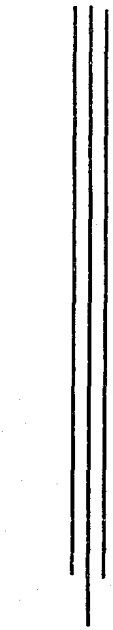
Universidad Nacional Autónoma de México

FACULTAD DE INGENIERIA



Sistema de medición por Coordenadas en base a medios ópticos, utilizando una Computadora Personal

T E S I S

QUE PARA OBTENER EL TITULO DE INGENIERO EN COMPUTACION

P R E S E N T A

RUBEN MANUEL MARQUEZ VILLEGAS

Director de Tesis:
ING. MAURICIO GARCIA ESTEBAN

MEXICO, D. F. 1990

UNAM – Dirección General de Bibliotecas Tesis Digitales Restricciones de uso

INDICE

I ANALISIS DE UN SISTEMA DE MEDICION POR COORDENADAS BASADO EN UNA MICROCOMPUTADORA.

1.1 DEFINICION DE UN SISTEMA COMPUTARIZADO DE MEDICION POR COORDENADAS.

Existe una gran variedad de procesos en los cuales el conocimiento de un desplazamiento relativo, una posición determinada o una tendencia de contornos, constituyen la información fundamental para el monitoreo, análisis y/o control de los eventos y acciones implícitas en el desarrollo del proceso mismo; en cualquier caso, esta información puede ser tratada como un conjunto de puntos, vectores o coordenadas espaciales.

El objetivo de un sistema de medición de coordenadas es el de generar esta información, realizando las mediciones dimensionales que demandan los procesos mencionados anteriormente. Al realizar mediciones precisas de desplazamientos con una resolución muy alta, se puede localizar con gran exactitud un conjunto de puntos en el espacio, asociando a cada uno de ellos un vector único que lo identifique; este conjunto puede representar cosas tan simples como la longitud de un objeto, o la digitalización del contorno bidimensional y tridimensional del mismo.

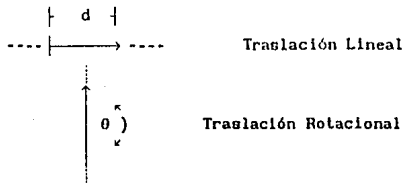
Un sistema de medición por coordenadas debe tener como características primarias una alta resolución, del orden alcanzado por métodos optoelectrónicos, además de gran estabilidad y excelente inmunidad en condiciones ambientales diversas, tales como vibración mecánica y temperaturas extremas. Estas características permiten que el conjunto de aplicaciones posibles para un sistema de este tipo se vea incrementado dramáticamente, abarcando áreas tales como la metrología dimensional industrial y de laboratorio.

1.1A Introducción a los Sistemas de Coordenadas.

Un Sistema de Coordenadas está definido por la representación que se le da a la posición de un evento en un espacio, con respecto a otro evento llamado origen o referencia.

Para el caso de un espacio tridimensional, la posición de un punto puede determinarse por un conjunto ordenado de tres valores numéricos reales, los cuales especifican el desplazamiento o distancia del mismo con respecto a la referencia utilizada en ese espacio; este conjunto, conocido como Vector, comprende la información geométrica acerca de la posición del punto. A los elementos del conjunto se les conoce como componentes o coordenadas, y cada uno de ellos representa un desplazamiento con respecto a un conjunto de planos y ejes perpendiculares entre sí.

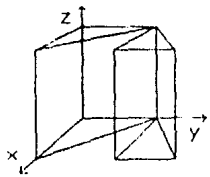
El espacio generado mediante estos sistemas es un Espacio Numérico Tridimensional Sólido (ENTS), en el cual existen una infinidad de puntos, que tienen asociado un vector único que los representa. La representación dada a la posición de un punto, depende del tipo de movimientos necesarios para alcanzarla. Si el movimiento se realiza a lo largo de una línea recta, se tiene un desplazamiento lineal, el cual es cuantificado en unidades de longitud (metros, pulgadas, etc.); si el movimiento realizado implica una traslación rotacional o giro con respecto a un eje, entonces se trata de un movimiento angular, el cual se expresa en unidades angulares (radianes, grados, etc.).



Debido a las característica del movimiento, se pueden obtener diversos sistemas, los cuales generan espacios tridimensionales topográficamente distintos, dependiendo del tipo de traslaciones (lineales y rotacionales) utilizadas. De acuerdo a la forma del espacio que generan son Rectangulares, Cilíndricos y Esféricos.

Sistema de Coordenadas Rectangulares.

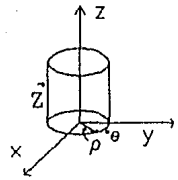
Para generar un ENTS Rectangular se necesitan tres traslaciones lineales. El Sistema de Referencia está formado por tres rectas mutuamente perpendiculares entre sí, generalmente denominadas como ejes X, Y y Z, y el origen está situado en el punto donde se intersectan las tres.



Las coordenadas $v(x, y, z)$ indican las distancias dirigidas, medidas con respecto a los 3 planos formados por los ejes. De esta forma la coordenada X cuantifica el desplazamiento efectuado sobre el eje X, medido desde el plano formado por los ejes Y y Z.

Sistema de Coordenadas Cilíndricas.

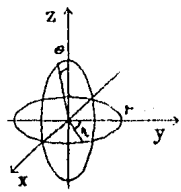
Las coordenadas Cilíndricas son una generalización de las coordenadas Polares en tres dimensiones. Están formadas por una traslación Lineal ρ o radio de alcance, una traslación Rotacional θ o azimut y una traslación Lineal z, la cual representa la distancia dirigida desde el plano Polar hasta un punto situado en un plano paralelo al plano polar.



El sistema de referencia sigue estando formado por tres ejes mutuamente perpendiculares, mientras que el origen está localizado en el punto donde se intersectan el eje de rotación Z con el polo del plano Polar $\rho\theta$.

Sistema de Coordenadas Esféricas.

Este sistema está formado por un plano Polar $\rho\theta$, y se utiliza una traslación rotacional o elevación η, para obtener el espacio tridimensional. El espacio se genera con dos movimientos azimutales y una traslación lineal o radio de alcance.

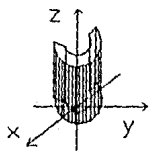


El sistema de referencia está formado por los tres ejes perpendiculares mencionados y el origen está determinado por la intersección del polo del plano Polar $\rho\theta$, con el polo del plano Polar $\rho\eta$ de la elevación.

Volumen Espacial y Espacio de Trabajo.

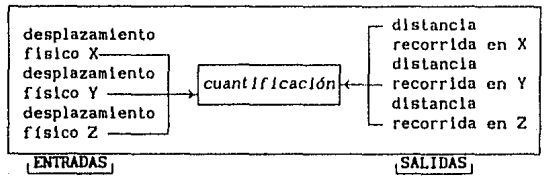
Las representaciónes propias de cada uno de los ENTS anteriormente descritos generan un Volumen Espacial, cuya forma está dada por la evolución de cada una de las traslaciones posibles y de acuerdo al tipo de coordenadas que lo definen; se considera que no existen limitaciones importantes en la magnitud de los movimientos longitudinales y que pueden realizarse rotaciones de hasta 360°. Considerando un dispositivo con un sistema de movimientos implementado mediante un conjunto de articulaciones lineales y rotacionales, se obtiene un Espacio de Trabajo, que es un subconjunto del Volumen Espacial. Las limitaciones físicas de las articulaciones utilizadas para movimientos longitudinales y angulares, introducen restricciones en la magnitud de los movimientos que pueden realizarse.

Usando coordenadas Cilíndricas (ρ, θ, z), donde ρ toma valores mínimos y máximos r1 y r2, el espacio está limitado por un cilindro hueco. La altura del cilindro está limitada por la longitud de recorrido $z = d$ de la articulación lineal. Si se introduce una restricción en la articulación rotacional, de tal forma que $0 < \theta < 2\pi$, entonces el espacio de trabajo disminuye en la sección de arco correspondiente.

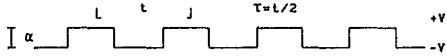


1.1B Definición de Entradas y Salidas del Sistema de Medición.

En su concepción más simple, el sistema tiene como entrada tres grupos de señales correspondientes a desplazamientos en los ejes X, Y y Z. Las señales deben ser acondicionadas y procesadas para cuantificar la magnitud real del desplazamiento; finalmente los valores numéricos son desplegados utilizando unidades del Sistema Internacional , del Sistema Inglés, etc.



Las señales de entrada provienen de un dispositivo que codifica en señales digitales la magnitud y el sentido del desplazamiento. Se necesita al menos un dispositivo por cada eje del sistema de referencia. Se trata de señales que normalmente están formadas con un ciclo de trabajo τ, la amplitud α de las señales está dada por los valores de polarización proporcionados a los dispositivos.



FORMA DE ONDA TÍPICA DE UN CODIFICADOR

La tarea principal del sistema es medir, por lo que la salida mínima que debe obtenerse es la magnitud numérica de los desplazamientos realizados. Las cifras deben reflejar con gran precisión y exactitud los datos obtenidos y codificados por el dispositivo sensor. Adicionalmente algunos sistemas de medición realizan algunas otras funciones, como son :

-Digitalización de contornos.
-Cambio de escalas de medición.
-Modificación de factores de resolución en los codificadores.
-Inicialización de puntos de referencia.

1.1c Ventajas del Procesamiento por Computadora.

Con el desarrollo de los semiconductores y de los circuitos de alta escala de integración, se ha facilitado el acceso a todo el poder de cómputo de un equipo. Al tener acceso al *hardware* de la computadora en una forma barata, el espectro de aplicaciones se ha incrementado dramáticamente. Uno de los equipos más comercializados, la computadora personal, es totalmente accesible en la componente electrónica de su arquitectura, ya que la información necesaria para desarrollar sistemas que mejoren y/o complementen sus capacidades, está disponible en el mercado de partes y componentes electrónicos, en bibliotecas, etc. El potencial de las computadoras ha dejado de radicar únicamente en su capacidad para realizar secuencias complejas de instrucciones, ahora también pueden usarse como un poderoso instrumento para la adquisición de datos y el control de procesos.

La utilización de equipos digitales de cómputo en el diseño, desarrollo e implementación de equipos de instrumentación, pone al alcance de los diseñadores e investigadores una amplia gama de sistemas lógicos, electrónicos y de cómputo cuya capacidad de procesamiento es muy elevada. Los sistemas de instrumentación desarrollados en base a computadoras (o microprocesadores) son muy versátiles, debido a que el procedimiento utilizado para medir se convierte en un algoritmo lógico, que se implementa por un lado en un circuito lógico, y por otro en un programa; este tipo de arquitecturas permite alcanzar tiempos de respuesta muy pequeños, a la vez que se incrementa significativamente la precisión de los resultados obtenidos.

Al diseñar el sistema de medición por coordenadas utilizando una computadora, los costos decrecen significativamente. Toda la etapa electrónica se desarrolla en una tarjeta que representa una extensión de la tarjeta principal, es decir, no se trata de un sistema dedicado, sino de una aplicación más de todas las existentes para la computadora. Los programas pueden hacer al sistema más accesible al usuario y proporcionar una gran variedad de funciones y aplicaciones adicionales relacionadas con la tarea principal. La digitalización de contornos, por ejemplo, se convierte en una tarea muy simple que puede realizarse a través de un comando; así mismo, la información del contorno digitalizado se puede almacenar fácilmente en un disco, sin tener que retransmitirla desde o hacia algún dispositivo externo que no forme parte de la estructura física del sistema; en otras palabras, se alcanza una mayor integración entre los sistemas informáticos de medición, análisis y diseño.

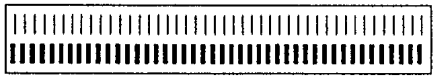
- Versatilidad.
- Modularidad.
- Velocidad de respuesta.
- Automatización.
- Abatimiento de costos.
- Mínima interdependencia entre *hardware* y *software*.
- Integración de aplicaciones de *hardware* y *software*.

1.2 DISPOSITIVOS OPTOELECTRONICOS DE MEDICION DE ALTA RESOLUCION.

1.2A CODIFICADORES OPTICOS.

El codificador óptico es el dispositivo más novedoso perteneciente a la clase de codificadores de no contacto, desarrollados para eliminar algunos de los problemas mecánicos asociados con su uso. Los codificadores ópticos actuales proporcionan la mayor precisión y pueden ser operados a altas velocidades. Los métodos optoelectrónicos de medición utilizados con mayor frecuencia en los Sistemas de Medición de Coordenadas están construídos usando dos elementos mecánicos básicos: en una articulación rotacional se utiliza un disco, y en una articulación lineal se utiliza una regleta. Estos elementos tienen grabados en su superficie varias pistas, formadas a su vez por una sucesión de zonas transparentes y opacas, alternadas y espaciadas a una distancia constante. En algunos casos se trata de depósitos de material opaco a una banda determinada de frecuencias de radiación electromagnética.

Para detectar la magnitud del movimiento, cada pista se examina con dispositivos optoelectrónicos capaces de distinguir entre las zonas transparentes y opacas. Los más sencillos y generalizados son el diodo emisor de luz (LED) y un fototransistor. Al moverse la articulación, el fototransistor genera una señal digital compuesta por un secuencia de pulsos correspondientes, por ejemplo a las zonas oscuras. El algoritmo utilizado para la codificación está definido por la distribución geométrica y la longitud de las zonas. Las señales digitales resultantes pueden variar desde una palabra binaria, que indique el desplazamiento mediante una cifra, hasta una señal cuadrada que proporcione mediante el número de pulsos que la componen, una indicación de la magnitud del desplazamiento.



Pistas de igual resolución, pero distinto ciclo de trabajo.

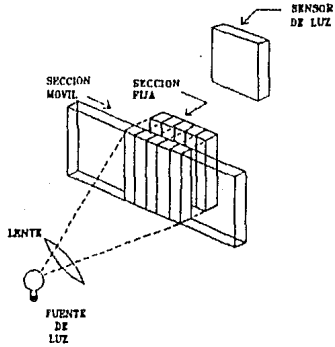
Cuando fueron desarrollados por primera vez, los codificadores ópticos no podían soportar condiciones ambientales adversas , sin embargo los modelos más recientes presentan gran resistencia a vibraciones mecánicas, temperaturas extremas (menores de 0°C y mayores de 50°C) y buena estabilidad durante impactos de alta energía.

La construcción física de la interfaz y de sus líneas de enlace con el codificador óptico es muy simple, pues se requieren únicamente dos líneas para transmitir la información desde la regla hasta el circuito de procesamiento. Los elementos mecánicos necesarios consisten en conectores de cualquier tipo, colocados en un extremo del codificador y en la tarjeta donde se encuentre el preprocesador; debido a las razones anteriores, los codificadores ópticos incrementales son ideales para aplicaciones industriales y de laboratorio.

1.2B CODIFICADORES INCREMENTALES.

A diferencia de otros codificadores , en estos existen únicamente dos pistas paralelas de zonas claras y oscuras; la relación en la disposición geométrica de las zonas permite que se generen dos señales con un defasamiento constante entre ellas. El sentido de la traslación puede determinarse examinando la fase, si se toma una de las dos señales como referencia. La cuantificación del movimiento puede realizarse mediante el conteo de los pulsos ocurridos en una o ambas señales, o bien mediante el conteo de los flancos de subida y de bajada presentes en ambas señales.

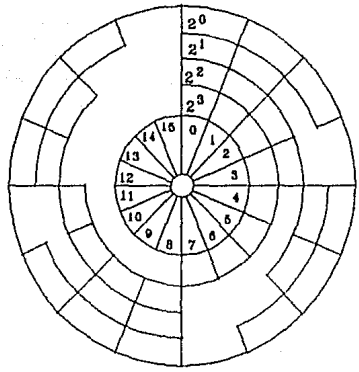
Los componentes básicos del codificador óptico incremental son: una fuente de luz (que puede ser una lámpara, un LED laser o un LED de emision infrarroja), un arreglo óptico de rejilla o reticula para filtrar la luz, y un sensor de luz. En su configuración más simple, el arreglo óptico está formado por una placa fija y otra móvil, las cuales están inscritas con franjas y espacios de igual longitud y anchura. El sensor usado en muchos de los codificadores es una celda fotovoltáica de área amplia.



La placa móvil del arreglo óptico se fija a una articulación lineal. Cuando las rejillas de la placa móvil están perfectamente alineadas con las rejillas de sección fija, el sensor detecta un máximo de luz. Al trasladar la placa móvil de tal forma que la franjas oscuras cubran las secciones transparentes de la rejilla en la placa fija, el sensor detectará un mínimo de luz. Mediante el procedimiento anterior y una etapa electrónica adecuada, se puede generar un frente de onda de suficiente calidad para ser usado en la cuantificación del movimiento. Las formas de las ondas correspondientes a las señales generadas por estos codificadores son las siguientes: senoidal, cuadrada y de pulsos.

1.2c Codificadores Absolutos.

El algoritmo utilizado en estos codificadores se basa en los procedimientos usados para generar un código binario. El código se obtiene en *n* bits a partir de *n* pistas adyacentes, las cuales se colocan de tal forma que se generan una serie de frentes de onda cuadrados. La relación de frecuencias de las señales así obtenidas representan las 2^n combinaciones posibles del código implementado.



1.3 PROPUESTA DE DISEÑO E IMPLEMENTACION.

Para el desarrollo de este trabajo se han utilizado como marco de referencia las necesidades del laboratorio de Metrología del Centro de Instrumentos de la UNAM. Para realizar mediciones y análisis en Metrología Dimensional se requiere de un sistema de alta resolución, gran velocidad de procesamiento, y la capacidad para memorizar coordenadas de puntos en el espacio tridimensional. La medición debe realizarse aún con desplazamientos simultáneos en las tres coordenadas, sin que la resolución y la velocidad se vean disminuidas o limitadas. Las características mínimas con que debe contar el sistema son las siguientes:

- 2 micras (millonésimas de metro) de resolución.
- Velocidad de medición de 25 cm/s.
- Capacidad para memorizar 1000 puntos en el espacio.
- Individualidad de operación en cada uno de los ejes.

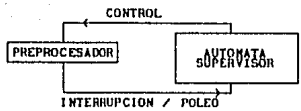
Los Sistemas de Medición por Coordenadas disponibles en el mercado son sistemas basados en equipos de propósito específico, su costo y complejidad son altos mientras que su disponibilidad está limitada. La operación de estos equipos requiere de cierta preparación técnica para su uso.

Un Sistema de Medición por Coordenadas que cumpla con los requisitos mencionados puede desarrollarse en base a codificadores ópticos incrementales y una computadora personal tipo IBM PC XT o alguna otra compatible. Utilizando estos equipos el costo y complejidad del sistema se ven reducidos, mientras que su versatilidad y disponibilidad se incrementa, como se mencionó en la sección 1.1c, alcanzando aplicaciones en áreas educativas e industriales.

Los sensores más apropiados para este sistema son los codificadores ópticos incrementales. Dado que cualquier punto inicial de desplazamiento puede considerarse como el origen, las mediciones realizadas son relativas. Esto permite que la operación sea muy rápida, ya que la articulación no tiene que llegar a una posición determinada para encontrarse en un punto inicial. En muchas aplicaciones los codificadores con referencia flotante son ideales, porque son los cuantificadores del movimiento y no el codificador óptico, los que definen el punto de origen, esto se logra mediante un comando de puesta a *"cero"*; tal es el caso de las máquinas de medición por coordenadas, de los graficadores X-Y (plotters), de las máquinas de electroerosión, etc.

Como se tiene un sistema tridimensional, se necesita implementar el mismo algoritmo de procesamiento en cada uno de los ejes. El método más usual para medir consiste en utilizar un microprocesador por cada eje para realizar mediciones individuales, además existe una lógica inteligente de despliegue y control que puede consistir en otro procesador. Con el método anterior se asegura que las condiciones existentes en un eje no afectan a las existentes en otro, sin embargo su costo se ve incrementado en forma proporcional al número de procesadores.

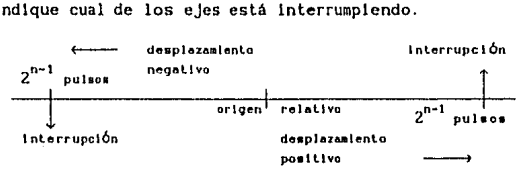
Si se utiliza un esquema de interrupciones, que codifique la información proveniente del sistema de medición en códigos de interrupción distintos para cada uno de los ejes y sus sentidos de movimiento, se puede desarrollar un dispositivo que utilice un solo procesador para atender los eventos que se generan en todos los ejes en forma simultánea, mientras que se encarga de monitorear y controlar otras tareas, tales como el despliegue de resultados o peticiones de tareas específicas, entre las que se incluye la memorización de puntos. El esquema de atención que se usará para accesar la información de los codificadores, está compuesto por una combinación de los dos métodos más usuales: el "poleo" o lectura de estado y las interrupciones.



FLUJO DE SEÑALES

La computadora será informada, mediante una interrupción, que la regla se ha desplazado N veces la distancia mínima de resolución de la regla. El valor de N está dado por el número de bits n del contador utilizado para cuantificar el desplazamiento, es decir $N=2^n-1$, cualquier desplazamiento ocurrido desde la última interrupción será conocido mediante la lectura del valor alcanzado por los contadores.

En el espacio tridimensional se tienen tres ejes de referencia, en cada uno de ellos existen dos sentidos de desplazamiento. Si se divide el grupo de seis direcciones posibles de traslación, en dos grupos que se denominarán de "*desplazamiento positivo*" y de "*desplazamiento negativo*" y se codifican las interrupciones pertenecientes al mismo tipo de desplazamiento, se requerirán solo dos líneas de interrupción, una por cada sentido de desplazamiento, adicionalmente se necesitará un código que indique cual de los ejes está interrumpiendo.



Las rutinas de servicio a las interrupciones deberán ser capaces de reconocer el código y actualizar las variables correspondientes a cada eje. Siguiendo esta estructura se tendrá un contador de pulsos con los bits menos significativos implementados en *hardware* y los más significativos implementados en *software*, el valor de este contador de pulsos representa la magnitud real del desplazamiento. El procedimiento que realiza la tarea anterior puede ser enunciado de la siguiente forma :

Sean Γ el eje intrinseco de una articulación de traslación lineal, R la resolución incremental de un sensor óptico, n el módulo de los contadores usados en una interfaz (el número de bits), I el número de pulsos generados mediante desplazamientos negativos desde la última interrupción, D el número de pulsos debidos a desplazamientos positivos, NI el número de interrupciones por desplazamientos negativos y ND el número de interrupciones debidas a desplazamientos positivos; entonces la distancia recorrida δ sobre el eje Γ está dada por:

$$\delta_{\Gamma} = [\ (ND-NI)(2^{n-1})+(D-I)\ ] * R \qquad (1)$$

donde el término (ND - NI) representa la información que se encuentra en registros o variables de *software* y (D - I) es la información que se encuentra en registros de *hardware* como contadores binarios. Por último, el término (2^{n-1}) representa el peso o valor que se asigna a cada interrupción, lo que resulta lógico ya que la interrupción puede ser generada por el n-ésimo bit de los contadores.

La computadora personal de IBM está desarrollada en base al esquema de interrupciones de los procesadores iAPX, lo que facilita implementar el procedimiento anterior utilizando un lenguaje de alto nivel que permita accesar el *hardware* de la tarjeta principal de la computadora, usando las ranuras de expansión que facilitan la adición de componentes a la tarjeta principal.

La utilización de un lenguaje de alto nivel proporciona la infraestructura apropiada para desarrollar un sistema modular y estructurado que incluya una interfaz hombre-máquina accesible para el usuario.

II SISTEMA DE INTERRUPCIONES DE LA COMPUTADORA PERSONAL DE IBM.

2.1 ESQUEMA DE INTERRUPCIONES DE LA FAMILIA DE PROCESADORES INTEL iAPX.

Los microprocesadores de la familia iAPX de INTEL (8086, 8088, 80186, 80188 y 80286) son procesadores compatibles con una arquitectura básica común, contienen un conjunto de registros totalmente compatibles, así como instrucciones y modos de direccionamiento comunes.

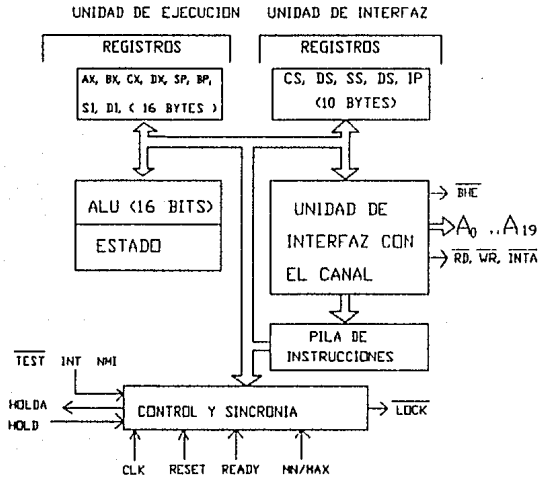
2.1A ORGANIZACIÓN LÓGICA.

Internamente los procesadores iAPX están divididos en dos unidades lógicas funcionales que interactúan para ejecutar una serie de instrucciones o programa, y se denominan como sigue:

- Unidad de Interfaz con el Canal (BIU)
- Unidad de ejecución (EU)

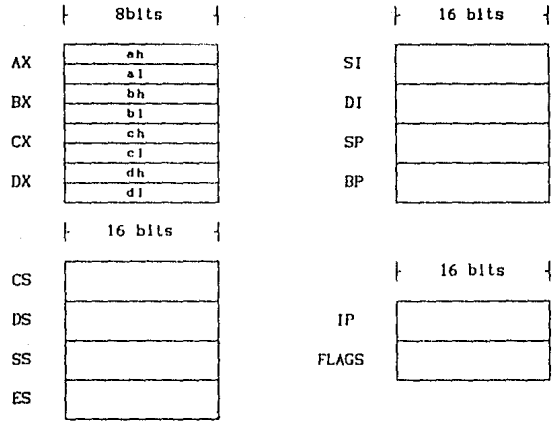
Estas unidades pueden interactuar directamente en forma síncrona si es necesario. Normalmente sólo operan en esta forma cuando alguna instrucción obliga a ambas unidades a sincronizarse, la mayoría de las veces operan en forma asíncrona, como dos procesadores independientes. La unidad de interfaz con el Canal proporciona las funciones para realizar la actualización del direccionamiento del programa, el acceso y almacenamiento de las instrucciones y operandos, además genera las señales básicas de control del Canal. La BIU actúa como una estructura de almacenamiento del tipo FIFO , es decir, el primero en entrar es el primero en salir, de esta forma, siempre están disponibles en una memoria de acceso rápido al menos seis bytes del programa.

La unidad de ejecución (EU) obtiene las instrucciones y operandos almacenados en la memoria FIFO del BIU para decodificarlas y ejecutarlas. También actualiza el direccionamiento de las instrucciones del programa informando al BIU de las nuevas direcciones y de los resultados que requieren ser almacenados.



La arquitectura de la familia iAPX tiene un "Modelo de Programación" o "Modelo de Registros" que está compuesto por catorce registros de dieciséis bits, agrupados en cuatro categorías:

-Registros de Propósito General (AX, BX, CX, DX);
-Registros de Segmento (CS, DS, SS, ES);
-Registros de Propósito Específico (SI, DI, SP, BP);
-Registros de Estado y Control (IP, FLAGS).



MODELO DE PROGRAMACION O DE REGISTROS.

Organización de memoria y direccionamiento.

Utilizando los registros de direccionamiento de segmento, se pueden formar cuatro bloques de memoria reservados para propósitos específicos. Dichos bloques comprenden el Área de Código o programa usando el *Code Segment* (CS), el Área de Datos Locales usando el *Data Segment* (DS), la Pila con el *Stack Segment* (SS) y un Área de Datos Externos con el *Extra Segment* (ES). Estos bloques tienen una longitud de 64KB y pueden estar traslapados.

Las direcciones físicas reales tienen un formato de veinte bits efectivos, es decir, se tiene una capacidad de direccionamiento de un MByte. Para formar una dirección se toma el contenido del registro de segmento correspondiente (código o datos) y se hace un corrimiento lógico hacia la izquierda de cuatro bits, entonces se suma el valor del IP.

dirección física = segmento * 16 + desplazamiento (2)

Sistema de Interrupciones.

El esquema de interrupciones que se ha implementado con esta familia de procesadores es conocido como "Interrupciones Vectorizadas", debido a que se reserva un bloque de "1" KByte en el espacio de direccionamiento a partir de la dirección inicial $0000O_{16}$, hasta la $003FF_{16}$ para almacenar una tabla de 256 vectores, los cuales apuntan hacia la dirección de inicio de un número igual de rutinas de atención a interrupciones. Cada elemento de la tabla es de cuatro Bytes, dos de ellos corresponden al *Code Segment* (CS) y los otros dos al *Instruction Pointer* IP. Con el contenido de este vector, una vez cargado en los registros correspondientes, se forma la dirección de inicio de la rutina que se ocupará de atender a la interrupción.

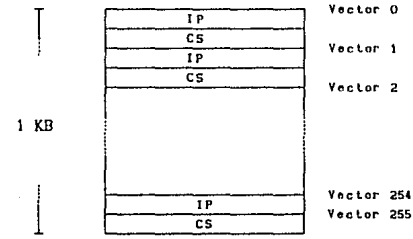


TABLA DE VECTORES DE INTERRUPCION

Cuando se presenta una interrupción, ocurre una transferencia de control hacia una nueva dirección del programa, es decir, el procesador se encuentra ejecutando una tarea, en cierto momento detecta la existencia de una interrupción, entonces termina de ejecutar la instrucción que estaba atendiendo y reconoce la interrupción, inmediatamente accesa el vector correspondiente que debe encontrarse en el Canal de datos. En ese momento se salvan los registros CS, IP y el registro de estado en el *Stack*, para tener una dirección de retorno al terminar la rutina de atención, y se copia en ellos la dirección contenida en la tabla de vectores de acuerdo al vector leído.

El vector de interrupción es un número entre 0 y 255. Para determinar la dirección de la localidad de la tabla de vectores en donde se encuentra la dirección de inicio de la rutina de interrupción, al código de identificación del vector se le hace un corrimiento de 2 Bits hacia la izquierda, equivalente a una multiplicación por 4.

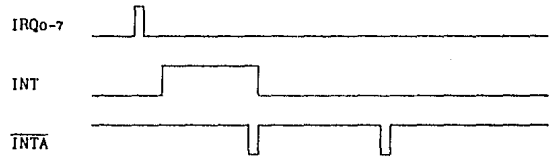
Al terminar la rutina de interrrupción, se ejecuta una instrucción de retorno de interrupción (IRET) que recupera del *Stack* los valores de CS, IP y el registro de estado, con lo cual se regresa al programa al mismo estado en que se encontraba, siempre que se cumpla la condición de que el contenido de los otros registros no haya sido modificado, o bien, que también haya sido salvado en el Stack y recuperado antes de ejecutarse la instrucción IRET. Existen dos tipos de interrupciones, las causadas por dispositivos externos o periféricos, también llamadas "Interrupciones de *hardware*", y las iniciadas por la ejecución de una instrucción especial del conjunto de instrucciones del procesador o "Interrupciones de *software*".

2.1B INTERRUPCIONES DE HARDWARE.

Las interrupciones de *hardware* pueden clasificarse en dos categorías, las interrupciones No filtrables y las interrupciones filtrables. La diferencia entre ambas categorías radica en la habilidad para impedir que una interrupción sea reconocida, esto se logra mediante la bandera de habilitación de interrupción (IF) en el registro de estado. Esta bandera se considera un filtro que permite o impide que una interrupción sea atendida.

Las interrupciones No filtrables son reconocidas y atendidas siempre que se presentan y tienen una prioridad mayor que las interrupciones filtrables. Pueden deshabilitarse internamente por *software* si se pone un "*cero*" en el Bit IF (Interrupt enable), del registro de estado (flags).

Para el caso de interrupciones de *hardware*, la señal con la que se solicita el servicio es una señal activa en un nivel lógico alto. Cuando el procesador detecta un flanco de subida en sus terminales NMI o INTR, inicia una secuencia de reconocimiento de interrupción mediante dos pulsos sucesivos en su terminal $\overline{INTA}$; después de enviar el segundo reconocimiento el procesador lee del canal un byte o vector que identifica a la fuente de donde proviene la interrupción, este byte es el vector de interrupción.



SECUENCIA DE RECONOCIMIENTO DE INTERRUPCION.

2.1c INTERRUPCIONES DE SOFTWARE.

Una interrupción de *software* no es generada por un dispositivo externo o periférico, se trata de una función que es invocada mediante una instrucción propia de un procesador iAPX. El formato para esta instrucción es el siguiente:

INT < int # >

en donde int # es un número entre 0 y 255, que corresponde a alguno de los vectores de la tabla de vectores de interrupción.

De la misma forma que en una interrupción de *hardware*, al ejecutarse una interrupción de *software*, se salva el contenido de los registros CS, IP y el registro de estados, se deshabilitan las interrupciones mediante el bit IF del nuevo registro de estado y se transfiere el control de programa hacia la dirección indicada por los nuevos valores de CS e IP contenidos en el vector <int #>. En este tipo de interrupciones no se generan los dos pulsos de reconocimiento de interrupción $\overline{INTA}$, ya que no es necesario avisar a ningún dispositivo que la interrupción ha sido reconocida.

2.2 DESCRIPCION DEL CONTROLADOR DE INTERRUPCIONES INTEL 8259A.

Al utilizarse un esquema de interrupciones vectorizadas como el método de atención a los dispositivos de Entrada Salida, sensores y otros periféricos, es necesario incluir circuitos adicionales en el sistema de cómputo. Estos dispositivos deben proporcionar la información acerca de la fuente u origen de la interrupción, y de ser necesario, del procedimiento o rutina de *software* que debe atenderlos. El controlador programable de interrupciones (CPI) 8259A de Intel, cumple con estas funciones y proporciona otras que facilitan la utilización de interrupciones.

La computadora personal de IBM incluye un CPI 8279A dentro de su sistema básico; este CPI se encuentra en la tarjeta principal y su tarea consiste en atender las interrupciones de algunos elementos del sistema básico y periféricos. La asignación de dispositivos en las ocho líneas de interrupción es de la siguiente forma:

IRQ0 - Base de tiempo	IRQ4 - Puerto Serie 1
IRQ1 - Teclado	IRQ5 - Disco Duro
IRQ2 - Video	IRQ6 - Disco Flexible
IRQ3 - Puerto Serie 2	IRQ7 - Puerto Paralelo

2.2A ORGANIZACIÓN LÓGICA.

El CPI 8259A está compuesto por ocho bloques funcionales: El registro de Petición de Interrupción (IRR) se usa para almacenar todas las interrupciones provenientes de dispositivos que solicitan ser atendidos; el registro de servicios (ISR) almacena todas las peticiones que están siendo atendidas, y el registro de filtros de interrupción (IMR) almacena los bits de las líneas de interrupción que serán filtradas o deshabilitadas. La lógica de

solución de prioridades (PR) verifica los valores de los registros IRR, ISR e IMR para determinar cual de las peticiones de interrupción debe ser transmitida al microprocesador. El registro de datos (DBB) es un registro bidireccional de 8 bits del tipo de alta impedancia (tercer estado), y se usa para realizar la conexión del CPI con el canal de datos del sistema o del microprocesador. Los comandos de control, la información de estado y los vectores de interrupción son transferidos por medio de este registro. La lógica de control de lectura y escritura (R/W CL) se encarga de programar al CPI para aceptar comandos de control y datos para los registros de control, así como de liberar la información de estado, vectores de interrupción y datos que le son solicitados al CPI.

El registro de cadena y comparador (CBC) se usa para permitir el enlace de varios controladores de interrupciones, mediante el esquema conocido como Maestro-Esclavo. La lógica de control (CL) es la última etapa funcional del CPI y se encarga de enviar la señal de interrupción al sistema, de acuerdo con lo indicado por la lógica de solución de prioridades; también se encarga de recibir las señales de reconocimiento ($\overline{INTA}$) y retransmitirlas hacia los bloques apropiados en el CPI.

2.2B Modos de Operación y Esquemas de Prioridad.

El CPI 8259A acepta dos tipos de palabras de control generadas por el CPU:

1) Comandos de inicialización (ICW's): Antes de que la operación normal pueda iniciarse, cada CPI en el sistema debe ser llevado a un estado inicial por una secuencia de 2 a 4 bytes, que son almacenados en sus registros de control.

2) Comandos de operación (OCW's): Estas son palabras de control que indican al CPI el modo de interrupción en que debe operar. Estos modos son:

a) Modo de Anidamiento Total:

Este modo soporta una estructura de interrupciones de nivel múltiple, en la cual el orden de prioridades de las ocho entradas IRQ está arreglado de menor a mayor prioridad. Después de inicializar al CPI IR0 tiene la mayor prioridad e IR7 tiene la más baja.

b) Modos de prioridad rotatoria:

Rotación Automática: Es utilizada en aplicaciones donde existen dispositivos de igual prioridad. En este modo, un dispositivo que ha sido atendido recibe la prioridad más baja, un nuevo dispositivo que solicite ser atendido, tendrá que esperar, en el peor de los casos, hasta que otros siete dispositivos hayan sido atendidos al menos una vez.

Rotación Especifica (Prioridad Especifica):

Las prioridades pueden ser alteradas explicitamente, programando el nivel con la prioridad más baja, quedando automáticamente fijadas las prioridades de los demás niveles. Los cambios de prioridad pueden realizarse durante un comando de fin de interrupción (EOI).

C) Modo especial de filtrado:

Filtros de Interrupción: Cada linea o nivel de interrupción puede filtrarse individualmente con el IMR sin alterar el estado de las demás lineas. En algunas aplicaciones se requiere que una rutina de atención a interrupciones altere en forma dinámica la estructura de prioridad del sistema de interrupciones, este esquema es conocido como modo especial de filtrado. En este modo, el *software* se encarga de controlar y seleccionar los canales de interrupción que estarán activos; esto se logra inhibiendo interrupciones posteriores del mismo canal y habilitando las interrupciones de todos los demás canales.

d) Modo de poleo:

En este modo la linea de interrupción (INT) no es utilizada, es decir, no se generan interrrupciones; un método alternativo consiste en apagar el bit IF del registro de estado del procesador, deshabilitándose de esta forma la recepción de interrupciones. El CPI interpreta la primera señal de lectura que recibe después de ser programado en el modo de poleo como un reconocimiento de interrupción, en ese momento enciende el bit correspondiente en el ISR si existe una petición de interrupción.

El byte enviado al canal de datos al finalizar la secuencia anterior tiene el siguiente formato:

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
I	-	-	-	-	w2	w1	w0

donde W_0-W_2 representan el código binario de la linea de interrupción que solicita ser atendida y tiene la prioridad más alta. El bit I es igual a "1" si existe una interrupción.

e) Modos de fin de interrupción:

Fin de interrupción (EOI): El bit que indica cual linea está siendo atendida (ISR) puede apagarse, ya sea en forma automática después del flanco de subida del segundo pulso de reconocimiento de interrupción ($\overline{INTA}$), o bien, mediante una palabra de control que debe ser enviada al CPI antes de terminar la rutina de atención a interrupciones (Comando EOI). Existen dos tipos de comandos EOI, los específicos y los NO específicos. Cuando el CPI 8279A se opera en aquellos modos que preservan la estructura de anidamiento total, se puede determinar cual bit de ISR debe ser apagado con el comando EOI actual; sin embargo, cuando se utilizan comandos EOI NO específicos, el CPI automáticamente apagará el bit de mayor prioridad del ISR, tomando en cuenta sólo aquellos que están encendidos, debido a que con el modo totalmente anidado, el

bit en ISR de mayor prioridad es necesariamente el último que fue reconocido y atendido. Cuando se trabaja en un modo en el que se altera la estructura de anidamiento total, el CPI no es capaz de determinar cual es el nivel de interrupción al que corresponde el último reconocimiento. En casos como este, debe utilizarse un comando EOI especifico, ya que debe incluirse en el comando el código del nivel de interrupción que debe ser apagado en el ISR.

Fin de Interrupción Automático (AEOI): En este modo, el CPI realizará automáticamente un comando EOI NO especifico con el flanco de subida del segundo pulso de reconocimiento.

2.2c ESTRUCTURA DE LAS PALABRAS DE CONTROL.

Comandos de inicialización: Siempre que se envia un comando con la línea de direcciones A_0=0 y la línea de datos D_4=1, éste es interpretado como el comando de inicialización ICW_1.

Comandos de inicialización (ICW):

ICW_1, ICW_2: En un sistema basado en procesadores 8086/88 los cinco bits más significativos del vector de interrupción son seleccionados por el usuario, mientras que los tres menos significativos son insertados por el CPI.

ADI: No tiene efecto.

LTIM: Indica el modo de interrupción por nivel. SNGL: Indica si existe más de un CPI en el sistema.

IC_4: Indica que es necesario leer el comando (ICW_4).

ICW_3: Este comando es leido únicamente cuando existe más de un CPI en el sistema y operan en cascada. Las funciones que cumple son:

a) En el controlador maestro un *"uno"* es colocado por cada esclavo en el sistema.

b) En el controlador esclavo los bits b_0-b_2 identifican al esclavo. El esclavo compara su entrada de cascada con estos bits y si son iguales libera un byte en el canal de datos.

ICW4: Se utiliza para programar las configuraciones posibles en una arquitectura de controladores múltiples:
SFNM: Programación del modo de anidamiento total especial.
BUF: Usado en el diseño de sistemas muy grandes que requieren de configuraciones maestro-esclavo.
M/S: Indica si el controlador es un maestro o un esclavo.
AEOI: Programa el fin de interrupción automático.
μPM: Selecciona el tipo de CPU con el cual interactuará el controlador.

Comandos de operación (OCW):

Durante la operación normal del CPI una selección de algoritmos puede ordenar al 8259A que opere en alguno de varios modos posibles.

OCW1: Esta palabra de operación enciende y/o apaga los bits en el registro de filtrado de interrupciones (IMR).

OCW2: Los bits R, SL y EOI seleccionan los modos de rotación y fin de interrupción, así como las combinaciones entre ellos; los bits L_0-L_2 determinan cual linea de interrupción se activará cuando el bit SL está activo.

OCW3: El bit ESMM se usa para habilitar el modo de filtrado especial. El bit SMM se usa para iniciar o terminar la operación bajo el esquema de filtrado especial.

2.3 El Canal de Expansión de la computadora personal.

El corazón de la computadora es, el microprocesador 1APX 86/88. El microprocesador se comunica con su medio ambiente mediante señales eléctricas que envía y recibe en sus terminales. Si por ejemplo, el procesador desea leer el contenido de la localidad de memoria ubicada en la dirección $00012D_{16}$, colocará este valor en las terminales de direcciones, después emitirá otras señales, las de control, en otras terminales, con lo cual indicará que desea leer el contenido de una memoria. La lógica de soporte decodificará la memoria correspondiente y el resultado será puesto en las terminales de datos para que pueda ser cargado en alguno de los registros.

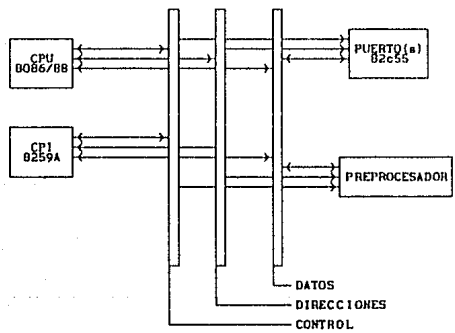
Todo el conjunto de señales eléctricas necesarias para realizar operaciones como la anterior y otras de distinta naturaleza, son agrupadas de acuerdo al propósito o uso específico que se les da. La clasificación que se hace determina tres grupos que consisten en a) señales de datos, b) señales de direcciones y c) señales de control.

Canal de Datos: Está formado por las señales de datos, es bidireccional y se utiliza para transferir datos entre el procesador y su medio ambiente. En función del procesador específico del que se trate puede tener 8 o 16 bits (D_0-D_7 o D_0-D_{15}).

Canal de Direcciones: En este grupo se encuentran las señales usadas para direccionar a la memoria y a los distintos dispositivos, tanto del sistema como periféricos. Es usado principalmente por la lógica de soporte y de decodificación. Está compuesto por 20 líneas (A_0-A_{20}) que permiten direccionar hasta 1 MB de memoria y 64 K puertos.

Canal de Control: Con esta denominación se abarca a todas las señales usadas para soportar la sincronización de los elementos de la arquitectura y la ejecución de las instrucciones propias del procesador. Está formado por las señales de lectura/escritura a memoria y a periféricos, de interrupción (7 en total), de acceso directo a memoria (DMA), de sincronia y temporización, etc.

Para interactuar directamente con el *hardware* del sistema es necesario tener acceso a todas las señales mencionadas en los párrafos anteriores. Para facilitar este proceso la computadora personal cuenta con una ranura de expansión, en la cual están comprendidos el canal de datos, el de direcciones y el de control, más algunas señales de polarización y otras reservadas por IBM. A todo este conjunto se le conoce como "Canal de Expansión". En la gráfica siguiente se detalla la distribución de las señales en la ranura de expansión:



SISTEMA CON COMPONENTES MINIMOS.

III. ANALISIS, DISEÑO E IMPLEMENTACION DEL SISTEMA DIGITAL DE MEDICION.

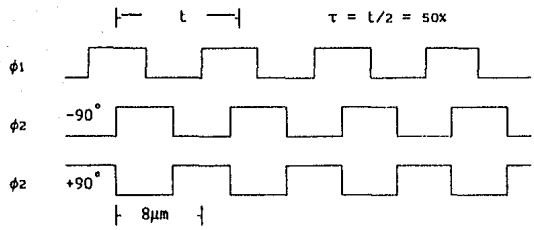
3.1 DISEÑO LOGICO DEL SISTEMA DE PREPROCESAMIENTO.

3.1A ANÁLISIS DE LAS SEÑALES DE ENTRADA.

Los sensores seleccionados son del tipo de los codificadores ópticos incrementales, tienen una resolución de 2 millonésimas de metro (micras) y una presición de 4 micras por metro. Estos codificadores consisten de una escala de presición y una cabeza optoelectrónica de lectura de NO contacto. La escala, de 1.1 metros de longitud, se encuentra finamente dividida en miles de líneas verticales paralelas, las cuales están formados con depósitos de un material opaco (Inconel) sobre una sección de vidrio transparente, usado por su bajo coeficiente de expansión térmica. La cabeza de lectura traduce el movimiento lineal en señales cuadradas de nivel TTL, con un defasamiento constante de 90° y ancho de pulso o ciclo de trabajo τ de 50%.

La relación de fase que tienen las señales es de 90° de adelanto o atraso relativo. Por convención se denotarán ambas señales por ϕ_1 y ϕ_2, donde ϕ_1 es la señal de referencia. Un defasamiento de $+90^{\circ}$ entre ambas señales implicará un "*desplazamiento positivo*", mientras que un defasamiento de -90° implicará un "*desplazamiento negativo*".

Para satisfacer las especificaciones iniciales, las reglas usadas tienen una resolución de dos micras. En la gráfica siguiente se presenta un ejemplo de las formas de onda típicas de las señales ϕ_1 y ϕ_2.



La frecuencia de las señales es la misma y es directamente proporcional a la velocidad de desplazamiento de la articulación sobre la cual se encuentra montada la regla, así como a la resolución de esta misma; es decir, a mayor velocidad la frecuencia de las señales es mayor. Si se incrementa la resolución de los sensores, la frecuencia de las señales se verá incrementada aún cuando se tenga una velocidad de desplazamiento constante, ya que está dada por la duración del pulso (t). Esta característica determina el espectro de velocidades en el cual el sistema puede operar correctamente, estableciéndolo como una función de la rapidez de los cambios de posición y de los tiempos de respuesta del algoritmo implementado en *hardware* y *software*.

3.1B OBTENCIÓN DE LAS FUNCIONES LÓGICAS PARA EL PROCESAMIENTO DE LAS SEÑALES DE ENTRADA.

De las gráficas anteriores se puede determinar la relación existente entre la distancia recorrida y el número de flancos ocurridos en ambas señales. En el caso de un "*desplazamiento positivo*" (+90° de defasamiento) se tiene una secuencia mixta de transiciones o flancos generados en ambas señales ($\phi 1$ y $\phi 2$) con el orden siguiente:

- $\phi 1_{\uparrow}$  flanco de subida
- $\phi 2_{\downarrow}$ flanco de bajada
- $\phi 1_{\downarrow}$  flanco de bajada
- $\phi 2_{\uparrow}$  flanco de subida

La secuencia anterior es ciclica y es excluyente con respecto a la secuencia obtenida durante un "*desplazamiento negativo*". La secuencia obtenida en este caso es la siguiente:

- $\phi 1_{\uparrow}$  flanco de subida
- $\phi 2_{\uparrow}$  flanco de subida
- $\phi 1_{\downarrow}$  flanco de bajada
- $\phi 2_{\downarrow}$  flanco de bajada

donde $\phi_{\uparrow}$ es un flanco de subida y $\phi_{\downarrow}$ es un flanco de bajada.

Si $\phi 1$ y $\phi 2$ representan a las señales de la regla en un estado lógico alto o "*uno*" lógico, y $\overline{\phi 1}$ y $\overline{\phi 2}$ representan a las mismas señales en un estado lógico bajo o "*cero*" lógico, entonces las ecuaciones que describen la generación de las secuencias de pulsos para un "*desplazamiento positivo*" y para un "*desplazamiento negativo*" respectivamente son:

"Desplazamiento positivo":

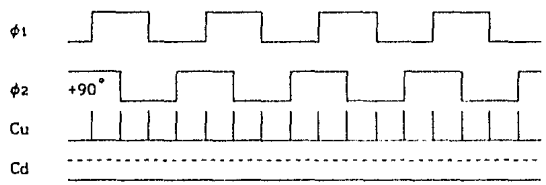
$$Cu = \phi 1_{\uparrow}\phi 2 + \phi 1\phi 2^{\downarrow} + \phi 1^{\downarrow}\overline{\phi 2} + \overline{\phi 1}\phi 2_{\uparrow} \qquad (3)$$

"Desplazamiento negativo":

$$Cd = \phi 1_{\uparrow}\overline{\phi 2} + \phi 1\phi 2_{\uparrow} + \phi 1^{\downarrow}\phi 2 + \overline{\phi 1}\phi 2^{\downarrow} \qquad (4)$$

De las gráficas de las señales se puede observar que hay una diferencia de 180° entre una señal $\phi 2$ debida a un "*desplazamiento positivo*" y la misma señal debida a un "*desplazamiento negativo*", esta diferencia de 180° se ve reflejada en las relaciones encontradas para Cu y Cd, ya que para obtener una ecuación a partir de la otra, sólo se necesita complementar lógicamente a la señal $\phi 2$.

Una vez obtenidas las expresiones para Cu y Cd se obtienen las siguientes señales para el caso de un "*desplazamiento positivo*".



3.1C IMPLEMENTACIÓN LÓGICA DEL SISTEMA DIGITAL DE PREPROCESAMIENTO.

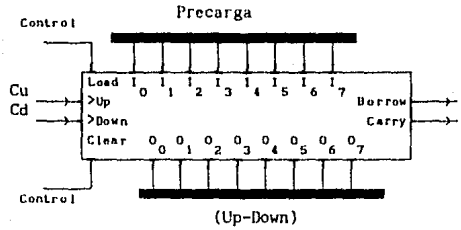
El algoritmo propuesto para medir los desplazamientos está descrito por la expresión (1).

$$\delta_r = [\ (ND - NI)\ (2^{n-1}) + (D - I)]\ *\ R \qquad (5)$$

El término (D-I) especifica el valor de los contadores de pulsos establecidos en *hardware*. Este término puede descomponerse en dos términos de acuerdo al procedimiento siguiente:

Se utiliza un contador binario bidireccional (*Up-Down*) de ocho bits para contabilizar los pulsos obtenidos de las señales determinadas por Cu y Cd (2 y 3). Este contador tiene dos terminales de reloj (*Up* y *Down*), una terminal de restablecimiento o puesta a *"cero"* (*Reset*) y dos terminales de sobreflujo o exceso (*Carry* y *Borrow*).

Las señales Cu y Cd se conectan a las terminales *Up* y *Down* respectivamente, con la intención de que el circuito contabilice los desplazamientos ocurridos en el rango de 2^8 pulsos (256 pulsos o 512 micras con la resolución de las reglas utilizadas). Las señales de sobreflujo se generan cuando se ha excedido el número máximo que se puede representar, al contar hacia arriba se genera un *Carry* cuando se llega al número 255, al contar hacia abajo se genera un *Borrow* cuando se llega a *"cero"*.

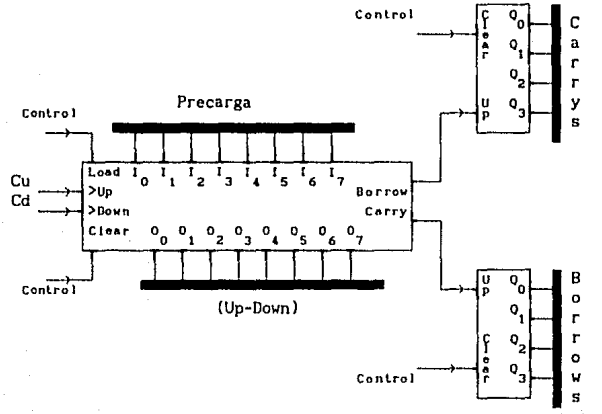


CONTADOR BINARIO ASCENDENTE/DESCENDENTE DE 8 BITS

Posteriormente se contabilizan por separado las señales de sobreflujo usando dos contadores binarios de 4 bits conectados en cascada con el contador binario de 8 bits, en otras palabras, se implementan contadores de *Carrys* y de *Borrows*. Al utilizar estos contadores la expresión inicial que incluía los contadores de *hardware* puede reescribirse de la siguiente forma:

$$(D - I) \rightarrow (Up - Down) + (Carry - Borrow)*(256) \qquad (6)$$

Donde (*Up - Down*) es el valor que existe en el contador binario y que representa la diferencia entre los desplazamientos positivos y negativos ocurridos en un intervalo menor de 256 pulsos. Por otro lado, (*Carry - Borrow*) es la diferencia de los sobreflujos ocurridos durante los desplazamientos positivos y negativos, iguales o mayores que el máximo de 256 pulsos. Dado que un sobreflujo representa la ocurrencia de 256 pulsos en una u otra dirección, es necesario agregar el término constante (256) que multiplica a la diferencia de sobreflujos para obtener el número total de pulsos ocurridos a lo largo de la articulación en un rango de 12 bits.



CIRCUITO CONTABILIZADOR DE 'CARRYS' Y 'BORROWS'

Hasta este punto se han desarrollado los contadores de pulsos en *hardware* para los dos sentidos de desplazamiento sobre el eje, sin embargo, de acuerdo con el algoritmo utilizado, es necesario generar una interrupción para transferir información a la capa de *software* del sistema, de tal forma que se actualicen los contadores o variables del programa. Para generar estas interrupciones se utiliza el bit más significativo del contador de 4 bits.

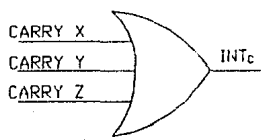
Desde el punto de vista del *software*, con el esquema anterior se tiene implementado un contador de 12 bits para cada sentido de desplazamiento. Tomando en cuenta que se usan reglas con una resolución de 2 micras, la expresión (1) puede rescribirse como:

$$\delta_{\Gamma} = [(ND-NI)(2^{n-1})+(Carry-Borrow)(256)+(Up-Down)](R) \quad (7)$$

$$\delta_{\Gamma} = [(ND-NI)(2048)+(Carry-Borrow)(256)+(Up-Down)](2) \quad (8)$$

Donde δ_{Γ} es el desplazamiento total sobre la articulación en micras, (2048) es el orden o peso del bit más significativo del contador (n = 12 bits), y por tanto, es el valor asociado a cada interrupción; (ND - NI) es la diferencia de las interrupciones generadas por desplazamientos positivos y negativos.

A las interrupciones generadas por desplazamientos en el eje X se les denominará Cx y Bx, dependiendo del sentido del desplazamiento; las interrupciones generadas en los demás ejes se denominarán Cy, By, Cz y Bz respectivamente. Con el fin de optimizar el número de líneas de interrupción usadas, se realiza un agrupamiento de interrupciones de acuerdo al sentido del desplazamiento que las genera: se agrupan Cx, Cy y CZ, mientras que por otro lado se agrupan Bx, By y Bz, después se realiza la suma lógica con las señales pertenecientes a un mismo grupo para obtener la señal de interrupción que será enviada a la computadora.



INTERRUPCIONES POR CARRYS

Una vez generadas las interrupciones, es necesario indicar a la computadora cual de los ejes está interrumpiendo. El sentido del movimiento queda automáticamente determinado por el tipo de interrupción que se genere. Cuando se presenta una interrupción del tipo INTc, por ejemplo, se debe a la realización de traslaciones positivas o *carrys*.

Una vez que el microprocesador ha reconocido la interrupción, debe actualizar los contadores o variables correspondientes. Los bits utilizados para generar las interrupciones se ordenan en un registro, con el fin de obtener un código que pueda utilizarse para localizar las variables. El registro tiene una longitud de 8 bits con la estructura siguiente:

Registro de Código de Movimiento							
7	6	5	4	3	2	1	0
*	*	*	*	*	x	y	z

El valor de los 5 bits más significativos puede fijarse elctrónicamente o filtrarse por el programa, ya sea durante los accesos al registro de código de movimiento, o bien, durante la decodificación del valor contenido en el mismo.

El concepto que existe detrás del valor contenido en el registro del código de movimiento es semejante al concepto de vector de interrupción asociado con el controlador de interrupciones de la computadora personal IBM. Cuando se genera una interrupción por *carry* (INTc) o por *borrow* (INTB), se libera el vector correspondiente en el canal de datos. Este vector determina cual es la rutina que debe atender la interrupción. Este concepto puede extenderse y ampliarse al valor contenido en el registro de código de movimiento; este valor se usa para determinar cual es el eje o ejes que interrumpen, por lo tanto, puede utilizarse para ejecutar las rutinas asociadas a cada eje.

Los códigos de movimiento para interrupciones INTc e INTB, se muestran a continuación:

TABLA 3.1

C7-3	Cx	Cy	Cz	Ejes que interrumpen
*	0	0	0	No definido
*	0	0	1	z
*	0	1	0	y
*	0	1	1	z,y
*	1	0	0	x
*	1	0	1	z,x
*	1	1	0	y,x
*	1	1	1	z,y,x

TABLA 3.2

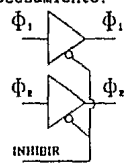
B7-3	Bx	By	Bz	Ejes que interrumpen
*	0	0	0	No definido
*	0	0	1	z
*	0	1	0	y
*	0	1	2	z,y
*	1	0	0	x
*	1	0	1	z,x
*	1	1	0	y,x
*	1	1	1	z,y,x

Las rutinas de atención a las interrupciones de *hardware* deben accesar los códigos anteriores e interpretarlos, de tal forma que los valores medidos de los desplazamientos sean actualizados correctamente en forma simultánea y en tiempo real, con el objeto de observar la variación en el desplazamiento en todos los ejes. La decodificación de estos valores es una tarea que lleva a cabo el programa de control del sistema.

3.2 Implementación física del preprocesador.

3.2a Estructura de los contadores de pulsos y generadores de interrupciones.

Las señales de las reglas son transferidas a la tarjeta de procesamiento por medio de cables; generalmente se utilizan cables trenzados y blindados para evitar la inducción de ruido, sin embargo, suele presentarse un efecto de atenuación de la amplitud, debido principalmente a la longitud de las líneas de transmisión. Para reacondicionar la señal se utilizan circuitos de alta impedancia (tercer estado), con lo cual se puede aislar el dispositivo sensor del dispositivo procesador. Además los circuitos de alta impedancia con terminal de habilitación le dan al sistema la facilidad de controlar, dependiendo de las necesidades, el paso de las señales desde las reglas hasta el sistema de preprocesamiento.



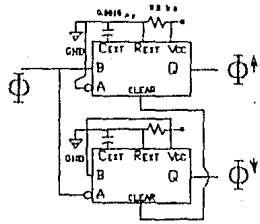
circuito habilitador

La implementación física de las expresiones encontradas para las secuencias de pulsos Cu y Cd, puede realizarse con compuertas lógicas AND y OR. Sin embargo, para poder obtener los productos que describen estas ecuaciones, es necesario obtener antes las señales correspondientes a los flancos positivos y negativos de las señales $\phi 1$ y $\phi 2$. Para obtener un pulso a partir de un flanco se utiliza un circuito que detecta una transición de nivel en su terminal de disparo y en correspondencia genera un pulso de salida, cuya duración puede programarse mediante componentes externos.

Un circuito monoestable es un dispositivo que, cuando se "excita" o se "dispara" mediante una señal, produce un pulso de salida cuya duración es independiente de la duración del pulso de disparo. La duración del pulso de salida puede programarse usando una red Resistor-Capacitor, ya que está en función de la constante de tiempo RC de la red.

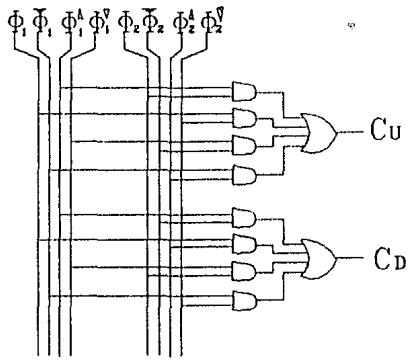
En el caso específico de las señales ϕ_1 y ϕ_2 es necesario generar un pulso cuando ocurren flancos positivos y negativos. Entre los circuitos comerciales de la familia TTL, que pueden realizar esta tarea, se encuentra el 74LS123. Este dispositivo integrado contiene dos monoestables con transición de disparo programable.

Cada dispositivo, como el (74LS123) mencionado, tiene tres entradas que permiten la elección de disparo o activación por flancos positivos o negativos. La terminal A es una entrada de disparo para transiciones negativas (flanco de bajada), mientras que la terminal B es una entrada de disparo para transiciones positivas (flancos de subida). La terminal *Clear* provoca que el pulso de salida termine en el momento en que esta señal se activa, independientemente de la constante de tiempo determinada por los componentes externos. La tabla de funciones que indica los modos de operación del dispositivo 74LS123, proporciona la información para realizar las conexiones que produzcan las señales deseadas.



La red resistor capacitor está diseñada para producir pulsos con una duración de t = 3.7185 μs, este ancho de pulso es suficientemente pequeño como para no afectar la operación del sistema a velocidades de desplazamiento, en la articulación lineal, mayores a 25 cm/seg.

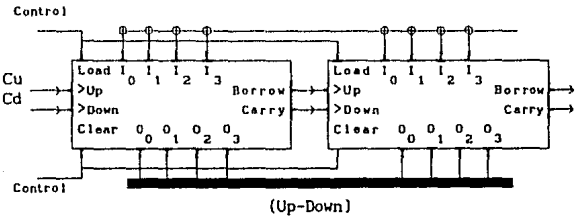
El circuito anterior se repite para ϕ_1 y ϕ_2, de esta forma obtenemos todas las secuencias de pulsos necesarias para realizar las operaciones indicadas en las expresiones de Cu y Cd (3, 4). Estas ecuaciones están formadas por una suma de productos, la cual puede realizarse con las funciones lógicas AND y NOR usando los circuitos 74LS08 y 14002B, en la forma que se muestra a continuación:



Con los circuitos anteriores se obtienen dos señales consistentes en secuencias de pulsos, cada uno representa una traslación de dos micras. La señal correspondiente al sentido de traslación no usado se mantiene en un nivel lógico alto; la señal del sentido de traslación usado incluye una serie de pulsos de nivel lógico bajo o *"ceros"*.

Para contabilizar los pulsos anteriores se utilizan dos contadores binarios de 4 bits conectados en cascada para formar un solo contador de 8 bits. El dispositivo 74LS193 tiene la configuración idónea para realizar la conexión anterior, ya que los niveles lógicos requeridos en sus terminales de reloj, son compatibles con los obtenidos en la implementación de las ecuaciones (3) y (4), además de que su capacidad de conteo ascendente y descendente es compatible con el algoritmo de procesamiento propuesto.

El circuito mencionado anteriormente representa la realización física del término simple de la ecuación (6), es decir, se trata de la primera etapa electrónica de los contadores implementados en *hardware*, (up-down).



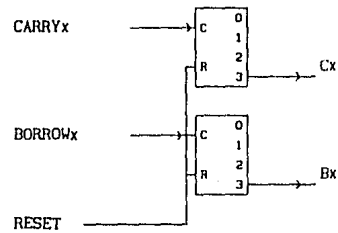
(Up-Down)

CONTADORES DE 4 BITS CONECTADOS EN CASCADA PARA FORMAR UN CONTADOR BINARIO ASCENDENTE/DESCENDENTE DE 8 BITS

Debido a la facilidad del circuito anterior para realizar cuentas en forma ascendente y descendente, se realiza en forma automática la diferencia entre las traslaciones positivas y negativas de la articulación lineal, sin utilizar circuitos aritméticos adicionales. El contador de 8 bits es funcionalmente idéntico al circuito de 4 bits, por lo que continúa generando las señales del *Carry y Borrow*.

El contador binario de 8 bits puede contabilizar hasta 512 micras de desplazamiento (256 pulsos) en cualquiera de los dos sentidos de traslación sobre un eje. Se pueden utilizar las señales de *Carry y Borrow* para separar la cuenta de pulsos en una etapa posterior. El objetivo de esta última etapa es proporcionar una función de preescalamiento adicional, además de servir junto con las etapas correspondientes a los demás ejes, como un registro de código de movimiento. Este registro se forma a partir de los bits más significativos de esta segunda etapa, por lo que tiene un total de 6 bits, y se usa para generar las interrupciones y decodificarlas. La función de preescalamiento de esta etapa es útil porque desacopla el contador de 8 bits, permitiendo que exista un tiempo de retraso de 256 pulsos al menos, durante el reconocimiento y decodificación de la interrupción.

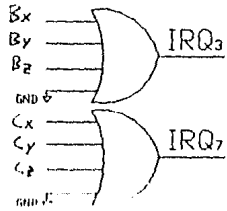
La segunda etapa de conteo (preescalamiento) se desarrolla en base a los contadores binarios CMOS 14520, cada uno de ellos cuenta el número de *Carrys* o de *Borrrows*, según sea el caso.



CIRCUITOS CONTADORES DE CARRYS Y BORROWS

Cada uno de los dispositivos anteriores (24LS193 y 14520) cuenta con una terminal de *Reset* o *Clear*, usadas para poner a *cero* el circuito contador. Estas terminales se emplean para desarrollar el algoritmo de atención a interrupciones y la secuencia de inicialización de cada uno de los ejes. Las señales enviadas a estas terminales se generan mediante el *software*, y se envían a través de un puerto usado para controlar el sistema.

Finalmente, el registro de código de movimiento se usa para generar las interrupciones en la forma descrita en la sección 3.1c. Se emplean dos compuertas NOR de cuatro entradas, el circuito utilizado es el 14002B y las conexiones son las que se muestran:

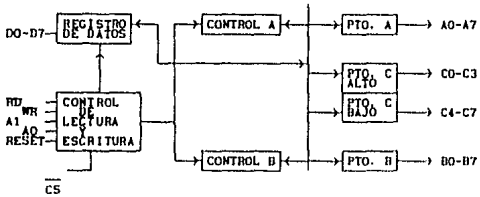


GENERACION DE INTERRUPCIONES

Las dos señales de interrupción obtenidas se conectan a dos de las líneas de interrupción disponibles en el canal de expansión. Todas estas líneas son las interrupciones de IRQ1 a IRQ7, pero se usan IRQ3 e IRQ7 ya que están reservadas para el puerto paralelo y el segundo puerto serial de comunicaciones. Dichas líneas se seleccionaron porque no son indispensables para la realización de la tareas más usuales de la computadora, como es el caso de las líneas de video, teclado, timer y disco.

3.2B Descripción del Puerto Paralelo Intel - 8255.

El 82C55A es una interfaz programable para periféricos, diseñado para usarse en los sistemas de microcomputadoras de Intel. Su función es la de una componente de entrada/salida de propósito general, que constituya la interfaz entre equipo periférico y el canal de datos de la microcomputadora. La configuración funcional del 82C55A está programada por el *software*, de modo que normalmente no se requiere de lógica externa para crear la interfaz con estructuras o dispositivos periféricos.



CONFIGURACION INTERNA DEL PUERTO 82C55.

Registro del Canal de Datos:

Este registro bidireccional de tres estados y 8 bits se usa como interfaz entre el 82C55A y el canal de datos del sistema. El registro transmite o recibe datos mediante la ejecución de instrucciones de entrada o salida provenientes del CPU. Además, por medio del registro se transfieren palabras de control e información de estado.

Lógica de Lectura/Escritura y de Control:

La función de este bloque es el manejo de todas las transferencias internas y externas, tanto de los datos como de las palabras de control y de estado. Acepta entradas de las direcciones del CPU y canales de datos de control y en correspondencia envía comandos a los grupos de control A y B.

Grupos de Control A y B:

La configuración funcional de cada puerto se programa mediante *software*. Básicamente, el CPU envía al 82C55A una palabra de control cuya información inicializa la configuración.

Cada uno de los grupos de control (A y B) aceptan "comandos" de la lógica de control de Lectura/Escritura, reciben "palabras de control" del canal de datos interno y proporcionan el comando adecuado a sus puertos correspondientes.

Grupo de Control A- Puerto A y Puerto C superior (C7-C4)
Grupo de Control B- Puerto B y Puerto C inferior (C3-C0)

Puertos A, B y C:

El 82C55A contiene tres puertos de 8 bits (A, B y C). Todos se pueden configurar en una amplia variedad de características funcionales mediante el *software*, pero cada uno tiene características particulares.

Puerto A: Un registro para salida de datos de 8 bits y otro de entrada de 8 bits.

Puerto B: Un registro para entrada/salida de datos de 8 bits.

Puerto C: Un registro para salida de datos y un canal para entrada de datos de 8 bits. Este puerto puede dividirse en dos puertos de cuatro bits bajo el modo de control. Cada puerto contiene un registro de 4 bits y puede usarse para salidas de señales de control y entradas de señales de estado, junto con los puertos A y B.

Descripción operacional del 82C55A.

Hay tres modos básicos de operación que pueden seleccionarse por *software:* Los modos para los puertos A y B se pueden definir separadamente, mientras que el puerto C se divide en dos porciones, según lo requieran las definiciones de los puertos A y B. Todos los registros de salida, incluyendo los *flip-flops* de estado, pueden ponerse a *"cero"* cada vez que el modo cambie. Los modos pueden combinarse de tal forma que su definición funcional puede extenderse a casi cualquier estructura de Entrada/Salida.

Cualquiera de los ocho bits del puerto C puede ponerse en *"cero"* o en *"uno"* usando una sola instrucción de salida proveniente del procesador. Esta característica reduce los requerimientos de *software* en las aplicaciones basadas en Control.

MODO CERO (Entrada/Salida básico): Esta configuración funcional proporciona operaciones de entrada y salida simples para cada uno de los tres puertos. No se requiere un protocolo de sincronización, los datos simplemente se escriben o leen de algún puerto específico.

Definiciones funcionales básicas del modo 0:

- Dos puertos de 8 bits y dos puertos de cuatro bits.
- Cualquier puerto puede ser una entrada o salida.
- Las salidas se conservan en un registro.
- En este modo hay 16 posibles configuraciones diferentes de entrada/salida.

MODO 1 (Entrada/Salida sincronizada): Esta configuración funcional constituye un medio para transferir datos entre E/S y un puerto determinado, en conjunto con señales de sincronía o protocolo.

Definiciones funcionales básicas del modo 1:

- Hay dos grupos (Grupo A y Grupo B).
- Cada grupo contiene un puerto de 8 bits de datos y un puerto de cuatro bits de control de datos.
- Los puertos de 8 bits de datos pueden ser tanto entrada como salida, y ambas se conservan en un registro.
- El puerto de 4 bits se usa para control y estado de el puerto de datos de 8 bits.

MODO 2 (Canal bidireccional síncrono de Entrada/Salida): Esta configuración funcional proporciona una manera de comunicarse con un dispositivo periférico o estructura mediante un canal simple de 8 bits, ya sea para transimisión o recepción. Se proporcionan señales de sincronía para mantener una disciplina apropiada de control de flujo en el canal, de una forma semejante a la ocurrida en el modo 1. También hay disponibles funciones de generación, habilitación e inhibición de interrupciones.

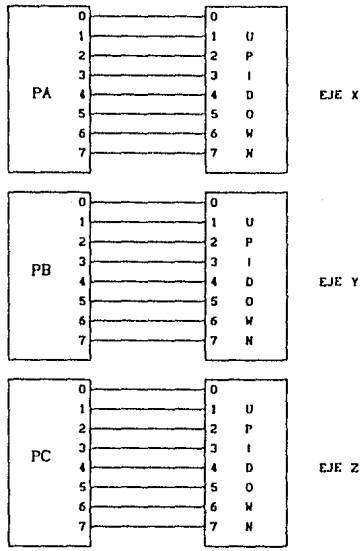
3.2c INTERFAZ DE ACCESO Y CONTROL DEL SISTEMA.

Para accesar la información de la etapa de preprocesamiento se utiliza una interfaz, que le permite a la computadora comunicarse con el medio ambiente. Mediante la interfaz es posible enviar hacia el exterior (equipo periférico) datos provenientes del procesador o de la memoria, y también es posible que la información generada en el exterior sea llevada hacia el procesador o hacia la memoria principal del sistema, usando técnicas como la de transferencia de acceso directo a memoria (DMA).

Los dispositivos que permiten realizar las operaciones anteriores son conocidos como "puertos", y están constituidos por un arreglo de registros de entrada/salida controlados por otro(s) registro(s) de control y/o estado. A cada puerto se le asigna una dirección base dentro del mapa de Entrada/Salida, la cual se utiliza para identificarlo entre todos los demás existentes en un sistema. Esta dirección se usa para accesar el puerto, y todos sus registros. El acceso a estos registros se realiza mediante operaciones de lectura y escritura.

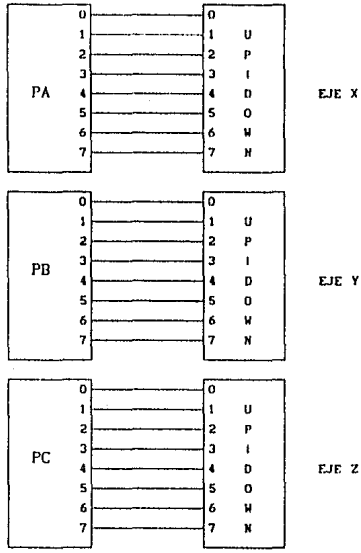
En la sección 3.2b se describió la estructura del circuito INTEL 8255, que está diseñado para servir como medio de comunicación entre un sistema, basado en un microprocesador y su medio ambiente. Para construir la interfaz de acceso y control se utilizan tres circuitos de este tipo. Los puertos se usan para controlar los contadores, el registro virtual de movimiento y el circuito de reacondicionamiento de las señales provenientes de las reglas; además permiten accesar los bits de los contadores y el código de interrupción del registro.

El valor de los contadores 74LS193, es decir la diferencia (*Up-Down*), se lee usando un puerto del circuito 82C55, para cada uno de los ejes. Los contadores del eje x se leen por el puerto A, los de Y por el puerto B, y los de Z por el puerto C. Para programar que cada puerto sea un puerto de entrada, se utiliza la palabra de control $9B_{16}$.



LECTURA DE LOS CONTADORES (UP-DOWN)

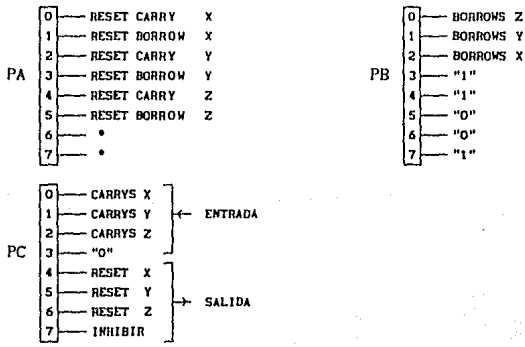
Para leer la información de los contadores de *Carrys y Borrows* se utiliza otro circuito 82C55, también programado con los tres puertos de entrada, la asignación de puertos sigue siendo la misma. Los cuatro bits menos significativos del puerto A leen el contador de *Borrows* de X, mientras que por los cuatro bits más significativos se lee el contador de *Carrys*. Los puertos B y C hacen lo mismo con los contadores de *Carrys y Borrows* de los ejes Y y Z respectivamente.



LECTURA DE LOS CONTADORES DE CARRYS Y BORROWS

La sección de control de la interfaz se implementa usando un 82C55 con una configuración de puertos distinta a las anteriores. El puerto A se utiliza como un puerto de salida cuya función es enviar las señales necesarias para controlar los contadores de *Carrys y Borrows*. El puerto B se usa para leer el registro virtual de código de movimiento; los tres bits menos significativos están conectados a los bits más significativos de los contadores de *Borrows*; B_0 está conectado a la línea de interrrupción del eje Z, B_1 a la línea de interrupción del eje Y, y B_2 a la línea de interrupción del eje X. Los demás bits (B_3 - B_7) se encuentran alambrados para formar un valor base del código o vector base igual a 98_{16}.

El puerto C está dividido en dos puertos de 4 bits, los cuatro menos significativos se configuran como un puerto de entrada y se usan de la misma forma que los bits menos significativos del puerto B, pero conectándolos en este caso a los bits más significativos de los contadores de *Carrys*; C_0 está conectado a la línea de de interrupción del eje Z, C_1 al eje Y y C_2 al eje X. El bit C_3 está alambrado para formar un vector base cuyos bits menos significativos se encuentren en el rango 0 -7. Los cuatro bits más significativos del puerto C están programados como un puerto de salida; estos bits se utilizan para controlar los contadores *(Up-Down)* de cada eje. Mediante estas señales se puede inicializar (poner un cero en cada bit de salida) individualmente a los contadores. El bit C_4 inicializa el contador del eje X, C_5 el del eje Y, y C_6 el del eje Z. Por último el bit C_7 se usa para habilitar y/o inhibir el paso de las señales provenientes de las reglas, por lo que se encuentra conectado a la terminal de habilitación de las compuertas de alta impedancia (tercer estado) usadas para reacondicionar estas señales; el bit C_7 se usa para habilitar las mediciones. La palabra de control usada para programar los puertos en la forma descrita anteriormente es un 83_{16}.



3.2D LÓGICA DE DECODIFICACIÓN DE LA INTERFAZ.

La lógica de decodificación de Entrada/Salida está compuesta por los circuitos que se usan para habilitar el acceso del procesador hacia algún dispositivo periférico, o bien que algún periférico tenga acceso a los canales del sistema (datos, direcciones y control). A cada dispositivo se le asigna una dirección base que sirve para identificarlo y habilitarlo. El conjunto de instrucciones de los procesadores 8088/88 incluye dos instrucciones de Entrada/salida; estas instrucciones aceptan un valor de 16 bits como una dirección, por lo tanto el mapa de decodificiación tiene un espacio de 64 K direcciones individuales.

El mapa de decodificación de la computadora personal de IBM tiene algunos segmentos reservados para los dispositivos utilizados con este equipo. En la tabla siguiente se muestran las direcciones y nombres de cada uno de los dispositivos.

MAPA DE DECODIFICACIÓN DE ENTRADA/SALIDA - IBM PC

DIRECCIÓN (Hexadecimal)	DISPOSITIVO (Reservado)
0000-000F	Transferencias DMA (8237A)
0020-0021	Controlador de Interrupciones 8259A
0040-0043	Base de tiempo (fase) 8254
0060-0063	Puerto paralelo 8255
0080-0083	Registros de página para DMA
00A0-00AF	Registro de código para NMI
0200-020F	Interfaz para juegos
0210-0217	Unidad de expansión
0220-024F	Reservado
0250-0277	No utilizado
0278-027F	Segundo puerto paralelo
0280-02EF	No usado

02F0-02F7	Reservado
02F8-02FF	Segundo puerto serie
0300-031F	Tarjeta prototipo
0320-032F	Disco duro
0330-0377	No usado
0378-037F	Primer puerto paralelo
0380-038C	Comunicaciones SDLC
0390-039F	Comunicaciones síncronas binarias (primario)
03A0-03A9	Comunicaciones síncronas binarias (secundario)
03B0-03BF	Video monocromático
03C0-03CF	Reservado
03D0-03DF	Tarjeta adaptadora de video (CGA)
03E0-03EF	Reservado
03F0-03F7	Disco flexible
03F8-03FF	Primer puerto serie
0790-0793	Adaptador 1
0B90-0B93	Adaptador 2
1390-1393	Adaptador 3
2390-2393	Adaptador 4

Las direcciones base 0680_{16}, 0700_{16} y 0780_{16} se encuentran disponibles en un rango de 15 direcciones al menos. Los tres circuitos INTEL 82C55 se decodifican usando las tres direcciones mencionadas. La asignación de direcciones es de la forma siguiente:

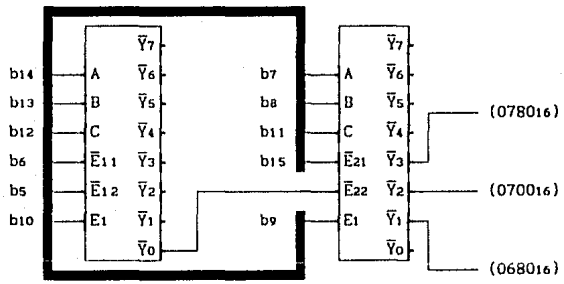
0680_{16}	Contadores (*Carrys & Borrows*)
0700_{16}	Códigos de interrupción y control del sistema
0780_{16}	Contadores (*Up-Down*)

En este sistema se utilizan los circuitos 74LS138 para decodificar los puertos. Este circuito es un decodificador de 8 líneas, cuenta con tres terminales de habilitación, dos de ellas son activas en el nivel bajo o *cero* y la otra en nivel alto o *uno* ($\overline{E1}$, $\overline{E2}$, E). Se utilizan dos circuitos como éste, conectados en cascada para decodificar los puertos. La asignación de terminales y señales está indicada en la tabla de decodificación.

TABLA 4.3																
Dirección	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0680	0	0	0	0	0	1	1	0	1	0	0	0	0	0	0	0
0700	0	0	0	0	0	1	1	1	0	0	0	0	0	0	0	0
0780	0	0	0	0	0	1	1	1	1	0	0	0	0	0	0	0
	$\bar{E}_{21}$	C_1	B_1	A_1	C_2	E_1	E_2	B_2	A_2	$\bar{E}_{11}$	$\bar{E}_{12}$					

Tabla de decodificación.

Donde A_1, B_1 y C_1 son las terminales de selección y $\bar{E}_{11}$, $\bar{E}_{12}$ y E_1 son las terminales de habilitación del decodificador 1, mientras que A_2, B_2, C_2, E_2 y $\bar{E}_{21}$ son terminales del segundo decodificador. Con la asignación de señales mostrada se selecciona la salida Yo del decodificador 1. Esta salida se usa para habilitar al decodificador 2 usando la terminal $\bar{E}_{22}$. Los bits 7 y 8 de la dirección seleccionan las salidas $\bar{Y}_1$, $\bar{Y}_2$ y $\bar{Y}_3$ del decodificador 2, que corresponden a las direcciones 0680_{16}, 0700_{16} y 0780_{16} respectivamente.



IV ESTRATIFICACION DEL SISTEMA.

En el presente capítulo se especifica la estructura general de los programas del sistema; esta estructura está desarrollada tomando como referencia el modelo de una máquina de estados finitos, de tal forma que implementa el algoritmo de procesamiento especificado en la sección 1.3, ecuación (1).

El conjunto de algoritmos desarrollados para el sistema pueden clasificarse en estratos, dependiendo de su funcionalidad o área de incidencia en la que actúan:

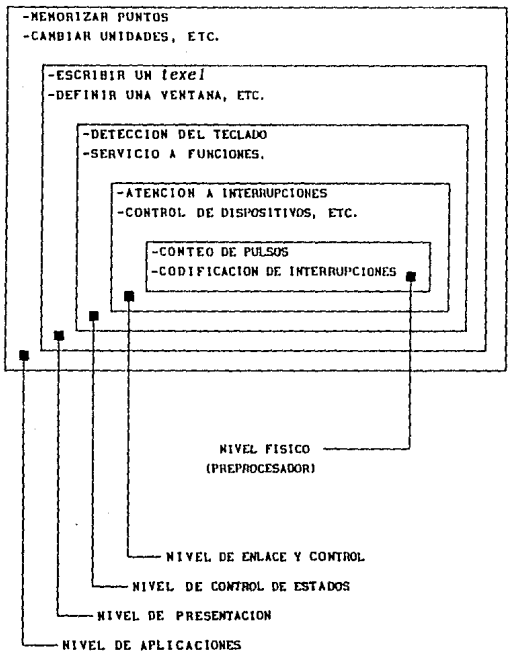
a) Nivel Físico : Comprende la etapa electrónica de preprocesamiento, es decir, los algoritmos de conteo, codificación de desplazamientos y generación de interrupciones. En los capítulos II y III se realiza el análisis y desarrollo de las componentes de este nivel.

b) Nivel de Enlace y Control de la Arquitectura : Está formado por las rutinas de atención a las interrupciones, y las que se encargan de iniciar y controlar los componentes electrónicos del preprocesador y de la tarjeta principal de la computadora.

c) Nivel de Control de Estados Lógicos : En este nivel se incluyen las rutinas con las que se implementa la arquitectura de un autómata de estados finitos. Este nivel supervisa las operaciones, es decir, se trata del administrador del sistema, sus funciones son atender y ejecutar las solicitudes de los usuarios, controlar el preprocesador y actualizar el despliegue de las lecturas de los desplazamientos, es decir, integra los servicios y funciones proporcionados por los demás niveles.

d) Nivel de Presentación : En este estrato se encuentran todas las rutinas de gestión de pantalla. Se encargan de los accesos a la memoria de vídeo, de las operaciones con ventanas y de la creación de la interfaz visual con el usuario.

e) Nivel de Aplicaciones : Esta es la capa o nivel superior, en ella se ejecutan todas las operaciones que son solicitadas por los usuarios. Las aplicaciones pueden ser una mezcla de llamadas a las rutinas de cualquiera de los niveles anteriores, o bien pueden ser algún tipo de tarea que no requiera de los niveles físico, de enlace y control, y de presentación.



4.1 DEFINICION DE LAS FUNCIONES DEL SISTEMA.

En las secciones correspondientes a la definición de entradas y salidas (1.1b), y a la propuesta de diseño (1.3) se mencionaron algunas de las características que deben implementarse con el fin de desarrollar un producto competente con aquellos existentes en el mercado de aplicaciones para metrología dimensional. El Sistema de Medición por Coordenadas es funcionalmente compatible con los equipos comerciales, pero posee características propias originadas por estar diseñado usando una computadora personal.

Las funciones que el sistema propuesto es capaz de efectuar son las siguientes:

i) Terminar la ejecución, restaurando las condiciones lógicas de los componentes del equipo (Terminar).

ii) Inicializarse mediante un comando provisto por el usuario (Reset o Inicializar Sistema).

iii) Memorizar las coordenadas de puntos en el espacio (Memorizar).

iv) Respaldar en medios magnéticos las coordenadas memorizadas (Salvar).

v) Inicializar el sistema de referencia de la medición (Inicializar).

vi) Deshabilitar y/o habilitar el sistema de medición (Habilitar).

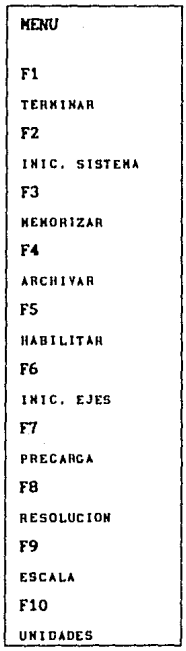
vi) Aceptar valores iniciales de desplazamiento (*offset*) para el sistema (Precarga).

viii) Ajustar la resolución de las reglas ópticas usadas (resolución).

ix) Definir el sistema de unidades de medición a utilizar (Unidades).

x) Variación de la escala de medición (Escala).

Algunas de las funciones anteriores no necesitan información adicional por lo que se ejecutan en cuanto son llamadas, sin embargo otras requieren de parámetros para determinar la componente del sistema de referencia y/o el tipo especifico de tarea que realizarán. Todas las funciones pueden ser llamadas desde cualquier parte en la estructura de estados del sistema, utilizando un comando especifico asignado previamente a cada una de ellas (las teclas de función F1..F10). En la pantalla del sistema estas funciones aparecen en la ventana de opciones (Menú) que ocupa el lado izquierdo de la misma. Cuando se requiere de información adicional esta es accesada usando ventanas superpuestas o usando la ventana de mensajes.



F1 - Terminar la ejecución (Terminar).

Con esta función se da por terminada la sesión donde se aplica el Sistema de Medición por Coordenadas. Las características lógicas del equipo que fueron modificadas, como los vectores de interrupción, se reestablecen y se inhibe al sistema de preprocesamiento.

Se requiere que la función sea confirmada para que sea efectivamente llamada, esto se realiza leyendo la confirmación desde la ventana de mensajes. La opción por omisión confirma el fin de la sesión.

F2 - Inicializar Sistema (Reset):

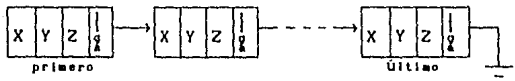
Esta función se encarga de reinstalar los vectores de interrupción y sus rutinas de atención, programar adecuadamente a los distintos dispositivos, inicializar los registros de *hardware* y *software* y redibujar la pantalla. Esta función es llamada siempre que el programa principal del sistema comienza a ejecutarse y realiza la secuencia siguente:

- Inhibe interrupciones mediante el bit IF del registro de estado del procesador.
- Programa el controlador de interrupciones.
- Programa los puertos paralelos.
- Respalda los vectores de interrupción utilizados.
- Establece los vectores de las nuevas rutinas de interrupción.
- Inicializa los registros de *software* y *hardware*.
- Habilita las interrupciones.

F3 - Memorizar:

Una de las principales aplicaciones de estos sistemas, la digitalización de contornos y envolventes tridimensionales, se realiza memorizando las coordenadas de un conjunto de puntos.

El número máximo de puntos que pueden memorizarse es determinado automáticamente al iniciarse la operación y depende de la cantidad de memoria RAM disponible en la computadora. Para memorizar los puntos digitalizados se utiliza una estructura de datos conocida como "lista ligada" en la que las coordenadas de cada punto (X, Y, Z) son almacenadas junto con la dirección de la siguiente componente (liga). Se trata de una estructura de datos dinámica del tipo FIFO (*first input - first output*).



LISTA LIGADA USADA PARA MEMORIZAR PUNTOS.

F4 - Salvar:

Esta función se complementa con la anterior, ya que una vez que los puntos han sido digitalizados, deben ser almacenados para un procesamiento y análisis posterior. La función se activa cuando es llamada, o bien cuando no es posible digitalizar más puntos por las limitaciones de la memoria disponible. Para salvar en disco la información de la digitalización se utilizan registros de cuatro campos, tres de ellos corresponden a las coordenadas X, Y, Z, y otro es usado para indexar el campo. La lista de puntos es destruida cuando éstos se salvan en disco, liberando la memoria para digitalizar más puntos.

INDICE	X	Y	Z
1	X_1	Y_1	Z_1
2	X_2	Y_2	Z_2
.	.	.	.
.	.	.	.
.	.	.	.
N	X_N	Y_N	Z_N

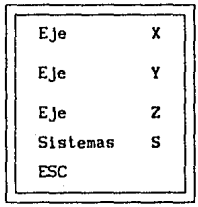
Para almacenar el archivo puede indicarse cualquier lugar en la estructura de directorios del disco mediante una trayectoria (*path*). El nombre del archivo es leído desde la ventana de mensajes.

F5 - Habilitar:

En múltiples ocasiones es necesario inhibir el paso de las señales provenientes de las reglas ópticas hacia el sistema procesador de movimiento. Con esta función puede realizarse la digitalización de contornos tridimensionales y todas las demás tareas en condiciones estables, de tal forma que cualquier acción tomada que se refleje en las lecturas de los desplazamientos pueda visualizarse y aceptarse antes de que los valores leídos cambien.

Esta función está diseñada para operar en sistemas mecánicos (mesas de medición por coordenadas) que tengan la capacidad de inmovilizar sus articulaciones traslacionales y fijarlos en una posición determinada con un margen de libertad menor a la resolución de las reglas ópticas utilizadas, ya que al inhibir la medición con esta función, se corre el riesgo de no detectar las traslaciones ocurridas mientras no se habilite el sistema para medir.

Las funciones siguientes requieren de un parámetro que les indique si deben de actuar únicamente sobre un eje, sobre todo el sistema de referencia o abortar la operación. A cada una de las opciones se le asigna un comando X, Y y Z para cada eje, S para todo el sistema, y ESC para abortar. Para seleccionar el comando se utiliza una ventana, la apariencia de la pantalla con esta ventana es la siguiente:



VENTANA DE SELECCION DE EJE(s)

F6 - Inicialización de ejes:

La secuencia de selección del origen para el sistema de referencia se realiza con esta función. Su objetivo es colocar en los registros de *hardware* y *software* un valor de cero para indicar que en este punto se encuentra el origen. Debido a que se utilizan codificadores ópticos incrementales, puede seleccionarse cualquier punto en un eje o en todo el sistema para utilizarse como origen o referencia.

F7 - Precarga:

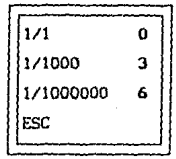
Al contrario de lo que ocurre con la función de inicialización de ejes, en este caso se le asigna un valor inicial distinto de cero al sistema de referencia, se trata de valores de *offset* que son sumados a los desplazamientos sobre el eje correspondiente para modificar el valor desplegado en pantalla. Se debe proporcionar el valor de *offset* en millonésimas de la unidad de medición utilizada (metros, pulgadas), y puede tener 9 dígitos como máximo.

F8 - Resolución:

Con este comando se le indica al sistema cual es la resolución del codificador óptico utilizado. Debido a las características de los codificadores comerciales, se requiere que la resolución esté dada en millonésimas de la unidad de medición, y que no tenga más de cuatro dígitos de longitud, es decir que no sea inferior a una centésima (1/100) de la unidad de medición.

F9 - Escala:

La selección de la escala de medición se realiza con este comando. Los resultados se despliegan dividiendo la unidad de medición (metros o pulgadas) por un factor de 10^0 (1/1), 10^3 (1/1000), o 10^6 (1/1000000). Los factores de escalamiento de las unidades son actualizados en pantalla cada vez que son modificados. Mediante una ventana especial se selecciona el factor de escalamiento.



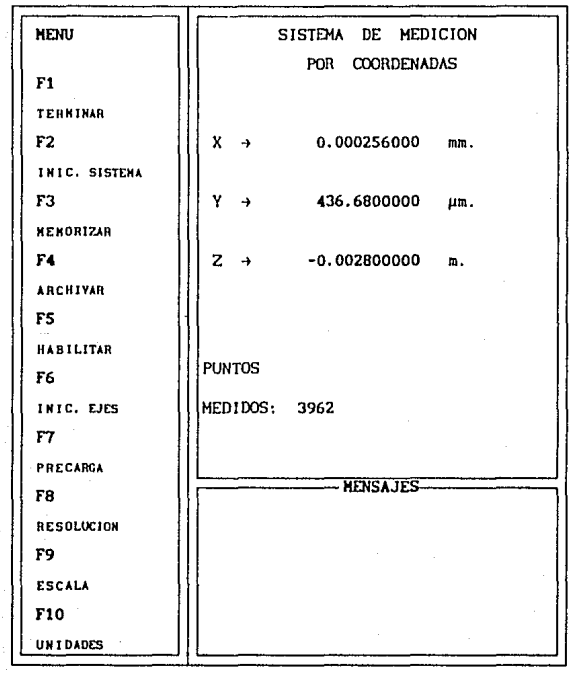
VENTANA DE SELECCION DE ESCALA

F10 - Unidades:

Se puede indicar cual es el sistema de unidades utilizado en cada uno de los ejes. En el Sistema Internacional las lecturas se despliegan en metros, milimetros y micras; en el Sistema Inglés se despliegan en pulgadas, milésimas de pulgada y millonésimas de pulgada. Para seleccionar las unidades se utiliza una ventana cuya apariencia es la que se muestra a continuación:

VENTANA DE SELECCION DE UNIDADES

La pantalla de trabajo del sistema esta compuesta por tres ventanas: a) la ventana de selección de funciones mostrada anteriormente, b) la ventana de despliegue de resultados o de medición y c) la ventana de mensajes. Estas ventanas son visualizadas siempre, mientras que las ventanas de selección de eje(s), escalas y unidades se visualizan únicamente cuando se les requiere.



VISTA GENERAL DE LA PANTALLA DEL SISTEMA

4.2 TECNICAS DE PRESENTACION Y MANIPULACION DE VIDEO.

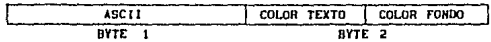
Las imágenes que son desplegadas en una pantalla pueden ser clasificadas en Textos y Gráficas, dependiendo del tipo de tarjeta adaptadora de video que se utilice. Cuando se define el modo de trabajo de la pantalla (texto o gráfico), el adaptador se encarga de accesar la pantalla en la forma adecuada. El elemento básico de un texto es un caracter o *"texel"* (text-cell), el cual está formado por una matriz de puntos o *"pixels"* (picture cell). En el modo gráfico se puede direccionar individualmente cada uno de los *pixels* que forman la pantalla.

Un texto está compuesto por simbolos alfanuméricos y algunos simbolos gráficos predefinidos en el código ASCII, tales como), ô, Ç, ∫, ⊥. El adaptador de video monocromático utilizado en algunos equipos tiene una resolución de 80 por 25 *texels* y puede generar textos en un solo color, mientras que el adaptador de gráficas y color (CGA) tiene dos resoluciones en modo texto, una de 80 por 25 y otra de 40 por 25 *texels*, ambas usan los colores definidos en una paleta de 16 colores para el texto y 8 para el fondo.

4.2A ACCESO A LA MEMORIA DE VIDEO.

En cada adaptador de video existe una cantidad de memoria conocida como memoria de video. Esta memoria es usada para almacenar el texto que es visualizado en la pantalla y tiene las mismas caracteristicas que la memoria RAM usada para contener al Sistema Operativo, los programas de aplicación, etc. El acceso a la memoria de video se realiza de la misma forma que se hace con cualquier localidad en el mapa de memoria de la computadora, sin embargo cualquier modificación en esta memoria se refleja como un cambio en la pantalla. La dirección de inicio de la memoria de video depende del adaptador de video utilizado, para la tarjeta CGA la dirección es $B800_{16}$.

En el modo de texto cada *texel* esta formado por dos bytes, el primero de ellos es el texto alfanumérico y/o gráfico, el segundo es el atributo, este último indica la apariencia del texto, ya que es usado para definir los colores del texel, tal como se indica en la siguiente gráfica:



ORGANIZACION DE UN TEXEL

Para modificar el contenido de un un byte de texto en la memoria de video, cuya posición en la pantalla está dada por el renglón 'r' y la columna 'c', se utiliza la expresión que determina su dirección:

$$[\ B8000_{16} + (r-1)x80 + (c-1)x2] = TEXTO \qquad (9)$$

donde $B8000_{16}$ es la dirección de inicio de la memoria de video en el adaptadoe CGA, $(r-1)x80$ es el número de bytes correspondiente a los $(r-1)$ *texels* de los renglones anteriores y $(c-1)x2$ es el desplazamiento sobre el renglón deseado, es decir el número de *texels* anteriores sobre el mismo renglón. El byte de atributo puede modificarse usando una expresión similar, únicamente se necesita cambiar la dirección base usada.

$$[\ B8000_{16} + (r-1)x80 + (c-1)x2] = ATRIBUTO \qquad (10)$$

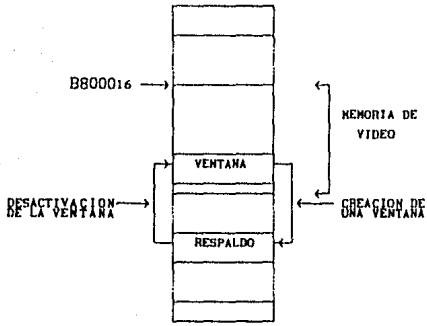
Por medio de los accesos directos a la memoria de video, se realizan escrituras instantáneas en la pantalla, a diferencia de los accesos mediante las llamadas al *BIOS*. Con el acceso directo a la memoria se introducen retrasos mínimos en el procesamiento, haciendo posible que el despliegue se realice en tiempo real, ya que los retrasos dependen únicamente de los tiempos de captura y cálculo de resultados. Además se evitan las interrupciones de *software* necesarias para hacer las llamadas a la sección del *BIOS* que realiza los accesos a la pantalla.

El sistema de medición de coordenadas aprovecha las características de un adaptador CGA o compatibles (EGA, VGA, HERCULES, etc). Se utiliza un modo de resolución de 40 x 25 *texeles* y una pantalla de 16 colores. En el apéndice B se incluyen todas las rutinas usadas para accesar la memoria de video, y para generar los marcos y/o ventanas necesarias.

4.2B VENTANAS

Una ventana de video es una sección de la pantalla que se puede accesar en forma independiente. Al definir una ventana deben especificarse los límites de sus bordes, generalmente se hace definiendo las coordenadas de las esquinas superior izquierda e inferior derecha, así como la posición en la que se desplegará, esto se hace definiendo las coordenadas de algunos de los *texeles* de la ventana en la pantalla, por ejemplo, las coordenadas en la pantalla de la esquina superior izquierda de la ventana.

El contenido de cada una de las ventanas se almacena en una sección de memoria independiente de la memoria primaria de video, que puede ser la memoria del segmento de datos o la memoria para páginas adicionales del adaptador de video. Cuando una ventana se activa, su contenido se copia en la memoria primaria de video, después de haber salvado el contenido de la región traslapada. Al desactivarse la ventana se restablece el contenido original de la región de video traslapada.



MANIPULACION DE MEMORIA DURANTE LAS OPERACIONES CON VENTANAS

En el apéndice B se encuentran las rutinas necesarias para trabajar con "ventanas virtuales", estas ventanas se crean en su totalidad en la memoria de video cada vez que se activan para ser realizadas en la pantalla. Las "ventanas virtuales" son marcos de trabajo; ya que no son respaldadas cuando se desactivan, su contenido se pierde cada vez que son traslapadas o desactivadas, además no establecen límites estrictos de trabajo que impidan accesar regiones de la pantalla que estén fuera de la ventana; esta última característica se debe al uso de accesos directos a la memoria de video, ya que no es necesario usar los parámetros definidos por el *BIOS*.

Las operaciones básicas que se realizan con las ventanas son:

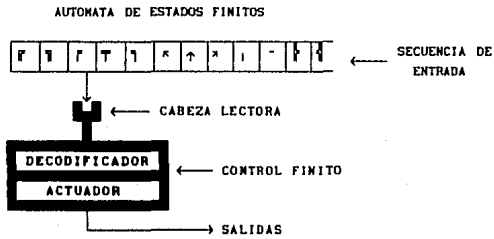
- Determinar color de fondo.
- Determinar color de texto.
- Borrar el contenido.
- Asignar el contenido a otra área de memoria (Salvar).
- Copiar el contenido de otra área de memoria (Restaurar).
- Dibujar un margen exterior.
- Crear una ventana.

4.3 MAQUINA DE ESTADOS DEL SISTEMA.

La estructura del control de estados tiene como modelo de referencia la organización funcional de un autómata o máquina de estados finitos. Un autómata es un modelo matemático de un sistema con las siguientes características:

- Entradas y salidas discretas.
- Un número finito de estados o representaciones internas.
- Un estado inicial.
- Al menos un estado final.

El estado en el que se encuentra el sistema sintetiza la información de las entradas y estados anteriores y se utiliza para calcular la respuesta a las siguientes entradas. Si describimos un autómata de estados finitos como una máquina, podemos decir que está compuesta por una cabeza lectora y un control finito para los estados. El autómata lee una secuencia de símbolos o cinta de entrada, accesando uno a la vez. El control finito especifica el número (finito) y las relaciones o transiciones entre los estados en los cuales puede encontrarse el autómata, formando un conjunto finito. Cada vez que la cabeza se mueve hacia un nuevo caracter de la secuencia de entrada para leerlo, ocurre una transición hacia un nuevo estado (posiblemente el mismo).



El código del programa principal del Sistema de Medición por Coordenadas está compuesta por las mismas partes descritas en los párrafos anteriores. El teclado es examinado para verificar si existe un caracter de entrada (comando), si existe es leido, después se determina el nuevo estado del sistema en función del estado y el caracter actuales, finalmente se usa el nuevo estado para determinar cual es la acción que debe realizarse.

```
if ( bioskey(0) ) ──────┐
   {   c = bioskey(1) ; ┘──────── CABEZA LECTORA
       state = state_decoder( c ) ; ┐
       state_actuator( state ) ;  ──┘── CONTROL FINITO
   }
```

4.3A CONTROL FINITO.

Una expresión para describir un autómata es mediante la representación matricial de una transformación, de tal forma que:

AUTOMATA ⟶ CONTROL FINITO ⟶ (K, V_T, M, Q_0, F)

donde K es un conjunto finito no vacio de elementos llamados estados, V_T es un alfabeto finito de entrada, M es una función lineal de transición $K \times V_T \rightarrow K$, Q_0 es el estado inicial de la representación ($Q_0 \subseteq K$), y F es un conjunto finito no vacio de estados finales ($F \subseteq K$).

Un autómata puede explorar una secuencia de entrada y, mediante las transiciones entre los estados del control finito, determinar si la secuencia lo conduce a un estado final. Si después de una serie finita de transiciones el autómata se encuentra en un estado final, se dice que la secuencia de entrada que origina la ocurrencia de transiciones desde un estado inicial, hasta un estado final, es aceptada por el autómata. El conjunto de secuencias que son aceptadas es conocido como el Lenguaje que describe al autómata.

En el nivel de control de Estados Lógicos del programa se realizan las operaciones necesarias para supervisar el secuenciamiento de los comandos del sistema. En la sección 4.1 se detallaron cada una de las funciones y se mencionaron los comandos asociados a ellas. Existen condiciones que son incorrectas dentro de la secuencia de comandos del programa. Así como es necesario seguir una secuencia predefinida para especificar la trayectoria y el nombre de acceso a un archivo, existe una secuencia predefinida que es reconocida por el autómata para indicar cual es la función especifica que se desea realizar.

De acuerdo con la notación formal introducida, el nivel de Control de Estado Lógico puede ser expresado de la siguiente forma:

$$(K, V_T, M, Q_0, F)$$

donde $K = \{ q_0..q_{10}, q_{12}..q_{16}, q_{18}..q_{22}, q_{24}..q_{28}, q_{30}..q_{51}, q_{54}..q_{56} \}$

$V_T = \{ F_1..F_{10}$, RETURN, ESC, $\uparrow$, $\downarrow$, BACK_SPACE, +, -, 0..9, a..z, A..Z, \, : $\}$

$Q_0 = \{ q_0 \}$

$F = \{ q_1..q_5, \quad q_{12}..q_{14}, \quad q_{18}..q_{20}, \quad q_{24}..q_{26}, q_{30}..q_{32}, \quad q_{35}..q_{37}, \quad q_{46}..q_{51} \}$

El conjunto K contiene 51 estados no consecutivos, numerados del 0 al 56. El alfabeto V_T está formado por las teclas de función, los símbolos necesarios para formar un número, un nombre de archivo, y para seleccionar una opción dentro de los menús, tal como se muestran en la sección (4.2). El estado desde el cual se inicia el reconocimiento de una secuencia es q_0, este estado puede ser alcanzado después de analizar una secuencia, y determinar si pertenece o no a la serie de comandos o tareas que pueden ser llevadas a cabo. El conjunto de estados finales F está formado por 26 estados, algunos de los cuales llaman a otro autómata para leer un número entero signado o un nombre de archivo.

La función $M = K \times V_T \longrightarrow K$, especifica las transiciones, inducidas por un elemento de V_T, que deben realizarse desde el estado en que se encuentra el autómata hacia un nuevo estado, ambos pertenecientes a K. Las tablas de la matriz M que se incluyen en esta sección, expresan brevemente y por comprensión las transiciones utilizadas para analizar las secuencias de comandos. Esta representación es necesaria debido a que la representación matricial de M tiene 51 renglones que corresponden a los estados de K, y 81 columnas de elementos de entrada comprendidos en V_T. Las transiciones mostradas corresponden a aquellas que conducen a secuencias correctas de comandos.

TABLA 4.1

FUNCION	F1	F2	F3	F4	F5	F6	F7	F8	F9	F10
ESTADO	1	2	3	4	5	6	7	8	9	10

Las teclas de funciones pueden ser utilizadas en cualquier parte de la estructura del autómata y siempre lo llevarán al estado asociado en la forma en que se indica en la tabla (4.1).

TABLA 4.2

		EJES (a)				
		X	Y	Z	SISTEMA	ESC
ESTADO	6 →	12	18	24	30	35
	7 →	13	19	25	31	35
	8 →	14	20	26	32	35
	9 →	15	21	27	33	35
	10 →	16	22	28	34	35
VALOR DEL EJE	→	+6	+12	+18	+24	

Cuando se selecciona una función que nos conduzca a los estados indicados en el lado izquierdo de la tabla (4.2), es necesario indicar si la acción será realizada sobre uno solo de los ejes (x, y, z), o sobre todo el sistema (s). Para determinar el estado siguiente en un caso como éste, lo que se hace es sumar al estado presente un valor constante asignado cada eje, en la forma en que se indica en la tabla (4.2).

TABLA 4.3

ESTADO	0	3	6	M,m	P,p	C,c	ESC
F9 + EJES(s)	46	47	48	56	56	56	36
F10 + EJES(s)	56	56	56	49	50	51	37

Las tablas anteriores son un ejemplo del tipo de secuencias que pueden ser generadas y aceptadas con el Control de Estado Lógico del sistema.

4.3B GRAMÁTICA.

Una forma alternativa para expresar por comprensión el lenguaje generado por el control finito, consiste en el uso de un conjunto de expresiones sintácticas relacionadas que generen el mismo lenguaje, este conjunto recibe el nombre de **Gramática.**

Una definición usada para representar una gramática es la siguiente:

$$G \longrightarrow (V_T, V_N, S, \Phi)$$

donde V_T es es un conjunto de terminales, es decir, simbolos no derivables; V_N es un conjunto de simbolos NO terminales derivables; S es un elemento de V_N, y por lo tanto del vocabulario, es conocido como el simbolo de inicio. Φ es un conjunto finito no vacio de relaciones o reglas de producción entre los elementos de V_T y V_N.

De acuerdo con esta última definición, el control finito introducido anteriormente puede ser expresado como $\S \longrightarrow (V_T, V_N, S, \Phi)$, donde:

V_T = { $F_1..F_{10}$, RETURN, ESC, $\uparrow$, $\downarrow$, BACK_SPACE,
+, -, 0..9, a..z, A..Z, \, : }

V_N = { α, δ, σ, γ, λ, ω, ρ, μ, τ, S, f_n, S_L,
f_a, f_b, T_0, T_1, T_2, d_1, d_2, NUM, NOM }

Φ =

1) $\S \longrightarrow f_N$
2) $\longrightarrow F_{10}\omega$
3) $\longrightarrow F_4$NOM
4) $\longrightarrow f_bT_1$
5) $\longrightarrow f_aT_1$NUM
6) $\longrightarrow F_9T_2\rho$
7) $\longrightarrow F_{10}T_2\rho$
8) $\longrightarrow \gamma$
9) $f_n \longrightarrow [F_1..F_{10}]$
10) $f_a \longrightarrow [F_7, F_8]$
11) $f_b \longrightarrow [F_6, f_a, F_9, F_{10}]$
12) $T_0 \longrightarrow [\gamma, \lambda, f_n]$
13) $T_1 \longrightarrow [\tau, \lambda\tau]$
14) $T_2 \longrightarrow [T_1, T_1\lambda]$
15) $d_1 \longrightarrow [\alpha, \alpha d_1, \delta, \delta d_1]$
16) $d_2 \longrightarrow [\backslash d_1, \backslash d_1 d_2]$
17) NOM $\longrightarrow$ [T_0, d_2T_0, d_1T_0, α:d_2T_0, α:d_1T_0]
18) NUM $\longrightarrow$ [T_0, σT_0, δT_0, $\sigma\delta T_0$]

19) $S_L \longrightarrow \lambda$RETURN

20) $\alpha \longrightarrow$ [a..z, A..Z]

21) $\delta \longrightarrow$ [0..9, 0δ..9δ]

22) $\sigma \longrightarrow$ [+, -]

23) $\gamma \longrightarrow$ [RETURN, ESC]

24) $\lambda \longrightarrow$ [$\uparrow$, $\downarrow$, $\uparrow\lambda$, $\downarrow\lambda$]

25) $\omega \longrightarrow$ [s, n, S, N, γ, f_n, λ]

26) $\rho \longrightarrow$ [0, 3, 6, γ, f_n, S_L]

27) $\mu \longrightarrow$ [m, p, c, M, P, C, γ, f_n, S_L]

28) $\tau \longrightarrow$ [x, y, z, s, X, Y, Z, S, γ, f_n, S_L]

Las secuencias de símbolos que se generan con esta gramática corresponden a las series de comandos quo se le aplican al sistema de Medición por Coordenadas para que realice alguna función o tarea. Una secuencia típica tendría la forma siguiente:

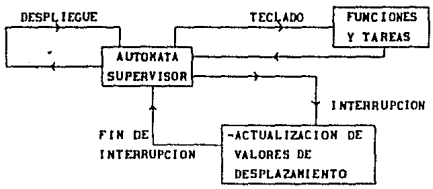
F_7	Y	1250	RETURN

La secuencia anterior puede formarse usando las producciones 5, 10, 13, 28, 18, 21, 12 y 23 en ese orden.

El estrato de Control de Estado Lógico está formado por las rutinas que implementan la gramática definida en los párrafos anteriores. Cualquier secuencia de símbolos que pueda ser derivada usando las producciones de Φ, es una secuencia de comandos que puede ser procesada por el sistema.

4.4 PROCESAMIENTO DE LAS INTERRUPCIONES.

En el estrato de Enlace y Control de la arquitectura se encuentran las rutinas que se encargan de atender a las interrupciones generadas en el estrato Físico. Estas rutinas se encargan de manipular y transportar la información de los eventos ocurridos en cada uno de los ejes, hacia los registros lógicos (variables) controlados por el autómata supervisor del estrato de Control de Estados Lógicos, es decir, su función es enlazar la etapa física con la lógica.



FLUJO DE SEÑALES DURANTE LA OPERACION NORMAL

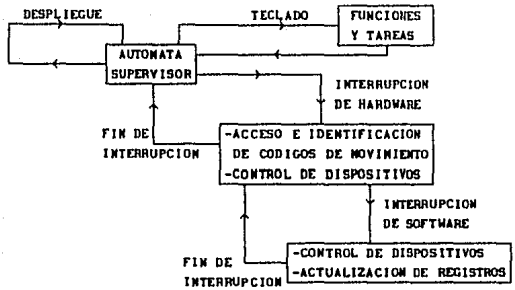
El preprocesador genera dos interrupciones, una para las señales de *Carry* y otra para los *Borrows*, éstas son las únicas interrupciones de *hardware* que se utilizan en el sistema y son generadas por el preprocesador. Para cada una de estas interrupciones se escribe un procedimiento que es capaz de atenderla.

Cuando se inicia la ejecución de una interrupción de *hardware*, automáticamente se copia en el *stack* (la "pila" del programa) el valor de los registros CS e IP para tener una dirección de retorno, en las interrupciones de *software* además de los registros anteriores, se copian también los registros SI y DI.

En el lenguaje "C" existe el calificador **INTERRUPT**, que se aplica a una función que está diseñada para atender interrupciones. Cuando el compilador del lenguaje encuentra una función de tipo **INTERRUPT**, automáticamente genera el código necesario para copiar en el *stack* los otros registros del procesador (desde AX hasta DX, ES y DS). Una vez que ha terminado la interrupción, los registros son reestablecidos con sus valores originales, obteniéndolos del *stack*. Las rutinas de atención a las interrupciones, al igual que todo el sistema, están escritas usando el lenguaje "C". Las operaciones de identificación y actualización de registros del estrato de Enlace y Control de la Arquitectura son transparentes para el usuario, debido a que al iniciarse la interrupción se salva en el *stack* el estado absoluto del procesador, de tal forma que al regresar al programa supervisor, lo hace en las condiciones exactas en las que se encontraba.

El proceso de actualización de registros, tanto lógicos como físicos, se inicia con una interrupción de *hardware*, cada una de ellas genera a su vez una interrupción de *software* con el fin de identificar el código de movimiento. Cada código es en realidad un vector de interrupción que corresponde a una rutina que actualiza los registros de acuerdo a las tablas (1) y (2) del capítulo III. Es esta última interrupción la que realmente modifica los registros. Este proceso es importante porque, de acuerdo a la ecuación (5) del capítulo III, el valor del desplazamiento sobre un eje se obtiene a partir de estos registros.

Dentro de la tabla de vectores de interrupción existente en la computadora, hay un cierto número de vectores que están reservados para utilizarse en aplicaciones desarrolladas por los usuarios. Los vectores comprendidos en los intervalos 96-102 y 241-247, están disponibles para instalar los manipuladores de las interrupciones de *software* necesarias, que son 14 en total.



FLUJO DE SEÑALES DURANTE UNA INTERRUPCION

Tomando en cuenta que cada interrupción de hardware corresponde a un sentido de desplazamiento, el procedimiento que se sigue para actualizar los registros adecuados es el siguiente:

a) Se accesa y corrige el código de movimiento.
b) Se determina el eje que originó la interrupción.
c) El registro contador de señales, el contador de *carrys* o *borrows* correspondiente, es borrado.
d) Se incrementa o decrementa el contador de interrupciones, según el caso.
e) El controlador de interrupciones es informado del fin de la interrupción.

En el caso de interrupciones por desplazamientos positivos, el acceso al código de movimiento se realiza con una lectura al puerto de control C en la dirección 070216; para desplazamientos negativos la lectura se realiza desde el puerto de control B en la dirección 070116.

Antes de utilizar el código es necesario ajustarlo para que corresponda a los vectores disponibles, ya que por las restricciones impuestas por la electrónica y la organización de la tabla de vectores, el valor leído no corresponde exactamente al vector necesario.

En los desplazamientos positivos la corrección se realiza porque el puerto de control C (070216) está dividido en dos secciones, una de entrada y otra de salida; por los cuatro bits menos significativos de la sección de entrada (C_0 hasta C_3) se lee el código de movimiento. Los vectores asignados a este código van de la 241 a la 247, por lo tanto la corrección se hace realizando una suma lógica (OR) entre el código leído y el dato $F8_{16}$.

CODIGO LEIDO	CORRECCION	VECTOR DE INTERRUPCION
8 + xyz	(8 + xyz) v $F8_{16}$	$F8_{16}$ + xyz

CORRECION DEL VECTOR PARA CARRYS

A los desplazamientos negativos se les asignan los vectores comprendidos desde el 96 hasta el 102, mientras que el código leído se encuentra en el intervalo 153 a 159. Para corregir el código es necesario obtener el complemento lógico de cada uno de los bits.

CODIGO LEIDO	CORRECCION	VECTOR DE INTERRUPCION
98_{16} + xyz	$\overline{98_{16}}$ + $\overline{xyz}$	60_{16} + $\overline{xyz}$

CORRECCION DEL VECTOR PARA BORROWS

Al ejecutarse la interrupción de *software* indicada por el código corregido, quedan identificados los ejes que interrumpen. Las rutinas de atención generan un pulso para borrar el contador de *carrys* o *borrows*, según el caso, del eje que haya generado la interrupción de hardware, para evitar que al regresar el control al programa se realicen llamadas redundantes debidas a una misma interrupción. El tiempo que transcurre desde el momento en que se inicia el procedimiento de interrupción de hardware, hasta que se borran los contadores de 4 bits (*carrys y/o borrows*) es crítico, ya que hay un desplazamiento de tan sólo 512 micras (256 pulsos) antes de que éstos contadores reciban un nuevo flanco en su terminal de reloj.

A continuación se incrementa o decrementa el contador de interrupciones de cada uno de los ejes que interrumpieron. En la ecuación (5) esta operación está especificada como la resta de las interrupciones debidas a *carrys* menos las debidas a *borrows*, (ND-NI), sin embargo no es necesario realizar la resta aritmética de estos valores, ya que se utiliza un solo registro lógico de 16 bits cuyo contenido está definido como un valor entero con signo (**signed int**). Mediante operaciones simples de incremento y decremento los dos tipos posibles de interrupciones de *hardware*, manipulan el contenido de este último registro, resultando en un algoritmo funcionalmente idéntico al de la resta aritmética.

Por último se regresa el control a las rutinas de atención de interrupciones de *hardware*, a su vez, esta rutina habilita a los contadores de 4 bits finalizando el pulso de borrado, e informa al controlador de interrupciones que la interrupción ha terminado y regresa el control al autómata administrador.

En el apéndice 3 se incluyen las rutinas de atención a interrupciones, así como los procedimientos necesarios para instalarlas en los vectores adecuados, se incluyen también las rutinas usadas para respaldar los vectores, programar el controlador de interrupciones y controlar el preprocesador.

V CONCLUSIONES.

El sistema desarrollado en el presente trabajo, está basado en un algoritmo que se puede denominar "incremental" o de "conteo" dado su principio de operación, como se explicó en las secciones (1.3) y (3.1). Se trata de un método de cuantificación de propósito general que hace uso de elementos de conteo elementales, distribuidos en diversos niveles de *hardware* y *software*.

El sistema de medición por coordenadas, tal como fué diseñado en este trabajo, ha sido construido y evaluado en condiciones reales de operación. Los resultados obtenidos son satisfactorios y pueden ser comparados favorablemente con los parámetros de rendimiento de otros equipos.

Algunos de los parámetros más significativos son los siguientes:

- En traslaciones simultáneas en los tres ejes, puede procesar desplazamientos efectuados con una velocidad de 40 cm/seg mínimo.

- Con traslaciones sobre un solo eje puede procesar desplazamientos de 60 cm/seg.

- Permite digitalizar un mínimo de 2500 puntos en el espacio de trabajo, antes de tener que respaldarlos en medios magnéticos.

Una de las principales diferencias entre el sistema desarrollado en este trabajo y los sistemas comerciales actuales, consiste en la utilización de tan sólo un microprocesador central; la función de este procesador, cuando está ejecutando el programa principal del Sistema de Medición, es atender los eventos ocurridos en cada uno de los ejes y actualizar el despliegue de las mediciones.

La evaluación del sistema fue realizada utilizando una computadora personal de tipo XT, con 640 KBytes de memoria y 4.7 MHz. como base de tiempo, además de codificadores ópticos de dos micras de resolución. Es posible utilizar codificadores ópticos más precisos, sin embargo la velocidad máxima a la que se puede trasladar una articulación disminuye proporcionalmente. Por otro lado, al utilizar equipos con velocidades de procesador de 8 a 12 MHz, se incrementa la velocidad máxima a la que se puede trasladar una articulación.

Por último, incluyendo el valor de la computadora requerida para instalar el sistema, el costo global resulta ser muy inferior al de los sistemas existentes en el mercado.

APENDICE A

```
            /*          Definición de constantes          */

#define  TRUE        0xFF
#define  FALSE       0x00
#define  BEEP        0x07

    /* Símbolos usados para las transiciones entre estados */

#define  ESC         0x011B
#define  RETURN      0x1C0D
#define  BACK_SPACE  0x0E08
#define  F1          0x3B00
#define  F2          0x3C00
#define  F3          0x3D00
#define  F4          0x3E00
#define  F5          0x3F00
#define  F6          0x4000
#define  F7          0x4100
#define  F8          0x4200
#define  F9          0x4300
#define  F10         0x4400
#define  key_0       0x0B30
#define  key_3       0x0433
#define  key_6       0x0736
```

```
#define  C_key       0x2E43
#define  c_key       0x2E63
#define  M_key       0x324D
#define  m_key       0x326D
#define  N_key       0x314E
#define  n_key       0x316E
#define  P_key       0x1950
#define  p_key       0x1970
#define  S_key       0x1F53
#define  s_key       0x1F73
#define  X_key       0x2D58
#define  x_key       0x2D78
#define  Y_key       0x1559
#define  y_key       0x1579
#define  Z_key       0x2C5A
#define  z_key       0x2C7A
#define  UP_ARROW    0x4800
#define  DW_ARROW    0x5000

/* Símbolos aritméticos */

#define  PLUS1       0x0D2B
#define  PLUS2       0x4E2B
#define  MINUS1      0x0C2D
#define  MINUS2      0x4A2D

/* Símbolos para la trayectoria de los archivos */

#define  UNDERSCORE  0x0C5F
#define  ROOT_MARK   0x2B5C
#define  TWO_POINTS  0x273A

/* Símbolo asignado a cada eje */

#define  X_axis      0x00
#define  Y_axis      0x01
```

```
#define  Z_axis      0x02

/*   Puertos del 8255 para los contadores Up-Down   */

#define  PA_PLS      0x780  /*       contador eje X   */
#define  PB_PLS      0x781  /*       contador eje Y   */
#define  PC_PLS      0x782  /*       contador eje Z   */
#define  PCTRL_PLS   0x783  /*        control         */

/*   Puertos del 8255 para los contadores de interrupciones   */
   Puerto Bajo     borrows (4)
   Puerto Alto     carry s (4)
#define  PA_TCud     0x680  /*       eje X        */
#define  PB_TCud     0x681  /*       eje Y        */
#define  PC_TCud     0x682  /*       eje Z        */
#define  PCTRL_TCud  0x683  /*      control       */


/*   Puertos del 8255 para la lógica de control   */

#define  PA_LGC  0x700  /*  A0  borrar carry  x
       A1  borrar borrow x
       A2  borrar carry  y
       A3  borrar borrow y
       A4  borrar carry  z
       A5  borrar borrow z */

#define  PB_LGC  0x701  /*  B0 interrupción por borrow en Z
                           B1 interrupción por borrow en Y
                           B2 interrupción por borrow en X
       B3-B7  código de interrupción alambrado */

#define  PC_LGC  0x702  /*  C0 interrupción por carry en Z
                           C1 interrupción por carry en Y
                           C2 interrupción por carry en X
       C3   código de interrupción alambrado
```

```
        C4   borrar interfaz en X
        C5   borrar interfaz en Y
        C6   borrar interfaz en Z
        C7   habilitación de los codificadores   */

#define  PCTRL_LGC  0x703  /*        control       */

/*  modos de operación de los puertos  */

#define  DATA_MODE  0x83   /*
         PA ( entrada )   contadores X
         PB ( entrada )   contadores Y
         PC ( entrada )   contadores Z */

#define  CTRL_MODE  0x9B   /* palabra de control para la lógica de
         interrupciones
         PA  (salida) modificar registro de código de movimiento
         PB  (entrada) código de interrupción por borrow
         PCl (entrada) código de interrupción por carry
         PCh (salida) borrar contadores Up-Down */
```

APENDICE B

```
/*---    Funciones de acceso directo a memoria de video    ---*/

void settexel_color( char x, char y, char atrib ) ;
void settexel_char ( char x, char y, char text  ) ;
void settexel( char x, char y, unsigned int texel ) ;
int  gettexel( char x, char y ) ;


void set_window    ( char left, char up, char right, char down,
                     char background, char foreground ) ;
void window_clear  ( char left, char up, char right, char down ) ;
void window_save   ( char left, char up, char right, char down,
                     unsigned int  *(*ptr) ) ;
void window_restore( char left, char up, char right, char down,
                     unsigned int *(*ptr) ) ;
void box( char left, char up, char right, char down ) ;
void make_window   ( char left, char up, char right, char down,
                     char background, char foreground ) ;
void xycputs ( char x, char y, char *string ) ;


/*  Fija atributos de color en la posicion x, y  */
void settexel_color( char x, char y, char atrib )
  {
  x -= 1 ;
  x <<= 1 ;
  *( (char far *) 0xB8000001 + ( y-1 )*80 + x ) = atrib ;
  }
```

```
/*   Escribe un caracter en la posicion x, y; usa atributos
existentes  */
void settexel_char( char x, char y, char text )
  {
  x -= 1 ;
  x <<= 1 ;
  *( (char far *) 0xB8000000 + ( y-1 )*80 + x ) = text ;
  }


/*  Obtiene un caracter y sus atributos en la posicion x, y  */
int gettexel( char x, char y )
 {
 x -= 1 ;
 x <<= 1 ;
 return *( (int far *) ( 0xB8000000+( y-1 )*80 + x ) ) ;
 }


/*  Escribe un caracter y sus atributos en la posicion x, y  */
void settexel( char x, char y, unsigned int texel )
 {
 x -= 1 ;
 x <<= 1 ;
    *( (int far *) ( 0xB8000000+( y-1 )*80 + x ) ) = texel ;
 }


/*  Define una ventana; fija colores de fondo y texto  */
void set_window( char left, char up, char right, char down,
    char background, char foreground )
  {
  char x, y, atrib ;
```

```
  background <<= 4 ;
  atrib = ( foreground + background ) & 0x7F ;
  textattr( atrib ) ;
  for ( y = up ; y <= down ; y++ )
   for ( x = left ; x <= right ; x++ )
      settexel_color( x, y, atrib ) ;
  }

/*   Borra el contenido de una ventana, utiliza el atributo
existente */
void window_clear( char left, char up, char right, char down )
  {
  char x, y ;
  for ( y = up ; y <= down ; y++ )
   for ( x = left ; x <= right ; x++ )
      settexel_char( x, y, 0x20 ) ;
  }

/*  Salva el contenido de la ventana especificada  */
void window_save( char left, char up, char right, char down,
      unsigned int  *(*ptr) )
  {
  char             x, y ;
  unsigned int     k    ;

  k = 0 ;

  *ptr = (unsigned int *)
        malloc( (right-left+1)*(down-up+1) << 1 ) ;
  for ( y = up ; y <= down ; y++ )
   for ( x = left ; x <= right ; x++ )
    *( *ptr + k++ ) = gettexel( x, y ) ;
  }
```

```
/*  Restaura el contenido de la ventana especificada  */
void window_restore( char left, char up, char right, char down,
      unsigned int *(*ptr) )
  {
  char   x, y ;
  int       k ;
  k = 0 ;
  for ( y = up ; y <= down ; y++ )
   for ( x = left ; x <= right ; x++ )
    settexel( x, y, *( *ptr + k++ ) ) ;
  free( *ptr ) ;
  *ptr = NULL ;
  }

/*  Dibuja un marco ; utiliza atributo existente  */
void box( char left, char up, char right, char down )
  {
  char x, y ;
  for ( x = left ; x < right ; x++ )
   {
   settexel_char( x, up, 0xC4 ) ;
   settexel_char( x, down, 0xC4 ) ;
   }
  for ( y = up ; y < down ; y++ )
   {
   settexel_char( left, y, 0xB3 ) ;
   settexel_char( right, y, 0xB3 ) ;
   }
  settexel_char( left, up, 0xDA ) ;
  settexel_char( left, down, 0xC0 ) ;
  settexel_char( right, up, 0xBF ) ;
  settexel_char( right, down, 0xD9 ) ;
  }
```

```
/* Crea una ventana */
void make_window( char left, char up, char right, char down,  char
                  background, char foreground )
  {
  set_window( left, up, right, down, background, foreground ) ;
  window_clear( left, up, right, down ) ;
  box( left, up, right, down ) ;
  }

/* Escribe una cadena de caracteres; utiliza atributo existente */
void xycputs( char x, char y, char *string )
  {
  char k ;
  k = x ;
  while ( *string != NULL )
     settexel_char( k++, y, *( string++ ) ) ;
  }
```

APENDICE C

Inclusion de librerias y archivos

```
#include <io.h>
#include <dos.h>
#include <math.h>
#include <conio.h>
#include <stdio.h>
#include <ctype.h>
#include <errno.h>
#include "define.smc"
#include "cga_drv.smc"
```

Declaracion de variables globales

```
unsigned char  state       ,   /* estado del sistema  */
               slct_y      ,   /* indices de ventanas */
               axis_y      ,
               scale_y     ,
               units_y     ,
               X_scale     ,   /* escala de medicion */
               Y_scale     ,
               Z_scale     ,
               X_units     ,     /* unidades de medicion */
               Y_units     ,     /* de la regla optica   */
               Z_units     ,
               X_convt     ,     /* convertir unidades ?  */
               Y_convt     ,
               Z_convt     ,
               set_enable  ,     /* habilitacion del sistema */
               file_ext[27]   , /* nombre del archivo en disco */
               str_digit[10]  , /* string numérico */
               isr_backup[64]  ;/* respaldo de los vectores de
                                   interrupción */
```

```
FILE        *file_int  ; /* representación interna de un
                           archivo                          */

unsigned int   X_res ,     /* resolución de los ejes */
               Y_res ,
               Z_res ,
               index ,     /* número de puntos memorizados */
               maxcoords , /* máximo número de puntos que se
                              pueden memorizar */
               *window1_ptr , /* apuntadores a las ventanas */
               *window2_ptr ,
               *window3_ptr ;

volatile long  int_x , /*  número de interrupciones  */
               int_y ,
               int_z ;

long    axis_valx ,   /*  valor de despliegue del eje   */
        axis_valy ,
        axis_valz ,
        offset_x  ,   /*  offset al sistema de referencia  */
        offset_y  ,
        offset_z  ;

struct list {     /* estructura usada para almacenar puntos  */
            long   x ;
            long   y ;
            long   z ;
            struct list  *link ;
            } ;

struct list  *coords  , /* apuntadores a la lista de puntos
              *downptr ,    memorizados                    */
              *topptr  ;
```

```
void   *heapptr ; /* apuntador al heap */

union REGS regsin  , /* pseudo registros del microprocesador */
          regsout ;
```

DECLARACION DE PROTOTIPOS

Funciones de atencion a interrupciones

```
void interrupt carry_3d ( void ) ;
void interrupt carry_x  ( void ) ;
void interrupt carry_y  ( void ) ;
void interrupt carry_z  ( void ) ;
void interrupt carry_xy ( void ) ;
void interrupt carry_xz ( void ) ;
void interrupt carry_yz ( void ) ;
void interrupt carry_xyz( void ) ;
void interrupt borrow_3d ( void ) ;
void interrupt borrow_x  ( void ) ;
void interrupt borrow_y  ( void ) ;
void interrupt borrow_z  ( void ) ;
void interrupt borrow_xy ( void ) ;
void interrupt borrow_xz ( void ) ;
void interrupt borrow_yz ( void ) ;
void interrupt borrow_xyz( void ) ;
```

Funciones de manejo de pantalla

```
void menu_drv( char *y, char new_y, char x1, char x2, char x1_2,
               char x2_2, char BACK1, char FRONT1, char BACK2,
               char FRONT2, char FOPT );
void windows_eraser( char wds ) ;
void axis_window ( void ) ;
void scale_window( void ) ;
void units_window( void ) ;
```

```
void slct_window ( void ) ;
void meassure_window( void ) ;
void message_window ( void ) ;
void system_screen  ( void ) ;
```

Configuracion del sistema; acceso y control del hardware

```
void vect_backup( unsigned int vector, unsigned char lim,
                  unsigned char *array, unsigned char back_off ) ;
void vect_restore( unsigned int vector, unsigned char lim,unsigned
                   char *array, unsigned char back_off ) ;
void enable_signal( char ok) ;
void enable_display( char ok) ;
void clear_axis( char axis, char mask1, char mask2, long *offset,
                 volatile long *int_axis, long *axis_val ) ;
void clear_coords( void ) ;
long get_axis_count( long resolution, volatile long int_axis,
                     int TCud_PORT, int PULSE_PORT  ) ;
void init_system ( void ) ;
void halt_system ( void ) ;
void start_system( void ) ;
```

Funciones de servicio específico no relacionadas con control

```
void display_coord( char x, char y, int scale, long axis_val ) ;
void use_axis( char x, char y, char axis_enable, char scale, char
               units, char convt, long offset, long resolution,
               long *axis_val, volatile long int_axis, int
               TCud_PORT, int PULSE_PORT ) ;
void use_system( char set_enable ) ;
void alert( char *str ) ;
long get_integer( char dgs, char x, char y ) ;
char *get_file( char *str, char dgs, char x, char y ) ;
void touch_coords( long coord1, long coord2, long coord3 ) ;
```

```
void save_coords( void ) ;
void load( long *offset ) ;
void set_resolution( unsigned *resol ) ;
void print_units( char x, char y, char scale, char mm_plg, char
                  convt ) ;
void set_scale( char scale ) ;

Funciones de control de estado lógico del sistema

void state_decoder ( void ) ;
void state_driver( void ) ;

Definicion de funciones

[ Interrupción de hardware para carrys ]
void interrupt carry_3d( void )
  {
  geninterrupt( inportb( PC_LGC ) : 0xF0 ) ;
  outportb( PA_LGC , 0x00 ) ;
  outportb( 0x20 , 0x20 ) ;
 }

[ Interrupción de hardware para borrows ]
void interrupt borrow_3d( void )
  {
  geninterrupt( ~inportb( PB_LGC )) ;
  outportb( PA_LGC , 0x00 ) ;
  outportb( 0x20 , 0x20 ) ;
  }

[ Interrupciones de software para actualizar variables ]
void interrupt carry_x( void )
  {
  outportb( PA_LGC , 0x01 ) ;
  int_x++ ;
  }
```

```
void interrupt carry_y( void )
  {
  outportb( PA_LGC , 0x04 ) ;
  int_y++ ;
  }
void interrupt carry_z( void )
  {
  outportb( PA_LGC , 0x10 ) ;
  int_z++ ;
  }
void interrupt carry_xy( void )
  {
  outportb( PA_LGC , 0x05 ) ;
  int_x++ ;
  int_y++ ;
  }
void interrupt carry_xz( void )
  {
  outportb( PA_LGC , 0x11 ) ;
  int_x++ ;
  int_z++ ;
  }
void interrupt carry_yz( void )
  {
  outportb( PA_LGC , 0x14 ) ;
  int_y++ ;
  int_z++ ;
  }
void interrupt carry_xyz( void )
  {
  outportb( PA_LGC , 0x15 ) ;
  int_x++ ;
  int_y++ ;
  int_z++ ;
 }
```

```
void interrupt borrow_x( void )
  {
  outportb( PA_LGC , 0x02 ) ;
  int_x-- ;
  }
void interrupt borrow_y( void )
  {
  outportb( PA_LGC , 0x08 ) ;
  int_y-- ;
 }
void interrupt borrow_z( void )
  {
  outportb( PA_LGC , 0x20 ) ;
  int_z-- ;
  }
void interrupt borrow_xy( void )
  {
  outportb( PA_LGC , 0x0A ) ;
  int_x-- ;
  int_y-- ;
  }
void interrupt borrow_xz( void )
  {
  outportb( PA_LGC , 0x22 ) ;
  int_x-- ;
  int_z-- ;
  }
void interrupt borrow_yz( void )
  {
  outportb( PA_LGC , 0x28 ) ;
  int_y-- ;
  int_z-- ;
  }
```

```
void interrupt borrow_xyz( void )
  {
  outportb( PA_LGC , 0x2A ) ;
  int_x-- ;
  int_y-- ;
  int_z-- ;
  }

[ Borra las ventanas creadas para obtener los parámetros de las
  funciones ]
void windows_eraser( char wds )
  {
  switch( wds )
  {
  case 3 :
  case 2 : if ( window3_ptr != NULL )
             window_restore( 2, 11, 13, 16, &window3_ptr ) ;
           if ( window2_ptr != NULL )
             window_restore( 2, 11, 13, 16, &window2_ptr ) ;
           if ( wds == 2 ) break ;
  case 1 : if ( window1_ptr != NULL )
             window_restore( 3,  4, 12, 10, &window1_ptr ) ;
  }
  }

[ Creación de la ventana para seleccionar un eje(s) ]
void axis_window( void )
  {
  make_window( 3, 4, 12, 10, DARKGRAY, LIGHTGRAY ) ;
  set_window( 11, 5, 11, 9, DARKGRAY, BROWN ) ;
  xycputs( 4, 5 , "Eje    X") ;
  xycputs( 4, 6 , "Eje    Y") ;
  xycputs( 4, 7 , "Eje    Z") ;
  xycputs( 4, 8 , "SistemaS") ;
```

```
  xycputs( 4, 9 , "ESC") ;
  set_window( 4, axis_y, 11, axis_y, LIGHTGRAY, DARKGRAY ) ;
  }

[ Creación de la ventana para elegir la escala ]
void scale_window( void )
  {
  make_window( 2, 11, 13, 16, DARKGRAY, LIGHTGRAY ) ;
  set_window( 12, 12, 12, 15, DARKGRAY, BROWN ) ;
  xycputs( 3, 12, "1/1      0" ) ;
  xycputs( 3, 13, "1/1000   3" ) ;
  xycputs( 3, 14, "1/10000006" ) ;
  xycputs( 3, 15, "ESC" ) ;
  set_window( 3, scale_y, 12, scale_y, LIGHTGRAY, DARKGRAY ) ;
  }

[ Creación de la ventana para elegir las unidades ]
void units_window( void )
  {
  make_window( 2, 11, 13, 16, DARKGRAY, LIGHTGRAY ) ;
  set_window( 3, 12, 3, 14, DARKGRAY, BROWN ) ;
  xycputs( 3, 12 , "METROS" ) ;
  xycputs( 3, 13 , "PULGADAS" ) ;
  xycputs( 3, 14 , "CONVERSION" ) ;
  xycputs( 10, 15 , "ESC" ) ;
  set_window( 3, units_y, 12, units_y, LIGHTGRAY, DARKGRAY ) ;
  }

[ Creación del menú principal ]
void slct_window( void )
  {
  char k ;

  make_window( 2, 2, 13, 24, CYAN, DARKGRAY ) ;
  set_window( 6, 3, 9, 3, CYAN, BROWN ) ;
  for( k=4 ; k<=22 ; set_window( 3,k,5,k,CYAN,BROWN ), k+=2 ) ;
```

```
  xycputs( 6, 3 , "MENU" ) ;
  xycputs( 3, 4 , "F1" ) ;
  xycputs( 3, 5 , "TERMINAR" ) ;
  xycputs( 3, 6 , "F2" ) ;
  xycputs( 3, 7 , "INIC.SIST." ) ;
  xycputs( 3, 8 , "F3" ) ;
  xycputs( 3, 9 , "MEMORIZAR" ) ;
  xycputs( 3, 10, "F4" ) ;
  xycputs( 3, 11, "ARCHIVAR" ) ;
  xycputs( 3, 12, "F5" ) ;
  xycputs( 3, 13, "HABILITAR" ) ;
  xycputs( 3, 14, "F6" ) ;
  xycputs( 3, 15, "INIC. EJES" ) ;
  xycputs( 3, 16, "F7" ) ;
  xycputs( 3, 17, "PRECARGA" ) ;
  xycputs( 3, 18, "F8" ) ;
  xycputs( 3, 19, "RESOLUCION" ) ;
  xycputs( 3, 20, "F9" ) ;
  xycputs( 3, 21, "ESCALA" ) ;
  xycputs( 3, 22, "F10" ) ;
  xycputs( 3, 23, "UNIDADES" ) ;
  set_window( 3, slct_y, 12, slct_y, LIGHTGRAY, DARKGRAY ) ;
  }

[ Generación de la ventana de medición ]
void meassure_window( void )
  {
  make_window( 14, 2 , 39, 17, LIGHTBLUE, WHITE ) ;
  xycputs( 17, 3 , "SISTEMA DE MEDICION" ) ;
  xycputs( 19, 4 , "POR COORDENADAS") ;
  xycputs( 16, 7 , "X > ") ;
  xycputs( 16, 9 , "Y > ") ;
  xycputs( 16, 11, "Z > ") ;
  print_units( 34,  7, 6, X_units, X_convt ) ;
  print_units( 34,  9, 6, Y_units, Y_convt ) ;
  print_units( 34, 11, 6, Z_units, Z_convt ) ;
```

```
  xycputs( 15, 14, "PUNTOS" ) ;
  xycputs( 15, 15, "MEDIDOS :" ) ;
  set_window( 23 , 7 , 33, 11, LIGHTBLUE, YELLOW ) ;
  set_window( 24 , 15 , 37, 15, LIGHTBLUE, YELLOW ) ;
  xycputs( 24, 15, (char *) ltoa( (long)index, str_digit, 10 )) ;
  }

[ Creación de la ventana de mensajes ]
void message_window( void )
  {
  make_window( 14, 18, 39, 24, LIGHTGRAY, BLACK ) ;
  xycputs( 23, 18, "MENSAJES" ) ;
  }

[ Creación de la pantalla del sistema ]
void system_screen( void )
  {
  textmode( C40 ) ;
  box( 1, 1, 40, 25 ) ;
  slct_window() ;
  message_window() ;
  meassure_window() ;
  /* reconfiguración del cursor, modo no visible  */
  regsin.h.ch = 0x20 ;
  regsin.h.cl = 0x20 ;
  regsin.h.ah = 0x01 ;
  int86( 0x10, &regsin, &regsout ) ;
  }

[ Dibuja el cursor en las ventanas de selección ]
void menu_drv( char *y, char new_y, char x1, char x2, char x1_2,
               char x2_2, char BAK1, char FRNT1, char BAK2, char
               FRNT2, char FOPTION )
  {
  set_window( x1,    *y, x2,    *y, BAK1, FRNT1 ) ;
  set_window( x1_2, *y, x2_2, *y, BAK1, FOPTION ) ;
```

```
  set_window( x1, new_y, x2, new_y, BAK2, FRNT2 ) ;
  *y = new_y ;
  }

[ Determina si se trata de una tecla de control o función ]
char not_hotkey( void )
  {
  return ( (char)bioskey(1) ) ;
  }

[ Mensaje de error para los accesos a disco ]
int disk_error( void )
  {
  alert("ERROR EN EL DISCO") ;
  hardretn(0) ;
  }

[ Impresión de mensajes de error ]
void alert( char *str )
  {
  set_window( 14, 18, 39, 24, LIGHTGRAY, RED ) ;
  set_window( 15, 23, 38, 23, RED, WHITE ) ;
  xycputs( 15, 23, str ) ;
  putchar( BEEP ) ;
  sleep(3) ;
  window_clear( 15, 19, 38, 23 ) ;
  while ( bioskey( 1 ) )
          bioskey( 0 ) ;
  set_window( 14, 18, 39, 24, LIGHTGRAY, BLACK ) ;
  }

[ respaldo de los vectores de interrupción ]
void vect_backup( unsigned int vector, unsigned char llm,
      unsigned char *array, unsigned char back_off )
  {
  unsigned char k ;
```

```
  vector <<= 2 ;
  lim <<= 2 ;
  for ( k=0 ; k<lim  ; k++ )
     *(array+back_off+k) = *((char far *)(0x00000000+vector+k)) ;
  }

[ restablecimiento de los vectores de interrupcion ]
void  vect_restore(  unsigned  int  vector,  unsigned  char  lim,
                  unsigned char *array, unsigned char  back_off )
  {
  unsigned char k ;
  vector <<= 2 ;
  lim <<= 2 ;
  for ( k = 0 ; k < lim  ; k++ )
     *((char far *)( 0x00000000+vector+k )) = *(array+back_off+k) ;
  }

[ inicializa un eje a cero ]
void clear_axis( char axis, char mask1, char mask2, long *offset,
     volatile long *int_axis, long *axis_val )
  {
  disable() ;
    outportb( PC_LGC , 0x0F ) ;
    outportb( PA_LGC , 0x00 ) ;
    outportb( PC_LGC , mask1 ) ;
    outportb( PA_LGC , mask2 ) ;
    *offset = 0 ;
    *int_axis = 0 ;
    *axis_val = 0 ;
    outportb( PC_LGC , 0x0F ) ;
    outportb( PA_LGC , 0x00 ) ;
    window_clear( 23, axis*2+7, 33, axis*2+7 ) ;
    state = 0 ;
  enable() ;
  }
```

```
[ inicializa ejes coordenados a cero ]
void clear_coords( void )
    {
  clear_axis( X_axis, 0x1F, 0x03, &offset_x, &int_x, &axis_valx ) ;
  clear_axis( Y_axis, 0x2F, 0x0C, &offset_y, &int_y, &axis_valy ) ;
  clear_axis( Z_axis, 0x4F, 0x30, &offset_z, &int_z, &axis_valz ) ;
    }

[ inicializa sistema, determina valores iniciales ]
void init_system( void )
  {
  X_res = Y_res = Z_res = 1 ;
  clear_coords() ;
  set_enable = TRUE ;
  state = index = 0 ;
  slct_y = axis_y = 5 ;
  scale_y = units_y = 12 ;
  downptr = topptr = NULL ;
  heapptr = (void *)malloc( 1 ) ;
  X_scale  = Y_scale  = Z_scale  = 6 ;
  offset_x = offset_y = offset_z = 0 ;
  X_units = Y_units = Z_units = TRUE ;
  X_convt = Y_convt = Z_convt = FALSE ;
  window1_ptr = window2_ptr = window3_ptr = NULL ;
  maxcoords = ((unsigned int) coreleft() >> 4 ) - 1000 ;
  }
[ Restaura la configuración de la computara ]
void halt_system( void )
  {
  outportb( PC_LGC , 0x8F ) ;
  vect_restore( 0x000B, 1, isr_backup, 0  ) ;
  vect_restore( 0x000F, 1, isr_backup, 4  ) ;
  vect_restore( 0x0060, 7, isr_backup, 8  ) ;
  vect_restore( 0x00F1, 7, isr_backup, 36 ) ;
  }
```

```
[ Inicialización del Sistema de Medición por Coordenadas ]
void start_system( void )
  {
  disable () ;
/*  programación del 8259   */
    outportb( 0x20 , 0x13 ) ;
    outportb( 0x21 , 0x08 ) ;
    outportb( 0x21 , 0x09 ) ;
    outportb( 0x21 , 0x00 ) ;
/*   Programación del 8255  */
    outportb( PCTRL_PLS,  0x9B ) ;
    outportb( PCTRL_TCud, 0x9B ) ;
    outportb( PCTRL_LGC,  0x83 ) ;
/*  Respaldo de las rutinas de interrupción  */
    vect_backup( 0x000B, 1, isr_backup, 0  ) ;
    vect_backup( 0x000F, 1, isr_backup, 4  ) ;
    vect_backup( 0x0060, 7, isr_backup, 8  ) ;
    vect_backup( 0x00F1, 7, isr_backup, 36 ) ;
/*  Instalación de los vectores de interrupción   */
    setvect(  15, carry_3d ) ;
    setvect( 241, carry_z ) ;
    setvect( 242, carry_y ) ;
    setvect( 243, carry_yz ) ;
    setvect( 244, carry_x ) ;
    setvect( 245, carry_xz) ;
    setvect( 246, carry_xy ) ;
    setvect( 247, carry_xyz ) ;
    setvect(  11, borrow_3d ) ;
    setvect( 102, borrow_z ) ;
    setvect( 101, borrow_y ) ;
    setvect( 100, borrow_yz ) ;
    setvect(  99, borrow_x ) ;
    setvect(  98, borrow_xz ) ;
    setvect(  97, borrow_xy ) ;
    setvect(  96, borrow_xyz ) ;
```

```
 harderr( disk_error ) ;
 init_system() ;
 enable() ;
}

[ Obtención del número de pulsos ]
long get_axis_count( long resolution, volatile long int_axis,
                     int TCud_PORT, int PULSE_PORT  )
 {
 signed   char term_count  ;
 unsigned char pulse_count ;
 term_count  = inportb( TCud_PORT ) ;
term_count  = ( term_count & 0x0F ) -
              (( term_count & 0xF0 ) >> 4 ) ;
 pulse_count = inportb( PULSE_PORT ) ;
return( ( int_axis*2048+term_count*256+pulse_count )
                                      *resolution ) ;
 }

[ Despliegue del valor de un eje ]
void display_coord ( char x, char y, int scale, long axis_val )
 {
 signed char   k, fp, length ;

 length = strlen( ltoa( (long)axis_val, str_digit, 10 )) ;
 fp = length - scale ;
 if ( str_digit[0] == '-' ) /* valor negativo ? */
 {
 settexel_char( x++, y, '-' ) ;
 for( k=0 ; k<length ; str_digit[k] = str_digit[k+1], k++ ) ;
 length-- ;
 fp-- ;
 }
 if ( fp >= 0 )
 {
```

```
  if ( fp == 0 )
     settexel_char( x++, y, '0' ) ;
  for( k=0 ; k<fp ; settexel_char( x++, y, str_digit[k++] )) ;
  settexel_char( x++, y, '.' ) ;
  for( k=fp; k<length; settexel_char( x++, y, str_digit[k++]));
  }
  else
  {
  settexel_char( x++, y, '0' ) ;
  settexel_char( x++, y, '.' ) ;
  for( k=fp ; k < 0 ; settexel_char( x++, y, '0' ), k++ ) ;
  for( k=0; k < length; settexel_char( x++, y, str_digit[k++]));
  }
  for( k=0 ; x <= 33 ; settexel_char( x++, y, '0' ), k++ ) ;
  }

[ Actualiza las variables de un eje ]
void use_axis( char x, char y, char axis_enable, char scale, char
               units, char convt, long offset, long resolution,
               long *axis_val,     volatile long int_axis, int
               TCud_PORT, int PULSE_PORT )
  {
  if ( axis_enable )
  {
  *axis_val = (long) (get_axis_count( resolution, int_axis,
                     TCud_PORT, PULSE_PORT) + offset ) ;
  if ( convt )
   if ( units )
      *axis_val =  *axis_val / 0.0254 ;
   else *axis_val =  *axis_val * 0.0254 ;
  display_coord( x, y, scale, *axis_val ) ;
  }
  }
```

```
[ Actualiza las coordenadas ]
void use_system( char set_enable )
  {
 use_axis( 23,7,set_enable,X_scale,X_units,X_convt,offset_x,X_res,
          &axis_valx,int_x,PA_TCud,PA_PLS ) ;
 use_axis( 23,9,set_enable,Y_scale,Y_units,Y_convt,offset_y,Y_res,
          &axis_valy,int_y,PB_TCud,PB_PLS ) ;
use_axis(23,11,set_enable,Z_scale,Z_units,Z_convt,offset_z,Z_res,
        &axis_valz,int_z,PC_TCud,PC_PLS ) ;
  }

[ Obtiene un digito entero ]
long get_integer( char dgs, char x, char y )
  {
  int   c ;
  char  k, sign ;
  long  digit ;
  sign = 1 ;
  digit = k = 0 ;
  settexel_char( x-1, y, '<' ) ;
  settexel_char( x+dgs, y, '>' ) ;
  do {
  if ( bioskey(1) )
     {
     if ( not_hotkey() )
     {
     c = bioskey(0) ;
     if ( isdigit( (char) c ))
     {
     if ( dgs == 0 )
     break ;
     settexel_char( x+(k++), y, (char) c ) ;
     digit = digit*10 + (char)( c-0x30 ) ;
     dgs-- ;
     }
```

```
else
switch ( c )
  {
  case PLUS1      :
  case PLUS2      : if ( k == 0 )
     {
     settexel_char( x+dgs, y, ' ' ) ;
     settexel_char( x+dgs+1, y, '>' ) ;
     settexel_char( x+(k++), y, '+' ) ;
     }
     else
     {
     alert( "DIGITO ERRONEO" ) ;
     return(0) ;
     }
     break ;
  case MINUS1     :
  case MINUS2     : if ( k == 0 )
     {
     settexel_char( x+dgs, y, ' ' ) ;
     settexel_char( x+dgs+1, y, '>' ) ;
     settexel_char( x+(k++), y, '-' ) ;
     sign = -1 ;
     }
     else
     {
     alert( "DIGITO ERRONEO" ) ;
     return(0) ;
     }
     break ;
  case RETURN     : set_window( 14, 18, 39, 24,
       LIGHTGRAY, BLACK ) ;
     window_clear( 15, 19, 38, 23 ) ;
     return( digit*sign ) ;
```

```
      case ESC        : window_clear( 15, 19, 38, 23 ) ;
        return(0) ;
      case BACK_SPACE : if ( k )
        {
        settexel_char( x+(--k), y, ' ' ) ;
        break ;
        }
      default     : settexel_char( x+(k++), y, (char) c ) ;
        alert( "DIGITO ERRONEO" ) ;
        return(0) ;
      }
    }
    else
    {
            window_clear( 15, 19, 38, 23 ) ;
    return(0) ;
    }
    }
  use_system( set_enable ) ;
  }
  while ( dgs >= 0 ) ;
  alert( "CANT. DEMASIADO GRANDE" ) ;
  return(0) ;
  }


[ Obtiene un nombre de archivo ]
char *get_file( char *str, char lts, char x, char y )
  {
  char          k ;
  int           c ;

  k = 0 ;
  settexel_char( x-1, y, '<' ) ;
  settexel_char( x+lts, y, '>' ) ;
```

```
do {
if ( bioskey(1) )
   {
   if ( not_hotkey() )
   {
   c = bioskey(0) ;
   if ( isalnum( (char) c ))
   {
   if ( k < lts )
   {
   settexel_char( x+k, y, (char) c ) ;
   *( str + (k++) ) = (char) c ;
   }
   else
   {
   alert( "NOMBRE DEMASIADO LARGO" ) ;
   return(NULL) ;
   }
   }
   else
   switch ( c )
     {
     case UNDERSCORE : settexel_char( x+k, y, '_' ) ;
        if ( k < 15 )
        *( str + (k++) ) = '_' ;
        break ;
     case RETURN     : str[k++] = '.' ;
        str[k++] = '3' ;
        str[k++] = 'D' ;
        str[k] = NULL ;
        return( str ) ;
     case ESC        : window_clear( x, y, x+lts, y ) ;
        return(NULL) ;
```

```
      case TWO_POINTS : if ( k == 1 )
         {
         settexel_char( x+k, y, ':' ) ;
         *( str + (k++)) = ':' ;
         break ;
         }
      case ROOT_MARK  : settexel_char( x+k, y, '\' ) ;
         if ( k < 15 )
         *( str + (k++)) = '\' ;
         break ;
      case BACK_SPACE : if ( k )
         {
         settexel_char( x+(--k), y, ' ' ) ;
         break ;
         }
      default         : if ( k < lts )
         settexel_char( x+k, y, (char) c ) ;
         alert( "CARACTER ERRONEO" ) ;
         return(NULL) ;
      }
    }
    else return( NULL ) ;
    }
  use_system( set_enable ) ;
  }
  while ( k <= lts ) ;
  alert( "NOMBRE DEMASIADO LARGO" ) ;
  return(NULL) ;
  }


[ Memoriza las coordenadas de un punto ]
void touch_coords( long coord1, long coord2, long coord3 )
  {
  coords = (struct list *) malloc( sizeof( struct list )) ;
```

```
  if ( coords != NULL )
  {
  if ( ( downptr == NULL ) && ( index == 0 ))
     downptr = coords ;
  else
     topptr -> link = coords ;
  coords -> x    = coord1 ;
  coords -> y    = coord2 ;
  coords -> z    = coord3 ;
  coords -> link = NULL ;
  topptr = coords ;
  index++ ;
  xycputs( 24, 15, (char *) ltoa( (long)index, str_digit, 10 )) ;
  }
  else
  alert( "MEMORIA INSUFICIENTE" ) ;
  }


[ Graba las coordenadas memorizadas en un archivo ]
void save_coords( void )
  {
  long i ;

  xycputs(20, 20, "! ARCHIVO : ") ;
  if ( get_file( file_ext, 22, 16, 22 ) != NULL )
  {
  file_int = fopen( (const char *) &file_ext, "w+" ) ;
  if ( file_int != NULL )
     {
     enable_signal( FALSE ) ;
     enable_display( FALSE ) ;
     window_clear( 15, 19, 38, 23 ) ;
```

```
   for ( i = 1  ;  i <= index ; i++ )
    {
    topptr =downptr ;
    if ( fprintf( file_int, "%05u ", i ) != EOF )
    if ( fprintf( file_int, "%12E %12E %12E",
      (float) downptr->x,
      (float) downptr->y,
      (float) downptr->z ) != EOF )
    {
    downptr = downptr -> link ;
    free( topptr ) ;
    xycputs( 24, 15, (char *) ltoa( i, str_digit, 10 )) ;
    }
    else i-- ;
    else i-- ;
    }
   if ( fclose( file_int ) != EOF )
   {
   index = 0 ;
   textbackground( LIGHTGRAY ) ;
   window_clear( 15, 19, 38, 23 ) ;
   window_clear( 24, 15, 36, 15 ) ;
   xycputs( 24, 15, (char *)ltoa((long)index, str_digit, 10)) ;
   }
 else
   alert( "ERROR AL CERRAR ARCHIVO") ;
   enable_signal( set_enable ) ;
   enable_display( set_enable ) ;
   }
else
   alert( "ACCESO NO PERMITIDO") ;
}
else window_clear( 15, 19, 38, 23 ) ;
}
```

```
[ Habilita o inhibe las señales de los codificadores ]
void enable_signal( char ok )
  {
  if ( ok )
  outportb( PC_LGC , 0x0F ) ;
  else
  outportb( PC_LGC , 0x8F ) ;
  }

[ Habilita o inhibe el despliegue de las coordenadas ]
void enable_display( char ok )
  {
  if ( ok ) set_window( 21 , 7 , 33, 11, LIGHTBLUE, YELLOW ) ;
  else set_window( 23 , 7 , 33, 11, LIGHTBLUE, LIGHTBLUE ) ;
  }

[ Obtiene un valor de offset para algun(os) eje(s) ]
void load( long *offset )
  {
  long new_offset ;
  xycputs( 16, 19, "Valor en Millonésimas." ) ;
  xycputs( 16, 20, "(solo Números Enteros)") ;
  new_offset = get_integer( 9, 22, 22 ) ;
  if ( !bioskey(1) )
  {
  *offset = new_offset ;
  windows_eraser(1) ;
  state = 0 ;
  }
  else
  {
  window_clear( 15, 19, 38, 23 ) ;
  state = 7 ;
  }
  }
```

```
[ Determina la resolución de los codificadores ópticos ]
void set_resolution( unsigned *resol )
  {
  unsigned new_res ;
  xycputs( 15, 19, "Resol. en Millonésimas." ) ;
  xycputs( 15, 20, "(solo Números Enteros)") ;
  new_res = get_integer( 4, 22, 22 ) ;
  if ( bioskey(1) )
  {
  window_clear( 15, 19, 38, 23 ) ;
  state = 8 ;
  }
  else
  {
  if ( new_res > 0 ) *resol = new_res ;
  else if ( new_res < 0 )
     alert( "RESOLUCION ERRONEA" ) ;
  windows_eraser(1) ;
  state = 0 ;
  }
  }

[ Imprime las unidades de medición de c/u de los ejes ]
void print_units( char x, char y, char scale, char mm_plg, char
                  convt )
  {
  if ( convt ) set_window( x, y, x+5, y, LIGHTBLUE, BROWN ) ;
  else set_window( x, y, x+5, y, LIGHTBLUE, WHITE ) ;
  switch( scale )
  {
  case 0 : if (( mm_plg )^( convt ))
              xycputs( x, y, " µm. ") ;
           else xycputs( x, y, " µin.") ;
           break ;
```

```
  case 3 : if (( mm_plg )^( convt ))
              xycputs( x, y, " mm. ") ;
           else xycputs( x, y, " min.") ;
           break ;
  case 6 : if (( mm_plg )^( convt ))
              xycputs( x, y, " m. ") ;
           else xycputs( x, y, " in.") ;
           break ;
   }
  }


[ Obtiene la escala de medición ]
void scale_units( char *x, char *y, char *z, char val )
  {
  switch ( axis_y )
    {
    case 5 : if ( state == 51 )
                *x ^= val ;
             else *x  = val;
             print_units( 34,  7, X_scale, X_units, X_convt ) ;
             break ;
    case 6 : if ( state == 51 )
                *y ^= val ;
             else *y  = val ;
             print_units( 34,  9, Y_scale, Y_units, Y_convt ) ;
             break ;
    case 7 : if ( state == 51 )
                 *z ^= val ;
             else *z  = val ;
             print_units( 34, 11, Z_scale, Z_units, Z_convt ) ;
             break ;
    case 8 : if ( state == 51 )
                *x = *y = *z ^= val ;
             else *x = *y = *z =   val ;
```

```
            print_units( 34,  7, X_scale, X_units, X_convt ) ;
            print_units( 34,  9, Y_scale, Y_units, Y_convt ) ;
            print_units( 34, 11, Z_scale, Z_units, Z_convt ) ;
            break ;
 }
  windows_eraser(3) ;
  state = 0 ;
  }


[Decodifica el estado siguiente, en función de la tecla oprimida]
void state_decoder( void )
  {
  switch( bioskey(0) )
  {
  case   F1     : state = 1  ; break ;
  case   F2     : state = 2  ; break ;
  case   F3     : state = 3  ; break ;
  case   F4     : state = 4  ; break ;
  case   F5     : state = 5  ; break ;
  case   F6     : state = 6  ; break ;
  case   F7     : state = 7  ; break ;
  case   F8     : state = 8  ; break ;
  case   F9     : state = 9  ; break ;
  case   F10    : state = 10 ; break ;

  case   key_0  : if ( state == 0 )
        state = 54 ;
      else if ( state <= 10 )
        state = 55 ;
      else if (( state <= 45 )&&( !( state & 0x01 )))
        state = 56 ;
      else if ( state & 0x01 )
        state = 46 ; break ;
```

```
case   key_3  : if ( state == 0 )
       state = 54 ;
     else if ( state <= 10 )
       state = 55 ;
     else if (( state <= 45 )&&( !( state & 0x01 )))
       state = 56 ;
     else if ( state & 0x01 )
       state = 47 ; break ;
case   key_6  : if ( state == 0 )
       state = 54 ;
     else if ( state <= 10 )
       state = 55 ;
     else if (( state <= 45 )&&( !( state & 0x01 )))
       state = 56 ;
     else if ( state & 0x01 )
       state = 48 ; break ;
case   M_key  :
case   m_key  : if ( state == 0 )
       state = 54 ;
     else if ( state <= 10 )
       state = 55 ;
     else if (( state <= 45 )&&( state & 0x01 ))
       state = 56 ;
     else if ( !( state & 0x01 ))
       state = 49 ; break ;
case   P_key  :
case   p_key  : if ( state == 0 )
       state = 54 ;
     else if ( state <= 10 )
       state = 55 ;
     else if (( state <= 45 )&&( state & 0x01 ))
       state = 56 ;
     else if ( !( state & 0x01 ))
       state = 50 ; break ;
```

```
case   C_key  :
case   c_key  : if ( state == 0 )
      state = 54 ;
    else if ( state <= 10 )
      state = 55 ;
    else if (( state <= 45 )&&( state & 0x01 ))
      state = 56 ;
    else if ( !( state & 0x01 ))
      state = 51 ; break ;
case   X_key  :
case   x_key  : if ( state == 0 )
      state = 54 ;
    else if (( state > 5 ) && ( state < 11 ))
      state +=  6 ;
    else state = 56 ; break ;
case   Y_key  :
case   y_key  : if ( state == 0 )
      state = 54 ;
    else if (( state > 5 ) && ( state < 11 ))
      state += 12 ;
    else state = 56 ; break ;
case   Z_key  :
case   z_key  : if ( state == 0 )
      state = 54 ;
    else if (( state > 5 ) && ( state < 11 ))
      state += 18 ;
    else state = 56 ; break ;
case   S_key  :
case   s_key  : if ( state == 0 )
      state = 54 ;
    else if (( state > 5 ) && ( state < 11 ))
      state += 24 ;
    else state = 56 ; break ;
```

```
case   ESC    : if ( state == 0 )
          state = 1 ;
        else if ( state <= 10 )
          state = 35 ;
        else if ( state & 0x01 )
          state = 36 ;
        else state = 37 ; break ;

case UP_ARROW : if ( state == 0 )
          state = 38 ;          /* in menu  */
        else if ( state <= 10 )
          state = 39 ;          /* in axes  */
        else if ( state & 0x01 )
          state = 40 ;          /* in scale */
        else state = 41 ; break ;

case DW_ARROW : if ( state == 0 )
          state = 42  ;
        else if ( state <= 10 )
          state = 43 ;
        else if ( state & 0x01 )
          state = 44 ;
        else state = 45 ; break ;

case RETURN   : if ( state == 0 )
           switch( slct_y )
          {
          case 5  : state = 1  ; break ;
          case 7  : state = 2  ; break ;
          case 9  : state = 3  ; break ;
          case 11 : state = 4  ; break ;
          case 13 : state = 5  ; break ;
          case 15 : state = 6  ; break ;
          case 17 : state = 7  ; break ;
```

```
        case 19 : state = 8  ; break ;
        case 21 : state = 9  ; break ;
        case 23 : state = 10 ; break ;
        }
      else if ( state <= 10 )
        switch( axis_y )
          {
          case 5 : state += 6 ; break ;
          case 6 : state += 12 ; break ;
          case 7 : state += 18 ; break ;
          case 8 : state += 24 ; break ;
          case 9 : state = 35 ; break ;
          }
      else if ( state & 0x01 )
        switch( scale_y )
          {
          case 12 : state = 46 ; break ;
          case 13 : state = 47 ; break ;
          case 14 : state = 48 ; break ;
          case 15 : state = 36 ; break ;
          }
      else switch( units_y )
          {
          case 12 : state = 49 ; break ;
          case 13 : state = 50 ; break ;
          case 14 : state = 51 ; break ;
          case 15 : state = 37 ; break ;
          }
      break ;
  default       : putchar( BEEP ) ;
    }
}
```

```
[ Sección de salida de los estados del control finito ]
void state_driver( void )

  {
  switch( state )
    {
    case 1 : windows_eraser(3) ;
    menu_drv( &slct_y, 5, 3, 12, 3, 3,CYAN, DARKGRAY, LIGHTGRAY,
              DARKGRAY, DARKGRAY ) ;
    window_clear( 15, 19, 38, 23 ) ;
    xycputs( 15, 21, "¿ TERMINAR ? [S/N] S" ) ;
    do{
      if ( bioskey(1) )
      if ( not_hotkey() )
      switch ( bioskey(0) )
        {
        case S_key :
        case s_key :
        case RETURN: disable() ;
            halt_system() ;
            free( (void *)heapptr ) ;
            textmode( C80 ) ;
            clrscr() ;
            exit() ;
        case N_key :
        case n_key :
        case ESC   : state = 0 ;
          break ;
        default    : putchar( BEEP ) ;
        }
      else state = 0 ;
      use_system( set_enable ) ;
      }
    while ( state == 1 ) ;
    window_clear( 15, 21, 35, 21 ) ;
    break ;
```

```
case 2 : disable() ;
            outportb( PC_LGC , 0x8F ) ;
 windows_eraser(3) ;
            free( (void *)heapptr ) ;
 start_system() ;
 system_screen() ;
 state = 0 ;
 outportb( PC_LGC , 0x0F ) ;
 enable() ;
 break ;
case 3  : disable() ;
 enable_signal( FALSE ) ;
 windows_eraser(3) ;
 menu_drv( &slct_y, 9, 3, 12, 3, 3,
     CYAN, DARKGRAY, LIGHTGRAY, DARKGRAY, DARKGRAY ) ;
 if ( index < maxcoords )
 {
 touch_coords( axis_valx, axis_valy, axis_valz ) ;
 state = 0 ;
 enable_signal( set_enable ) ;
 enable() ;
 break ;
 }
 else
 alert( "MEMORIA INSUFICIENTE" ) ;
case 4  : disable() ;
 windows_eraser(3) ;
 menu_drv( &slct_y, 11, 3, 12, 3, 3,
     CYAN, DARKGRAY, LIGHTGRAY, DARKGRAY, DARKGRAY ) ;
            window_clear( 15, 19, 38, 23 ) ;
 save_coords() ;
            state = 0 ;
 enable() ;
 break ;
```

```
case 5  : set_enable = !set_enable ;
 enable_signal( set_enable ) ;
 enable_display( set_enable ) ;
 windows_eraser(3) ;
 menu_drv( &slct_y, 13, 3, 12, 3, 3,
     CYAN, DARKGRAY, LIGHTGRAY, DARKGRAY, DARKGRAY ) ;

 state = 0 ;
 break ;
case 6  :
case 7  :
case 8  :
case 9  :
case 10 : menu_drv( &slct_y, (state<<1)+3, 3, 12, 3, 3,
     CYAN, DARKGRAY, LIGHTGRAY, DARKGRAY, DARKGRAY ) ;
 windows_eraser(2) ;
 if ( window1_ptr == NULL )
 {
 window_save( 3, 4, 12, 10, &window1_ptr ) ;
 axis_window() ;
 }
 break ;
case 12 : menu_drv( &axis_y, 5, 4, 11, 11, 11, DARKGRAY,
                    LIGHTGRAY, LIGHTGRAY, DARKGRAY, BROWN ) ;
 clear_axis( X_axis, 0x1F, 0x03, &offset_x,
    &int_x, &axis_valx ) ;
 windows_eraser(1) ;
 break ;
case 13 : menu_drv( &axis_y, 5, 4, 11, 11, 11, DARKGRAY,
                    LIGHTGRAY, LIGHTGRAY, DARKGRAY, BROWN ) ;
 load( &offset_x ) ;
 break ;
case 14 : menu_drv( &axis_y, 5, 4, 11, 11, 11, DARKGRAY,
                    LIGHTGRAY, LIGHTGRAY, DARKGRAY, BROWN ) ;
```

```
 set_resolution( &X_res ) ;
 break ;
case 15 : menu_drv( &axis_y, 5, 4, 11, 11, 11, DARKGRAY,
                    LIGHTGRAY, LIGHTGRAY, DARKGRAY, BROWN ) ;
 if ( window2_ptr == NULL )
 {
 window_save( 2, 11, 13, 16, &window2_ptr ) ;
 scale_window() ;
 }
 state = ( ( slct_y-3 ) >> 1 ) + ( axis_y-4 )*6 ;
 break ;
case 16 : menu_drv( &axis_y, 5, 4, 11, 11, 11, DARKGRAY,
                    LIGHTGRAY, LIGHTGRAY, DARKGRAY, BROWN ) ;
 if ( window3_ptr == NULL )
 {
 window_save( 2, 11, 13, 16, &window3_ptr ) ;
 units_window() ;
 }
            state = ( ( slct_y-3 ) >> 1 ) +  ( axis_y-4 )*6 ;
 break ;
case 18 : menu_drv( &axis_y, 6, 4, 11, 11, 11, DARKGRAY,
                    LIGHTGRAY, LIGHTGRAY, DARKGRAY, BROWN ) ;
 clear_axis( Y_axis, 0x2F, 0x0C, &offset_y,
    &int_y, &axis_valy ) ;
 windows_eraser(1) ;
 break ;
case 19 : menu_drv( &axis_y, 6, 4, 11, 11, 11, DARKGRAY,
                    LIGHTGRAY, LIGHTGRAY, DARKGRAY, BROWN ) ;
 load( &offset_y ) ;
 break ;
case 20 : menu_drv( &axis_y, 6, 4, 11, 11, 11, DARKGRAY,
                    LIGHTGRAY, LIGHTGRAY, DARKGRAY, BROWN ) ;
 set_resolution( &Y_res ) ;
 break ;
```

```
case 21 : menu_drv( &axis_y, 6, 4, 11, 11, 11, DARKGRAY,
                    LIGHTGRAY, LIGHTGRAY, DARKGRAY, BROWN ) ;
 if ( window2_ptr == NULL )
 {
 window_save( 2, 11, 13, 16, &window2_ptr ) ;
                scale_window() ;
 }
 state = ( ( slct_y-3 ) >> 1 ) + ( axis_y-4 )*6 ;
 break ;
case 22 : menu_drv( &axis_y, 6, 4, 11, 11, 11, DARKGRAY,
                    LIGHTGRAY, LIGHTGRAY, DARKGRAY, BROWN ) ;
 if ( window3_ptr == NULL )
 {
 window_save( 2, 11, 13, 16, &window3_ptr ) ;
 units_window() ;
 }
 state = ( ( slct_y-3 ) >> 1 ) + ( axis_y-4 )*6 ;
 break ;
 case 24 : menu_drv( &axis_y, 7, 4, 11, 11, 11, DARKGRAY,
                    LIGHTGRAY, LIGHTGRAY, DARKGRAY, BROWN ) ;
 clear_axis( Z_axis, 0x4F, 0x30, &offset_z,
    &int_z, &axis_valz ) ;
 windows_eraser(1) ;
 break ;
 case 25 : menu_drv( &axis_y, 7, 4, 11, 11, 11, DARKGRAY,
                    LIGHTGRAY, LIGHTGRAY, DARKGRAY, BROWN ) ;
 load( &offset_z ) ;
 break ;
 case 26 : menu_drv( &axis_y, 7, 4, 11, 11, 11, DARKGRAY,
                    LIGHTGRAY, LIGHTGRAY, DARKGRAY, BROWN ) ;
 set_resolution( &Z_res ) ;
 break ;
 case 27 : menu_drv( &axis_y, 7, 4, 11, 11, 11, DARKGRAY,
                    LIGHTGRAY, LIGHTGRAY, DARKGRAY, BROWN ) ;
```

```
  if ( window2_ptr == NULL )
  {
  window_save( 2, 11, 13, 16, &window2_ptr ) ;
                scale_window() ;
  }
  state = ( ( slct_y-3 ) >> 1 ) +  ( axis_y-4 )*6 ;
  break ;
case 28 : menu_drv( &axis_y, 7, 4, 11, 11, 11, DARKGRAY,
                    LIGHTGRAY, LIGHTGRAY, DARKGRAY, BROWN ) ;
  if ( window3_ptr == NULL )
  {
  window_save( 2, 11, 13, 16, &window3_ptr ) ;
  units_window() ;
  }
  state = ( ( slct_y-3 ) >> 1 ) +  ( axis_y-4 )*6 ;

  break ;
 case 30 : menu_drv( &axis_y, 8, 4, 11, 11, 11, DARKGRAY,
                    LIGHTGRAY, LIGHTGRAY, DARKGRAY, BROWN ) ;
  clear_coords() ;
  windows_eraser(1) ;
  break ;
 case 31 : menu_drv( &axis_y, 8, 4, 11, 11, 11, DARKGRAY,
                    LIGHTGRAY, LIGHTGRAY, DARKGRAY, BROWN ) ;
  load( &offset_x ) ;
  offset_z = offset_y = offset_x ;
  break ;
 case 32 : menu_drv( &axis_y, 8, 4, 11, 11, 11, DARKGRAY,
                    LIGHTGRAY, LIGHTGRAY, DARKGRAY, BROWN ) ;
  set_resolution( &X_res ) ;
  Z_res = Y_res = X_res ;
  break ;
 case 33 : menu_drv( &axis_y, 8, 4, 11, 11, 11, DARKGRAY,
                    LIGHTGRAY, LIGHTGRAY, DARKGRAY, BROWN ) ;
```

```
 if ( window2_ptr == NULL )
 {
 window_save( 2, 11, 13, 16, &window2_ptr ) ;
                scale_window() ;
 }
 state = ( ( slct_y-3 ) >> 1 ) + ( axis_y-4 )*6 ;
 break ;
case 34 : menu_drv( &axis_y, 8, 4, 11, 11, 11, DARKGRAY,
                    LIGHTGRAY, LIGHTGRAY, DARKGRAY, BROWN ) ;
 if ( window3_ptr == NULL )
 {
 window_save( 2, 11, 13, 16, &window3_ptr ) ;
 units_window() ;
 }
 state = ( ( slct_y-3 ) >> 1 ) + ( axis_y-4 )*6 ;
 break ;
case 35 : windows_eraser(1) ;
 state = 0 ;
 break ;
case 36 : window_restore( 2, 11, 13, 16, &window2_ptr ) ;
 state = ( slct_y-3 ) >> 1 ;
 break ;
case 37 : window_restore( 2, 11, 13, 16, &window3_ptr ) ;
 state = ( slct_y-3 ) >> 1 ;
 break ;
case 38 : if ( slct_y == 5 )
   menu_drv( &slct_y, 23, 3, 12, 3, 3, CYAN, DARKGRAY,
    LIGHTGRAY, DARKGRAY, DARKGRAY ) ;
 else menu_drv( &slct_y, slct_y-2, 3, 12, 3, 3, CYAN,
    DARKGRAY, LIGHTGRAY, DARKGRAY, DARKGRAY ) ;
 state = 0 ;
 break ;
case 39 : if ( axis_y == 5 )
   menu_drv( &axis_y, 9, 4, 11, 11, 11, DARKGRAY,
    LIGHTGRAY, LIGHTGRAY, DARKGRAY, BROWN ) ;
```

```
 else menu_drv( &axis_y, axis_y-1, 4, 11, 11, 11, DARKGRAY,
                LIGHTGRAY, LIGHTGRAY, DARKGRAY, BROWN ) ;
 state = ( slct_y-3 ) >> 1 ;
 break ;
case 40 : if ( scale_y == 12 )
 menu_drv( &scale_y, 15, 3, 12, 12, 12, DARKGRAY,
          LIGHTGRAY, LIGHTGRAY, DARKGRAY, BROWN ) ;
 else menu_drv( &scale_y, scale_y-1, 3, 12, 12, 12, DARKGRAY,
                LIGHTGRAY, LIGHTGRAY, DARKGRAY, BROWN ) ;
 state = ( ( slct_y-3 ) >> 1 ) +  ( axis_y-4 )*6 ;
 break ;
case 41 : if ( units_y == 12 )
 menu_drv( &units_y, 15, 3, 12, 3, 3, DARKGRAY,
     LIGHTGRAY, LIGHTGRAY, DARKGRAY, BROWN ) ;
 else menu_drv( &units_y, units_y-1, 3, 12, 3, 3, DARKGRAY,
    LIGHTGRAY,LIGHTGRAY, DARKGRAY, BROWN ) ;
 state = ( ( slct_y-3 ) >> 1 ) +  ( axis_y-4 )*6 ;
 break ;
case 42 : if ( slct_y == 23 )
   menu_drv( &slct_y, 5, 3, 12, 3, 3, CYAN, DARKGRAY,
    LIGHTGRAY, DARKGRAY, DARKGRAY ) ;
 else menu_drv( &slct_y, slct_y+2, 3, 12, 3, 3, CYAN,
    DARKGRAY, LIGHTGRAY, DARKGRAY, DARKGRAY ) ;
 state = 0 ;
 break ;
case 43 : if ( axis_y == 9 )
   menu_drv( &axis_y, 5, 4, 11, 11, 11, DARKGRAY,
    LIGHTGRAY, LIGHTGRAY, DARKGRAY, BROWN ) ;
 else menu_drv( &axis_y, axis_y+1, 4, 11, 11, 11, DARKGRAY,
    LIGHTGRAY, LIGHTGRAY, DARKGRAY, BROWN ) ;
 state = ( slct_y-3 ) >> 1 ;
 break ;
case 44 : if ( scale_y == 15)
   menu_drv( &scale_y, 12, 3, 12, 12, 12, DARKGRAY,
    LIGHTGRAY, LIGHTGRAY, DARKGRAY, BROWN ) ;
```

```
    else menu_drv( &scale_y, scale_y+1, 3,12, 12, 12, DARKGRAY,
       LIGHTGRAY, LIGHTGRAY, DARKGRAY, BROWN ) ;
    state = ( ( slct_y-3 ) >> 1 ) +  ( axis_y-4 )*6 ;
    break ;
   case 45 : if ( units_y == 15 )
      menu_drv( &units_y, 12, 3, 12, 3, 3, DARKGRAY,
       LIGHTGRAY, LIGHTGRAY, DARKGRAY, BROWN ) ;
    else menu_drv( &units_y, units_y+1, 3, 12, 3, 3, DARKGRAY,
       LIGHTGRAY, LIGHTGRAY, DARKGRAY, BROWN ) ;
    state = ( ( slct_y-3 ) >> 1 ) +  ( axis_y-4 )*6 ;
    break ;
   case 46 : menu_drv( &scale_y, 12, 4, 12, 11, 11, DARKGRAY,
                       LIGHTGRAY,   LIGHTGRAY, DARKGRAY, BROWN);
    scale_units( &X_scale, &Y_scale, &Z_scale, 6 ) ;
    break ;
  case 47 : menu_drv( &scale_y, 13, 4, 12, 11, 11, DARKGRAY,
                      LIGHTGRAY,   LIGHTGRAY, DARKGRAY, BROWN ) ;
    scale_units( &X_scale, &Y_scale, &Z_scale, 3 ) ;
    break ;
  case 48 : menu_drv( &scale_y, 14, 4, 12, 11, 11, DARKGRAY,
                      LIGHTGRAY,   LIGHTGRAY, DARKGRAY, BROWN ) ;
    scale_units( &X_scale, &Y_scale, &Z_scale, 0 ) ;
    break ;
   case 49 : menu_drv( &units_y, 12, 3, 12, 3, 3, DARKGRAY,
                      LIGHTGRAY,   LIGHTGRAY, DARKGRAY, BROWN ) ;
    scale_units( &X_units, &Y_units, &Z_units, TRUE ) ;
    break ;
   case 50 : menu_drv( &units_y, 13, 3, 12, 3, 3, DARKGRAY,
                      LIGHTGRAY,   LIGHTGRAY, DARKGRAY, BROWN ) ;
    scale_units( &X_units, &Y_units, &Z_units, FALSE) ;
    break ;
  case 51 : menu_drv( &units_y, 14, 3, 12, 3, 3, DARKGRAY,
                      LIGHTGRAY,   LIGHTGRAY, DARKGRAY, BROWN ) ;
    scale_units( &X_convt, &Y_convt, &Z_convt, TRUE ) ;
    break ;
```

```
    case 54 : putchar( BEEP ) ;
     state = 0 ;
     break ;
    case 55 : putchar( BEEP ) ;
     state = ( slct_y-3 ) >> 1 ;
     break ;
    case 56 : state = ( ( slct_y-3 ) >> 1 ) +  ( axis_y-4 )*6 ;
                putchar( BEEP ) ;
    }
  use_system( TRUE ) ;
  }

[ Programa principal ]
void main( void )
  {

  start_system() ;
  system_screen() ;

  while ( TRUE )
     {
     use_system( set_enable ) ;
     if ( bioskey(1) )
     {
     state_decoder() ;
     if ( state )
       state_driver() ;
     }
     }
  }
```

BIBLIOGRAFIA

Hopcroft, J. E.; Ullman J. D., Introduction to Automatization Theory, Languajes and Computation, EUA, 1979.

Leblanc, G., Turbo C para IBM-PC y compatibles, Colección Informática de Gestión, Barcelona, 1988.

Márquez, V.; García, E., Memorias VI Congreso Nacional de Instrumentación, "Sistema de Medición por Coordenadas", SOMI, México, 1990.

Mosich, D.; Shammmas, N.; Flamig, B., Advanced Turbo C Programmer's Guide, Wiley, EUA, 1988.

Tremblay, J.; Sorenson, P. G., The Theory and Practice of Compiler Writing, Mc Graw-Hill, Singapore, 1985.

CMOS Logic Data, 2a ed., Motorola Inc., EUA, 1985.

Microprocesor and Peripherial Handbook, Vols. I y II, Intel, EUA, 1987.

Techniques for Digitazing Rotary and Linear Motion, Dynamics Research Corporation, EUA, 1976.