
291. setvect(0xf, anteriorParalelo);
292. enable();
293.}

294.void FormaCiclos( AMA *ciclo )
295.{
296. int    i=1, j=0, maxpts=Cant.cant_pts-1;
297. float  t1, t2;

298. while( i < Cant.cant_pts && j < CANT_CICLOS )
299. {
300.    t1 = (float)Func[i-1].t / 200.0;
301.    while( Func[i].y > 0 && i < maxpts )
302.    {
303.       i++;
304.       if( i == Cant.cant_pts )
305.          Func[i].y = 0;   /* Para protegerse */
306.    }
307.    t2 = (float)Func[i].t / 200.0;

308.    ciclo[j].periodo = t2-t1;
309.    ciclo[j].amplitud = Func[i-1].signo * Func[i-1].y;
310.    ciclo[j].t = t2;
311.    i++;
312.    j++;
313. }
314. Cant.cant_ciclos = j;
315.}

316.void FormaFuncion( AMA *ciclo, FUNCION *f )
317.{
```

```
318. int    i, j=0;
319. float  teta, t, t_ant=0.0;

320. for( i=0; i<Cant.cant_pts && j<Cant.cant_ciclos; i++ )
321. {
322.    if( ciclo[j].periodo == 0.0 )
323.    ciclo[j].periodo = 0.00001;  /* Para protegerse de división por
   cero */
324.    t = (float)Func[i].t / 200.0;
325.    teta = (t-t_ant) * M_PI / ciclo[j].periodo;
326.    f[i].y = ciclo[j].amplitud * sin(teta);
327.    f[i].t = t;
328.    if( Func[i].y == 0 && i > 0 )
329.    {
330.       t_ant = (float)ciclo[j].t;
331.       j+;
332.    }
333. }
334. Cant.cant_pts = i; /* La función es válida hasta este punto */
335. Edo_Arch = ABIERTO;/* Por si se desea salvaguardar */
336.}

337.void FormaSierra( FUNCION *f )
338.{
339. int  i;

340. for( i=1; i<Cant.cant_pts; i++ )
341. {
342.    if( f[i].y > 0 )
343.       Func[i].signo = 1;
344.    else
345.       Func[i].signo = -1;
346.    if( Func[i].signo != Func[i-1].signo )
347.       Func[i].y = 0;
348.    else
349.       Func[i].y = Func[i-1].y + 1;
350.    Func[i].t = (long)( f[i].t * 200.0 );
```

```
 351. }
352.}

353.void GrafCuadricula( int factx, int facty )
354.{
355. int    i, mitad;
356. char   cad[10];
357. float  coord;

358. /* dibuja líneas verticales */
359. for(i=CGRAFX1; i<CGRAFX2; i+=50)
360. {
361.    setcolor( AZUL_OSC );
362.    line( i, CGRAFY1+10, i, CGRAFY2-20 );
363.    coord = (float)(i-CGRAFX1) / (float)factx;
364.    sprintf( cad, "%-5.2f", coord );
365.    setcolor( GRIS );
366.    outtextxy( i+1, CGRAFY2-8, cad );
367. }
368. outtextxy( (CGRAFX2-CGRAFX1)/2-20, CGRAFY2-21, "Tiempo" );

369. /* Para protegerse de division por cero */
370. if( facty == 0 )
371.    facty = 1;
372. if( factx == 0 )
373.    factx = 1;

374. if( Pant.graf == MOV )
375. {
376.    mitad = CGRAFY1 + (CGRAFY2 - CGRAFY1) / 2;
377.    /* Dibuja líneas horizontales desde 0 hasta maxy */
378.    for(i=mitad; i>CGRAFY1; i-=50)
379.    {
380.       setcolor( AZUL_OSC );
381.       line( CGRAFX1+10, i, CGRAFX2, i );
382.       coord = (float)(mitad-i) / (float)facty;
383.       sprintf( cad, "%-5.2f", coord );
```

```
 384.       setcolor( GRIS );
385.       outtextxy( CGRAFX1+10, i+3, cad );
386.    }

387.    /* Dibuja líneas horizontales desde 0 hasta -maxy */
388.    for(i=mitad; i<CGRAFY2; i+=50)
389.    {
390.       setcolor( AZUL_OSC );
391.       line( CGRAFX1+10, i, CGRAFX2, i );
392.       coord = (float)(mitad-i) / (float)facty;
393.       sprintf( cad, "%-5.2f", coord );
394.       setcolor( GRIS );
395.       outtextxy( CGRAFX1+10, i-8, cad );
396.    }
397.    line( CGRAFX1+2, mitad, CGRAFX2-2, mitad );
398. }
399. else
400.    for(i=CGRAFY2; i>CGRAFY1; i-=50)
401.    {
402.       setcolor( AZUL_OSC );
403.       line( CGRAFX1+10, i, CGRAFX2, i );
404.       coord = (float)(CGRAFY2-i) / (float)facty;
405.       sprintf( cad, "%-5.2f", coord );
406.       setcolor( GRIS );
407.       if( coord > 0.0 )
408.       outtextxy( CGRAFX1+10, i-8, cad );
409.    }
410.    line( CGRAFX1+10, CGRAFY2, CGRAFX2-2, CGRAFY2 );
411.}

412.void Graficar( AMA *ciclo, FUNCION *f )
413.{
414. float  maxx=0.0001, maxy=0.0001, den;
415. float  ampl[CANT_CICLOS], period[CANT_CICLOS], frec[CANT_CICLOS];
416. int    factx, facty, mitad, i;

417. if( Cant.cant_ciclos < 2  )
```

```
418. {
419.    outtextxy( MENS_X1+10, MENS_Y1+10, "No hay datos por graficar"
   );
420.    GetKey();
421.    bar(MENS_X1, MENS_Y1, MENS_X2, MENS_Y2 );
422.    return;
423. }

424. if( Pant.graf == MOV )
425. {
426.    for( i=0; i<Cant.cant_ciclos; i++ )
427.       maxy = mayor( maxy, fabs(ciclo[i].amplitud) );
428. }
429. for( i=0; i<Cant.cant_ciclos; i+=2 )
430. {
431.    switch( Pant.graf )
432.    {
433.       case AMP:
434.          ampl[i] = fabs(ciclo[i].amplitud) +
     fabs(ciclo[i+1].amplitud);
435.          maxy = mayor( maxy, ampl[i] );
436.          break;
437.       case PERIOD:
438.          period[i] = ciclo[i].periodo + ciclo[i+1].periodo;
439.          maxy = mayor( maxy, period[i] );
440.          break;
441.       case FREC:
442.          den = ciclo[i].periodo + ciclo[i+1].periodo;
443.          if( den == 0.0 )
444.             den = 0.01;
445.          frec[i] = 1 / den;
446.          maxy = mayor( maxy, frec[i] );
447.          break;
448.    }
449. }
450. maxx = ciclo[Cant.cant_ciclos-1].t;
```

```
451. /* condiciones para protegerse de división por cero */
452. while( maxx == 0.0 )
453.    maxx = ciclo[--Cant.cant_ciclos].t;
454. if( maxy == 0.0 )
455.    maxy = 1.0;

456. factx = (CGRAFX2-CGRAFX1) / ceil(maxx);
457. facty = (CGRAFY2-CGRAFY1) / ceil(maxy);
458. if( Pant.graf == MOV )
459.    facty = (CGRAFY2-CGRAFY1) / ( ceil(maxy)*2 );

460. settextstyle( DEFAULT_FONT, HORIZ_DIR, 1 );
461. GrafCuadricula( factx, facty );
462. mitad = CGRAFY1 + (CGRAFY2 - CGRAFY1) / 2;

463. settextstyle( DEFAULT_FONT, VERT_DIR, 1 );
464. switch( Pant.graf )
465. {
466.    case MOV:
467.       outtextxy( CGRAFX1+9, mitad-20, "F(Tiempo)" );
468.       setcolor( AMARILLO );
469.       for(i=1; i<Cant.cant_pts; i++ )
470.       line(  f[i-1].t*factx+CGRAFX1, mitad-f[i-1].y*facty,
471.            f[i].t*factx+CGRAFX1, mitad-f[i].y*facty );
472.       break;
473.    case AMP:
474.       outtextxy( CGRAFX1+9, mitad-20, "Amplitud" );
475.       setcolor( AMARILLO );
476.       for(i=2; i<Cant.cant_ciclos; i+=2 )
477.         line(  ciclo[i-2].t*factx+CGRAFX1, CGRAFY2-ampl[i-2]*facty,
478.           ciclo[i].t*factx+CGRAFX1, CGRAFY2-ampl[i]*facty );
479.       break;
480.    case PERIOD:
481.       outtextxy( CGRAFX1+9, mitad-20, "Periodo" );
482.       setcolor( AMARILLO );
483.       for(i=2; i<Cant.cant_ciclos; i+=2 )
```

```
484.        line(  ciclo[i-2].t*factx+CGRAFX1, CGRAFY2-period[i-
  2]*facty, ciclo[i].t*factx+CGRAFX1, CGRAFY2- period[i]*facty );
485.        break;
486.    case FREC:
487.      outtextxy( CGRAFX1+9, mitad-20, "Frecuencia" );
488.      setcolor( AMARILLO );
489.      for(i=2; i<Cant.cant_ciclos; i+=2 )
490.        line(  ciclo[i-2].t*factx+CGRAFX1, CGRAFY2-frec[i-2]*facty,
491.             ciclo[i].t*factx+CGRAFX1, CGRAFY2- frec[i]*facty );
492.      break;
493. }

494. GetKey();
495. setfillstyle( SOLID_FILL, NEGRO );
496. bar(CGRAFX1, CGRAFY1, CGRAFX2, CGRAFY2 );
497. settextstyle( TRIPLEX_FONT, HORIZ_DIR, 2 );□
498. setfillstyle( SOLID_FILL, AZUL_GRIS );
499.}

500.void interrupt Int1C(void)
501.{
502. Tiempo++;
503. outportb(0x20,0x20);
504.}

505.void IntTimer(void)
506.{
507. disable();
508. anterior1C = getvect(0x1C);
509. setvect(0x1C,Int1C);
510. enable();
511.}

512.void Paralelo(void)
513.{
514. /*habilitar interrupción en el puerto paralelo  (bit de enable) */
515. outportb(Puerto+2,inportb(Puerto+2) | 0x10);
```

```
516. /* habilitar la máscara de interrupción en el 8259 */
517. outportb(0x21,inportb(0x21) & 0x7f);

518. /* asignar ISR rutina de servicio de interrupción */
519. disable();
520. anteriorParalelo = getvect(0xf);
521. setvect(0xf,CapParalelo);
522. enable();
523.}

524. /*   Provoca que el timer interrumpa a 200 ticks por segundo
525. devuelve el valor del puerto 43h para restablecerlo al final */
526. void Tiemponts(unsigned char hi, unsigned char lo)
527.{
528. char valor;

529. outportb(0x43, 0x34) ;
530. outportb(0x40, lo) ; /* 5965 en hexadecimal LSB-MSB */
531. outportb(0x40, hi) ;
532.}

533.void  Tabular( AMA *ciclo )
534.{
535. int     i, limi=0;
536. char    num[8];
537. float   ampl[CANT_CICLOS], period[CANT_CICLOS], frec[CANT_CICLOS],
   den;

538. settextstyle( DEFAULT_FONT, HORIZ_DIR, 1 );
539. outtextxy( CGRAFX1+40, CGRAFY1+10, "Tiempo  Amplitud       Periodo
   Frecuencia" );
540. for( i=0; i<Cant.cant_ciclos && limi<CGRAFY2; i+=2 )
541. {
542.    limi = i*5;

543.    gcvt( ciclo[i+1].t, 6, num );
```

```
544.    outtextxy( CGRAFX1+50, CGRAFY1+30+limi, num );

545.    ampl[i] = fabs(ciclo[i].amplitud) + fabs(ciclo[i+1].amplitud);
546.    gcvt( ampl[i], 6, num );
547.    outtextxy( CGRAFX1+150, CGRAFY1+30+limi, num );

548.    period[i] = ciclo[i].periodo + ciclo[i+1].periodo;
549.    gcvt( period[i], 6, num );
550.    outtextxy( CGRAFX1+250, CGRAFY1+30+limi, num );

551.    den = ciclo[i].periodo + ciclo[i+1].periodo;
552.    if( den == 0.0 )
553.       den = 0.01;
554.    frec[i] = 1 / den;
555.    gcvt( frec[i], 6, num );
556.    outtextxy( CGRAFX1+350, CGRAFY1+30+limi, num );
557. }
558. GetKey();
559. setfillstyle( SOLID_FILL, NEGRO );
560. bar(CGRAFX1, CGRAFY1, CGRAFX2, CGRAFY2 );
561. settextstyle( TRIPLEX_FONT, HORIZ_DIR, 2 );
562. setfillstyle( SOLID_FILL, AZUL_GRIS );
563.}

564./*-------------------------------------------------------------------
565.Funciones utilitarias y de gráficas
566.-------------------------------------------------------------------*/
567.int AbrirArch( FUNCION *f )
568.{
569. REG     dev;
570. WORD    tecla;
571. char    comp[15], cad[25];
572. int     i, edo=0;
573. FILE    *fp;

574. bar(MENS_X1, MENS_Y1, MENS_X2, MENS_Y2 );
575. if( Edo_Arch == ABIERTO )
```

```
576. {
577.    outtextxy( MENS_X1+10, MENS_Y1+10, "Salvaguardar datos actuales?
   S/N" );
578.    do
579.    {
580.       tecla = GetKey();
581.       tecla = May( tecla );
582.    } while( tecla != 'S' && tecla != 'N' && tecla == ENTER );
583.    if( tecla == 'S' || tecla == ENTER )
584.       Salvaguarda( f );
585. }
586. bar(MENS_X1, MENS_Y1, MENS_X2, MENS_Y2 );
587. outtextxy( MENS_X1+10, MENS_Y1+10, "Nombre del archivo por abrir:"
   );
588. settextstyle( DEFAULT_FONT, HORIZ_DIR, 1 );
589. tecla = CapturaDato( MENS_X1+340, MENS_Y1+20, Pant.nom_arch, 4, 13,
   &dev );
590. settextstyle( TRIPLEX_FONT, HORIZ_DIR, 2 );
591. if( tecla == ESC )
592. {
593.    bar(MENS_X1, MENS_Y1, MENS_X2, MENS_Y2 );
594.    return 0;
595. }
596. if( tecla == ENTER )
597. {
598.    if( strlen( dev.c ) )
599.       strcpy( Pant.nom_arch, dev.c );
600. }
601. else
602. {
603.    bar(MENS_X1, MENS_Y1, MENS_X2, MENS_Y2 );
604.    return 0;
605. }

606. /* Codigo necesario para abrir un archivo
607. Debe verificarse la compatibilidad del archivo */
608. if( (fp=fopen( Pant.nom_arch, "r" )) != NULL )
609. {
```

```
610.    fgets( comp, 15, fp );  /* Necesario para validar un archivo */
611.    if( strncmp( comp, "ARV9GENLCC", 10 ) )
612.    {
613.      bar( MENS_X1, MENS_Y1, MENS_X2, MENS_Y2 );
614.      outtextxy( MENS_X1+10, MENS_Y1+10, "Archivo incompatible..."
    );
615.      GetKey();
616.      bar( MENS_X1, MENS_Y1, MENS_X2, MENS_Y2 );
617.      fclose( fp );
618.      Edo_Arch = CERRADO;
619.    }
620.    else
621.    {
622.      /* Inicialización de parámetros */
623.      Tiempo = 0;
624.      Cant.cant_pts = 1;
625.      Cant.cant_ciclos = 0;
626.      Func[0].y = 0;
627.      Func[0].t = 0;
628.      Func[0].signo = 0;
629.      for( i=0; fgets( cad, 20, fp ); i++ )
630.      {
631.        cad[6] = 0;
632.        f[i].t = atof( cad );
633.        f[i].y = atof( cad+7 );
634.      }
635.      Cant.cant_pts = i;
636.      fclose(fp);
637.      Edo_Arch = ABIERTO;
638.      edo = 1;
639.    }
640. }
641. else
642. {
643.    bar( MENS_X1, MENS_Y1, MENS_X2, MENS_Y2 );
644.    outtextxy( MENS_X1+10, MENS_Y1+10, "No pudo abrir archivo..." );
645.    GetKey();
```

```
646.    bar( MENS_X1, MENS_Y1, MENS_X2, MENS_Y2 );
647.    Edo_Arch = CERRADO;
648. }
649. bar( MENS_X1, MENS_Y1, MENS_X2, MENS_Y2 );

650. return edo;
651.}

652.void  ActualizaPantalla( char que )
653.{
654. char *arch[2] = {"Abrir", "Salvaguardar"};
655. char *pant[4] = {"Movimiento", "Periodo", "Frecuencia", "Amplitud"
   };
656. setfillstyle(SOLID_FILL, AZUL_GRIS );
657. setcolor( AMARILLO );
658. settextstyle( TRIPLEX_FONT, HORIZ_DIR, 2 );
659. bar( GRAF_X1, GRAF_Y1, GRAF_X2, GRAF_Y2 );
660. bar( ARCH_X1, ARCH_Y1, ARCH_X2, ARCH_Y2 );
661. if( que == ARCHIVO )
662. {
663.    Pant.arch = !Pant.arch;
664.    outtextxy( ARCH_X1+20, ARCH_Y1, arch[Pant.arch] );
665. }
666. if( que == GRAFICA )
667. {
668.    if(Pant.graf < AMP ) Pant.graf++;
669.    else Pant.graf = MOV;
670.    outtextxy( GRAF_X1+20, GRAF_Y1, pant[Pant.graf] );
671. }
672. bar( MENS_X1, MENS_Y1, MENS_X2, MENS_Y2 );

673.}

674.WORD  CapturaDato( int x1, int y1,char *dato, char tipo,
675.       int tam, REG *reg )
676.{
677. int     anch, alt;
```

```
678. WORD   tecla;
679. struct fillsettingstype info;

680. anch = (textwidth("H") * tam);
681. alt = textheight("H");

682. getfillsettings( &info );
683. setcolor( AMARILLO );
684. setfillstyle( SOLID_FILL, AZUL_GRIS );
685. bar( x1, y1, x1+anch, y1+alt );
686. outtextxy( x1, y1, dato );
687. while( 1 )
688. {      /* entero = 1, flotante = 2, cadena = 4 */
689.    tecla = LeeCadena( x1, y1, reg, AZUL_GRIS );
690.    if((tipo & 1) && reg->ent && strlen(reg->c) <= tam )    /* Entero
   */
691.    break;
692.    if((tipo & 2) && reg->flot && strlen(reg->c) <= tam )    /*
   Flotante */
693.    break;
694.    if((tipo & 4) && reg->alfa && strlen(reg->c) <= tam ) /* Cadena
   */
695.    break;
696.    /* se acepta el dato ya existente */
697.    if( (TeclaEsp(tecla) ) && !strlen(reg->c) )
698.      break;
699.    bar( x1,y1, x1+anch, y1+alt );
700. }

701. bar( x1, y1, x1+anch, y1+alt );
702. if( !strlen(reg->c) )
703.    outtextxy( x1, y1, dato );
704. else
705.    outtextxy( x1, y1, reg->c );
706. setfillstyle( info.pattern, info.color );

707. return tecla;
```

```
708.}

709.void ColorRgb(int re,int gr,int bl,int reg )
710.{
711. union REGS inregs;
712. inregs.h.ah = 0x10; inregs.h.al = 0x10;
713. inregs.x.bx = reg;  inregs.h.ch = gr;
714. inregs.h.cl = bl;   inregs.h.dh = re;
715. int86( 0x10, &inregs, &inregs );
716.}

717.WORD  GetKey(void)
718.{
719. union  {
720.    WORD scan;
721.    struct {
722.       UCHAR  lo;
723.       UCHAR  hi;
724.    } sep;
725. } key;

726. key.scan = bioskey(0);
727. return( (key.sep.lo) ? (WORD) key.sep.lo:
728.    (WORD) key.sep.hi + 0x100 );

729.}  /*------     FIN DE:  getkey()  ---------*/

730.WORD  GetIfKey(void)
731.{
732. if (bioskey(1))
733.    return GetKey();
734. else
735.    return 0;
736.}  /*------     FIN DE:  getifkey()  ---------*/

737.int InicializaGraficas( void )
738.{
```

```
 739. int  graphdriver = VGA, graphmode= VGAHI;
740. int  g_err;

741. initgraph(&graphdriver, &graphmode, "c:\\tc\\bgi" );
742. g_err = graphresult();
743. if(g_err < 0)
744. { printf("error de initgraph : %s\n", grapherrormsg(g_err));
745.   return -1;
746. }

747. if(  LeePaleta( ARCHIVO_PCX ) == -1 )
748.   return -1;

749. LoadScrn16(  ARCHIVO_VGA );
750. /* settextjustify( CENTER_TEXT, TOP_TEXT ) ;*/

751. return 1;
752.}

753.void InicializaParametros( void )
754.{
755. Pant.arch = ABRE;
756. memset(Pant.nom_arch, 0, 30 );
757. Pant.graf = MOV;
758. strcpy( Pant.nom_arch, "pract1.ama" );
759. setfillstyle(SOLID_FILL, AZUL_GRIS );
760. setcolor( AMARILLO );
761. settextstyle( TRIPLEX_FONT, HORIZ_DIR, 2 );
762.}

763./* lee una cadena de caracteres, la convierte a entero, flotante o
764.no lo convierte, según sea el caso.    */
765.WORD LeeCadena( int ctx1, int cty1, REG *reg, int bkcolor )
766.{
767. int    tecla=0, i=0, cont=0;
768. int    c=ENTER, edo[MAXCAR], alt, band_bspc;
769. struct fillsettingstype infofill;
```

```
770. int tabla[6][5] = { {2,3,4,1,6}, {2,3,4,3,6}, {2,3,4,3,6},
771.                  , {5,3,3,3,6}, {5,3,3,3,6} };
772. reg->ent = 0;
773. reg->flot = 0;
774. reg->alfa = 0;
775. reg->real = 0.0;
776. reg->entero = 0;
777. reg->c[0]   = 0;

778. while(1)
779. {
780.    edo[cont] = i;
781.    c = GetKey();
782.    band_bspc = 0;

783.    if( c>='0' && c<='9' )
784.      i = tabla[i][0], tecla = 1;
785.    if( (c>='a' && c<='z') || (c>= 'A' && c<='Z') || c==32 )
786.      i = tabla[i][1], tecla = 1;
787.    if( TeclaEsp(c) )
788.      i = tabla[i][4] ;
789.    switch( c )
790.    {
791.      case '.'  : i = tabla[i][2]; tecla = 1; break;
792.      case '-'  :
793.      case '+'  : i = tabla[i][3]; tecla = 1; break;
794.      case BKSPCE :
795.        if( cont > 0 )
796.          cont--;
797.        i = edo[cont];
798.        reg->c[cont] = 0;
799.        band_bspc = 1;
800.        break;
801.    }
802.    if( tecla )
803.    {
```

```
804.       reg->c[cont] = c;
805.       cont++;
806.       tecla = 0;
807.       reg->c[cont] = 0;
808.     }
809.     else
810.       if( !cont && !band_bspc ) return TeclaEsp( c );

811.     if ( cont >= MAXCAR ) return ENTER;
812.     alt  = textheight("H");
813.     getfillsettings(&infofill);
814.     setfillstyle( SOLID_FILL, bkcolor );
815.     bar(ctx1 , cty1, MENS_X2, cty1+alt+4 );
816.     setfillstyle(infofill.pattern, infofill.color);
817.     outtextxy( ctx1, cty1, reg->c );
818.     if( i == 6 )
819.     {
820.       switch( edo[cont] )
821.       {
822.         case 2:
823.           reg->ent = 1;
824.           reg->entero = atoi( reg->c );
825.           return c;
826.         case 4:
827.           if( cont <= 2 && (reg->c[0] == '-' || reg->c[0] == '+') )
828.           {
829.             reg->alfa = 1;
830.             return c;
831.           }
832.           if(cont < 2)
833.           {
834.             reg->alfa = 1;
835.             return c;
836.           }
837.           else
838.           {
839.             reg->ent = 1;
```

```
 840.              reg->c[cont-1] = 0;
841.              reg->entero = atoi( reg->c );
842.            }
843.            return c;
844.          case 5:
845.            reg->real = atof(  reg->c);
846.            reg->flot = 1;
847.            return c;
848.          case 1:
849.          case 3:
850.            reg->alfa = 1;
851.            return c;
852.       }
853.    }
854. }
855.}

856.int LeePaleta(char *archivo_pcx)
857.{
858. FILE *f1;
859. PCXHEAD header ;

860. if( (f1 = fopen( archivo_pcx, "rb" )) == NULL )
861.    return -1;
862. fread( (char *) &header, 1, sizeof(PCXHEAD), f1 ) ;
863.    PaletaPcx16( header.palette ) ;
864. fclose(f1);

865. return 1 ;
866.}



867.void LoadScrn16( char *nom )
868.{
869. char far *MemSVGA;
870. FILE *f1;
```

```
 871. int i;
872. char aux[50];

873. MemSVGA = (char far *) 0xA0000000L;

874. for(i=0 ; i<4 ; i++)
875. {
876.    sprintf(aux,"%s.%d",nom,i);
877.    f1=fopen( aux, "rb");
878.    outp(0x3c4,2);
879.    outp(0x3c5,1<<i);
880.    fread(MemSVGA,1,LONG_PLANO,f1);
881.    fclose(f1);
882. }
883. outp(0x3c4,2);
884. outp(0x3c5,0xF);
885.}

886.WORD May(WORD c)
887.{
888. if( c >= 'a' && c <= 'z' )
889. c -= 32;
890. return c;
891.}

892.void PaletaPcx16( char *p )
893.{

894. int i ;
895. unsigned char a, b, c ;

896. for(i=0 ; i<16 ; i++)
897. {
898.    a = p[3*i] ;
899.    b = p[3*i+1] ;
900.    c = p[3*i+2] ;
901.    a/=4, b/=4, c/=4 ;
```

```
 902.    ColorRgb( a, b, c, COLOR16[i] ) ;
903. }
904.}

905.WORD  TeclaEsp( WORD c )
906.{
907. WORD tecla[25] = { CTR_C, BKSPCE, TAB, ENTER, ESC,
908.    F1, F2, F3, F4, F5, F6, F7, F8, F9, F10,
909.    HOME, PGUP, PGDN, INS, DEL, END, UP, DOWN,IZQ, DER };
910. int i;

911. for( i=0; i<25 && c!=tecla[i]; i++ );
912. if ( i == 25 ) return 0;
913. return tecla[i];
914.}

915.void Salvaguarda( FUNCION *f )
916.{
917. REG     dev;
918. int     i;
919. FILE    *fp;
920. WORD    tecla;

921. bar( MENS_X1, MENS_Y1, MENS_X2, MENS_Y2 );
922. if( Cant.cant_pts < 10 || Edo_Arch == CERRADO )
923. {
924.    outtextxy( MENS_X1+10, MENS_Y1+10, "No hay datos..." );
925.    GetKey();
926.    bar( MENS_X1, MENS_Y1, MENS_X2, MENS_Y2 );
927.    Edo_Arch = CERRADO;
928.    return;
929. }
930. outtextxy( MENS_X1+10, MENS_Y1+10, "Nombre del archivo a
   salvaguardar:" );
931. settextstyle( DEFAULT_FONT, HORIZ_DIR, 1 );
932. tecla = CapturaDato( MENS_X1+390, MENS_Y1+20, Pant.nom_arch, 4, 13,
   &dev );
```

```
933. settextstyle( TRIPLEX_FONT, HORIZ_DIR, 2 );
934. if( tecla == ESC )
935. {
936.    bar(MENS_X1, MENS_Y1, MENS_X2, MENS_Y2 );
937.    return;
938. }
939. if( tecla == ENTER )
940. {
941.    if( strlen( dev.c ) )
942.       strcpy( Pant.nom_arch, dev.c );
943. }
944. else
945. {
946.    bar(MENS_X1, MENS_Y1, MENS_X2, MENS_Y2 );
947.    return;
948. }

949. /* Codigo necesario para Salvaguardar un archivo */
950. if( (fp=fopen( Pant.nom_arch, "w" )) != NULL )
951. {
952.    fputs( "ARV9GENLCC\n", fp ); /* Necesario para validar un archivo
   */
953.    for( i=0; i < Cant.cant_pts; i++ )
954.       fprintf( fp, "%6.3f %6.3f\n", f[i].t, f[i].y );
955.    fclose(fp);
956.    Edo_Arch = CERRADO;
957. }
958. else
959. {
960.    bar( MENS_X1, MENS_Y1, MENS_X2, MENS_Y2 );
961.    outtextxy( MENS_X1+10, MENS_Y1+10, "No pudo salvaguardar..." );
962.    GetKey();
963.    bar( MENS_X1, MENS_Y1, MENS_X2, MENS_Y2 );
964.    Edo_Arch = CERRADO;
965. }
966. bar( MENS_X1, MENS_Y1, MENS_X2, MENS_Y2 );
967.}
```