54
24.

Universidad Nacional Autónoma de México.

Escuela Nacional de Estudios Profesionales
"Aragón"

INSTRUMENTACIÓN DE UN PRODUCTO DE SOFTWARE PARA EL CONTROL DE UN EQUIPO DE DESARROLLO "X-Y"

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

QUE PARA OBTENER EL TÍTULO DE:

INGENIERO EN COMPUTACIÓN

P R E S E N T A:

JAVIER FELIPE REYES DAMIÁN

E. N. E. P.

ARAGÓN

ASESOR DE TESIS.
ING. ERNESTO PEÑALOZA ROMERO

ASESOR TÉCNICO.
ING. GERARDO CASTRO ZAVALA

SAN JUAN DE ARAGÓN 1997

UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

ESCUELA NACIONAL DE ESTUDIOS PROFESIONALES
ARAGÓN
DIRECCION

JAVIER FELIPE REYES DAMIAN
P R E S E N T E.

En contestación a su solicitud de fecha 7 de mayo del año en curso, relativa a la autorización que se le debe conceder para que el señor profesor, Ing. ERNESTO PEÑALOZA ROMERO pueda dirigirle el trabajo de Tesis denominado "INSTRUMENTACIÓN DE UN PRODUCTO DE SOFTWARE PARA EL CONTROL DE UN EQUIPO DE DESARROLLO 'X-Y' ", con fundamento en el punto 6 y siguientes, del Reglamento para Exámenes Profesionales en esta Escuela, y toda vez que la documentación presentada por usted reúne los requisitos que establece el precitado Reglamento; me permito comunicarle que ha sido aprobada su solicitud.

Aprovecho la ocasión para reiterarle mi distinguida consideración.

ATENTAMENTE
"POR MI RAZA HABLARA EL ESPÍRITU"
San Juan de Aragón, México, 10 de mayo de 1996.
EL DIRECTOR

M en I CLAUDIO C. MERRIFIELD CASTRO

c c p Jefe de la Unidad Académica.
c c p Jefatura de Carrera de Ingeniería en Computación.
c c p Asesor de Tesis.

CCMC'AIR'lla.

VNIVERSIDAD NACIONAL
AVPNMA DE
MEXICO

UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO
CAMPUS ARAGÓN

UNIDAD ACADEMICA

Ing. JUAN GASTALDI PÉREZ
Jefe de la Carrera de Ingeniería en Computación,
P r e s e n t e

En atención a la solicitud de fecha 6 de mayo del año en curso, por la que se comunica que el alumno JAVIER FELIPE REYES DAMIAN, de la carrera de Ingeniero en Computación, ha concluido su trabajo de investigación intitulado "INSTRUMENTACIÓN DE UN PRODUCTO DE SOFTWARE PARA EL CONTROL DE UN EQUIPO DE DESARROLLO 'X-Y'", y como el mismo ha sido revisado y aprobado por usted, se autoriza su impresión; así como la iniciación de los trámites correspondientes para la celebración del Examen Profesional.

Sin otro particular, le reitero las seguridades de mi atenta consideración.

ATENTAMENTE
"POR MI RAZA HABLARA EL ESPÍRITU"
San Juan de Aragón, México, 7 de mayo de 1997
EL JEFE DE LA UNIDAD

LIC. ALBERTO IBARRA ROSAS

c c p Asesor de Tesis.
c c p Interesado.

AIR'lla.

AGRADECIMIENTOS

Quiero hacer un agradecimiento muy especial al Ing. Gerardo Castro Zavala, asesor de está tesis por parte del CINVESTAV, por todo su apoyo, confianza y paciencia que me brindo durante mi estancia en este centro.

También quiero agradecer al departamento de Ingeniería Eléctrica del Centro de Investigaciones y Estudios Avanzados del I.P.N. (CINVESTAV), por el apoyo y facilidades que ofrece a los egresados de las diferentes instituciones para la realización de su tesis.

Agradezco la valiosa ayuda del Ing. Ernesto Peñaloza Romero, asesor de está tesis por parte de la Escuela Nacional de Estudios Profesionales Aragón, por que todas sus intervenciones fueron de gran utilidad para el desarrollo y conclusión de está tesis.

Por último, seria difícil realizar un reconocimiento de manera individual a todas aquellas personas, familiares, maestros e instituciones que contribuyeron de manera muy significativa en mi formación como persona y profesionista y con los cuales me encuentro en deuda. Por todo ello: *Muchas gracias*

X.F.R.D.

Dedico está tesis a:

Dios

Por mantener en mí, siempre viva la llama de la esperanza, que hizo de este sueño, una realidad.

Javier, mi padre.

Por todos los esfuerzos y sacrificios que tuvo que hacer para dejarme está herencia.

¡Que Dios te bendiga Padre!

Angela, mi madre.

Por que supisteis guiar mis palabras y mis pasos y me enseñaste lo que es ser un hombre.

¡Que Dios te bendiga Madre!

Angélica, mi hermana.

Por que siempre te llevare en mi corazón.

Roberto.

Por ser más que un amigo, un verdadero hermano.

A mi alma mater.

¡Por darme la libertad, al quitarme las cadenas de la ignorancia.!

México, mi patria.

¡La razón de mi superación.!

ÍNDICE

Contenido Pág.

ÍNDICE

Introducción

Está tesis se llevo acabo con la intención de alcanzar el siguiente objetivo:

Diseñar un sistema en el lenguaje de programación Turbo C que permita al usuario realizar perforaciones de circuitos impresos mediante la identificación de las coordenadas de los punto contenidos en los archivos fuente de los paquetes TANGO, SMARTWORK Y ORCAD. Así como también, que se tenga la posibilidad de maquinar una pieza mediante la edición de un programa o en forma guiada .

La sección de Control Automático del Centro de Investigaciones y Estudios Avanzados (CINVESTAV) en Zacatenco y en especial en el laboratorio de robótica del departamento de Ingeniería Eléctrica, no cuenta con un sistema automático de perforado de piezas y tarjetas de circuitos impresos.

Normalmente, está actividad se viene relizando en forma manual, y corresponde desde luego al diseñador del prototipo llevar a cabo la perforación de su tarjeta. Está tarea resulta ser entonces, tediosa e improductiva, es decir, este tiempo bien se podría emplear en la planeación de algún otro aspecto del diseño. Esto se agudiza aún más si se trata de realizar una gran cantidad de perforaciones. Y a pesar de que se tenga mucho cuidado al perforar, el individuo no es tan preciso en su movimiento, y mucho menos si se le compara con un sistema dedicado especialmente a esto.

Una vez detectada está necesidad, sólo existen dos alternativas posibles para dar solución definitiva a está situación: la primera consiste en la adquisición de un equipo existente en el mercado, y la segunda, en la implantación del sistema de perforado a partir de herramienta y equipo con que cuenta la sección. Ambas alternativas presentan sus ventajas y desventajas.

En cuanto a la compra de un equipo se garantiza una gran capacidad de producción de tarjetas perforadas, en un corto periodo de tiempo y con una alta precisión en el trabajo. Además este tipo de equipos viene acompañados en su mayoría con un paquete de software que se utiliza para el diseño de circuitos impresos por computadora, de está manera el sistema se hace más versátil y completo. Pero esto significaría un desembolso muy grande como se puede constatar, en el equipo de perforado Marca: KEPRO, Modelo: CNC128RCC6, el cual se encuentra en el mercado y tiene un costo aproximado de $29,975. Además cuenta con todas las características descritas anteriormente.

En cuanto al desarrollo de un sistema como el anterior resulta difícil, pero no imposible reunir los recursos necesarios y adquirir exclusivamente lo indispensable para lograrlo. Desde luego, no se pretende tener la misma capacidad de perforado y rapidez. Pero se busca ante todo, buena precisión, en un tiempo mucho menor al que se emplearía en forma manual.

El CINVESTAV cuenta ya con una mesa y un taladro industrial, a está estructura se le integró un par de motores para dar el desplazamiento tanto en el eje X como en el eje Y. Así también se le colocaron una serie de sensores los cuales indicarán en un momento dado si han llegado al inicio o al fin de la mesa. También cuenta con un sistema desarrollado en este centro de investigación y que se encarga de recibir la información enviada por la PC para efectuar alguna tarea de movimiento o de perforado. Este sistema se baso en el microcontrolador 8031 de la familia Intel.

Tomando en cuenta que la parte de la estructura mecánica y electrónica ya se encuentra solucionada, lo único que faltaría para contar con un equipo de perforado sería desarrollar un software que se cargue y ejecute en la computadora y otro que permanezca resiente en el microcontrolador y se ejecute al momento de encenderlo.

Con respecto al software para la computadora, este se debe desarrollar en el lenguaje de programación **C** y debe estar dividida en dos modalidades principales:

La primera modalidad del sistema se le denominara **Perforado de Tarjetas,** la cual a partir del desarrollo de un circuito impreso en cualquiera de los siguientes paquetes ORCAD, SMARTWORK y TANGO, se podrán realizar las perforaciones contenidas en dicho circuito impreso, a partir del archivo que se genere. La segunda modalidad del sistema a la cual se le denominara **Maquinado de Piezas**, está modalidad se dividido en tres modos de ejecución:

El primer modo denominado **Automático**, tiene por objetivo generar y ejecutar un programa para que se puedan efectuar una serie de desplazamientos y perforaciones, pudiéndose realizar un maquinado de alguna pieza en especial. El segundo modo denominado **Interactivo**, tiene por objeto, que a partir de un programa ya creado se puedan ejecutar una a una las instrucciones contenidas en el mismo sin importar el ordenen en el cual se encuentren. El tercer modo denominado **Manual,** debe realizar algún movimiento o perforación en forma manual o guiada por medio de la elección de alguna tecla que especifique dicha tarea.

Con respecto al software para el microcontrolador, este se debe diseñar en le lenguaje máquina del microcontrolador 8051 y su estructura dependerá de los resultados que se obtenga del software anterior.

Está etapa de programación se llevará a cabo mediante dos proyectos de tesis. Donde, el programa que corresponde a la primera etapa se describe a continuación, quedando por cubrir el desarrollo del programa para el funcionamiento del microcontrolador. Para llevar acabo el desarrollo de está tesis se planteo la siguiente hipótesis: En la modalidad **Perforado de Tarjetas**, se tendrá que identificar los puntos mediante la apertura del archivo fuente y filtrar de este solo los datos que correspondan a las coordenadas de cada punto, para después efectuar el ordenamiento en zig zag de estos mediante el empleo de alguna herramienta de ordenamiento básica. En la modalidad **Maquinado de Piezas**, el editor se construirá mediante un Array de punteros, en donde se almacenaran las líneas del código del programa. La revisión de sintaxis, se debe realizar línea por línea para localizar como primer punto, los caracteres que no sean permitidos en el programa, como segundo punto se debe contar con una tabla en donde se encuentren todas las sentencias y se debe realizar una comprobación del código contenido en cada línea del editor contra el contenido de la tabla para identificar las sentencias mal escritas, para finalizar se debe verificar él contenido del argumento con respecto a un valor máximo y mínimo permitido para cada sentencia.

El capítulo uno, comprende la descripción general de los elementos mecánicos, eléctricos y electrónicos con los que se cuenta para llevar acabo el proyecto y de está manera tener una idea más clara del software que se requiere diseñar.

El capítulo dos, realiza el planteamiento paso a paso del ciclo de desarrollo del sistema, mediante el análisis y diseño de los puntos fundamentales y los objetivos que se desean alcanzar en los cuatro modos de ejecución.

El capítulo tres, explica y describe la forma de acceder a las órdenes contenidas en los menús del sistema diseñado en el capítulo anterior para la ejecución de cualquiera de las dos modalidades de operación básicas. El capítulo, trata el uso del editor para la edición de programas encaminados a la realización de una tarea de perforado, así como también contiene una descripción formal de las sentencias, la estructura de control y modo de empleo de las subrutinas.

El capítulo cuarto, es describe el programa de prueba a nivel ensamblador que se tuvo que desarrollar para verificar cada una de las partes del sistema anterior y comprobar su buen funcionamiento.

El capítulo cinco, es la continuación del capítulo cuatro, ya que aquí se prueba el funcionamiento del sistema poniéndolo en marcha junto con un sistema de desarrollo que cuenta con los elementos necesarios para poder entablar la comunicación entre ellos, este sistema contiene el programa a nivel ensamblador descrito anteriormente y así examinar los diferentes modos de ejecución disponibles en las dos modalidades.

CAPÍTULO

I

DESCRIPCIÓN DEL EQUIPO X-Y

Una máquina puede hace el trabajo de cien hombres normales, pero ninguna máquina puede hacer el trabajo de un hombre extraordinario

Elbert Habbaard

I Equipo X-Y

El equipo se compone por un taladro industrial que puede operar en forma manual o en forma automática, la mesa X-Y es una estructura que tiene en sus extremos un motor a pasos respectivamente, los cuales desplazan una plataforma en donde se colocan las tarjetas o piezas a perforar, también cuenta con dos módulos, en uno de los cuales se encuentran las fuentes de alimentación o etapa de potencia que es la que se encarga de suministrar el voltaje necesario a los motores y la otra es la de control en donde se encuentra el microcontrolador.

I.1 Electrónica del equipo X-Y

La electrónica del equipo X-Y, está compuesta en su parte fundamental por la etapa de control en el cual se encuentra como elemento principal de este sistema el microcontrolador 8031, éste se auxilia para el buen funcionamiento de la etapa de potencia, la etapa de los sensores y la comunicación serial, la etapa de potencia fue diseñada para proporcionarle al microcontrolador y a los motores el voltaje requerido para su funcionamiento, la de los sensores fue diseñada para informar al microcontrolador en que momento se llega al inicio o al fin del recorrido, así como también la posición que guarda el taladro ya sea arriba o abajo, la comunicación serial se estableció para que el microcontrolador pueda transmitir y recibir información de la computadora ya sea por la vía RS-232 o por RS-485.

I.1.1 Microcontrolador 8031

El sistema mínimo que gobierna el equipo X-Y está compuesto básicamente por un microcontrolador (μC), lo cual ofrece un enorme panorama hacia el mundo de la compatibilidad. Este dispositivo contiene: Una CPU (basado principalmente en un microprocesador de 8 bits) , memoria RAM, puertos paralelos de entrada y salida, puerto serie, timers, contadores y memoria EPROM .

Un microprocesador está encaminado básicamente hacia aplicaciones concretas en donde, el espacio, y número de componentes es mínimo, además, los cambios y ampliaciones son casi nulos. Por otro lado un microprocesador se destina a sistemas donde su expansión a corto o mediano plazo es factible. A pesar de que un microprocesador es más rápido que un microcontrolador para la ejecución de sus instrucciones en la mayoría de los casos es necesario interconectarlo con dispositivos periféricos.

El término genérico 8051 es usado para referir colectivamente a los microcontroladores 8031, 8051, 8751 de la familia de Intel. Las diferencias entre estos chips es la siguiente:

- El µC 8051, cuenta con una ROM interna, la cual es programada directamente por el fabricante.
- El µC 8751, cuenta con una EPROM interna, la cual puede ser programada por el usuario.
- El µC 8031, no cuenta con ningún tipo de memoria interna

La familia de µC 8051 es variada, y se encuentra en diversas presentaciones, la selección de uno o de otro tipo de microcontrolador dependerá principalmente de las necesidades a satisfacer.

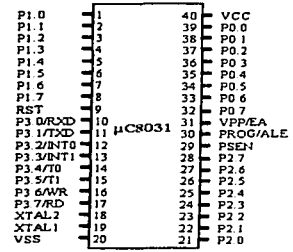


Figura 1-1. Microcontrolador 8031

El elemento más básico de la familia µC 8051 es el µC 8031 que carece de EPROM o PROM, el cual es direccionado externamente. El µC8031 es fundamentalmente un chip de 40 líneas, como lo muestra la Figura 1-1; se basa en los microprocesadores de 8 bits, contiene internamente una CPU de 8 bits, 3 puertos de entrada y salida en paralelo, un puerto de control el cual a su vez contiene: un puerto serie, 2 entradas para interrupciones externas, las señales de RD y WR para la toma o almacenamiento de datos externos en RAM, la señal de PSEN para lectura de instrucciones almacenadas en la EPROM. La memoria del µC 8031 se clasifica en tres partes fundamentales: La primera, llamada memoria de programa, en donde se encuentran todas las instrucciones que van a ser ejecutadas por el 8031. El segundo espacio de memoria denominado memoria de datos, es accesado mediante la activación de las señales RD y WR, durante la lectura o escritura de datos respectivamente. El tercer espació en memoria es la memoria RAM interna, el cual se subdivide en 128 bytes de memoria bajos y 128 bytes de memoria altos.

En los 128 bytes bajos se encuentran 4 bancos de 7 registros cada uno. Estos registros son de gran ayuda para la simplificación de los programas, debido a que cada uno de ellos nos permiten almacenar datos momentáneamente dentro de este espacio, y se encuentran 16 bytes que pueden ser direccionados directamente por bit.

En la parte alta de la memoria RAM interna, se encuentran el contenido de los registros de funciones especiales; Puertos, Registros de Control, Acumuladores, Registros de interrupción, etc. La Tabla 1-1, presenta la descripción de cada una de las líneas microcontrolador 8031.

SÍMBOLO	No. DEL PIN	TIPO	FUNCIÓN
P0.0 - P0.7	32 - 39	Ent./Sal.	**Puerto 0.** Es un puerto bidireccional con salidas en colector abierto.
P1.0 - P1.7	1 - 8	Ent./Sal.	**Puerto 1.** Es un puerto quasi-bidireccional, cuando se escribe 1's en el puerto, el puerto puede ser utilizado como entrada.
P2.0 - P2.7	21 - 28	Ent./Sal.	**Puerto 2.** Es un puerto utilizado para direccionar memoria externa.
P3.0 - P3.7	10 - 17	Ent./Sal.	**Puerto 3.** Es utilizado para producir señales de control a dispositivos externos como son :
P3.0	10	Ent.	**RxD** Puerto serie de entrada
P3.1	11	Sal.	**TxD** Puerto serie de salida
P3.2	12	Ent.	**INTR0** Interrupción externa
P3.3	13	Ent.	**INTR1** Interrupción externa
P3.4	14	Ent.	**T0** Entrada externa timer 0
P3.5	15	Ent.	**T1** Entrada externa timer 1
P3.6	16	Sal.	**WR** Habilitador de escritura para memoria externa
P3.7	17	Sal.	**RD** de datos. Habilitador de lectura para memoria externa de datos.
PSEN	29	Sal.	**Program Store Enable.** Habilitador de lectura para memoria de programas externos.
VSS	20	Ent.	**Tierra:** 0 volts de referencia.
RST	9	Ent.	**Reset:** Una lectura alta detiene el dispositivo.
ALE	30	Ent./Sal.	**Address Latch Enable.** Un pulso positivo de salida permite fijar el byte bajo de la dirección durante el acceso a una memoria externa. En operación normal.
XTAL1	19	Ent.	**Crystal 1.** Es la entrada del cristal para el circuito oscilador que amplifica e invierte la entrada.
XTAL2	18	O	**Crystal 2.** Es la salida del amplificador oscilador inversor.
EA	31	Ent.	**External Access Enable.** En posición baja habilita el almacenamiento que elige el código de las localizaciones de la memoria de programas externos. En posición alta, el dispositivo ejecuta los programas que se encuentran en la memoria interna ROM.

Tabla 1-1. Líneas del microcontrolador 8031..

I.1.2 Etapa de potencia

La etapa de potencia comprende tres fuentes de alimentación :

A) Fuente de alimentación para los motores a pasos.
B) Fuente de alimentación para la electrónica de potencia.
C) Fuente de alimentación para la etapa de control.

Estas fuentes son llamadas Vcc1 Vcc2 y Vcc3 respectivamente y se encuentran integradas en una sola unidad llamada etapa de potencia, la cual cuenta con los conectores necesarios para interconectarse con la etapa de control y los motores a pasos. Las fuentes Vcc1 y Vcc2 se encuentran aisladas ópticamente de Vcc3 esto para proteger la electrónica de control de cualquier eventualidad. Las fuentes Vcc1 y Vcc2 se encuentran en un mismo circuito impreso, requiriéndose dos de estos para cada motor. Esto debido a la limitada capacidad de los reguladores de voltaje.

Los motores a pasos empleados para el desplazamiento de la plataforma de mesa X-Y son de potencia, para ello la fuente Vcc1 debe proporcionar la corriente requerida por estos motores en movimiento y más aun debe ser capaz de soportar la corriente exigida cuando un motor se detiene ya que es el momento en que se presenta un consumo de corriente durante un mayor tiempo. Mientras el motor se encuentra en movimiento la corriente a través de la bobina circula por un breve instante ya que las bobinas están conectadas una a una. Cuando se detiene el movimiento una bobina se queda conectada circulando corriente por más tiempo. La fuente de potencia para los motores a pasos proporciona una corriente de 5 ampers sobre un voltaje de 1.2v a 32v. El voltaje de salida de este regulador se ajusta mediante un potenciómetro para obtener el voltaje requerido por cada motor a pasos.

Esta fuente cuenta con un sistema que detecta si se presenta un corto circuito. El sensor de corto circuito se forma por una resistencia sensora y un comparador de voltaje LM311. La resistencia es de un valor muy pequeño (0.22Ω) de forma tal que no exista una diferencia de potencial muy significativa entre sus terminales. Está diferencia sólo se presentará cuando exista un flujo de corriente muy alto la cual sería posible cuando existiese un corto circuito. Está resistencia también tiene la característica de ser de potencia debido a la corriente que tiene que soportar.

Los nodos de entrada y salida de la resistencia sensora se conectan con un comparador de voltaje, este se ajusta para que en el momento en que exista una diferencia de voltaje establecida como corto circuito, este produzca una señal cual será enviada al microcontrolador y al indicador visual.

La fuente Vcc2 se le considera como la fuente de alimentación para la electrónica de potencia, está fuente se encarga de proporcionar el voltaje necesario para alimentar al sistema de encendido de la fuente Vcc1 y también para generar la señal de corto. El sistema de encendido es activado por medio del microcontrolador, y está formado por un relevador de estado sólido, mediante un arreglo triac TIC226 y triac driver MOC3010. Los voltajes para la electrónica de control son proporcionados por la fuente denominada Vcc3 la cual es una fuente regulada de 5 volts que alimenta al módulo de control, y se encuentra aislada ópticamente de Vcc1 y Vcc2, debido a que el microcontrolador debe estar siempre energizado ya que es el encargado de enviar la señal para encender la fuente de alimentación Vcc1, esto también previene algún desperfecto en el módulo de control si existiese algún corto en cualquiera de las otras dos fuentes ya sea en Vcc1 o Vcc2. Está etapa de potencia en su conjunto cuenta con un adecuado sistema de ventilación y disipación de calor. El sistema de ventilación está conformado por un orificio acoplado con un ventilador extractor de aire, el cual empieza a funcionar al alimentar está la etapa de potencia; el sistema de disipación de calor está conformado por un disipador de calor donde son acoplados lo reguladores de voltaje, este disipador se encuentra alineado con el ventilador para obtener una mejor disipación de calor.

I.1.3 Etapa de Sensores

La etapa de sensores se compone principalmente de optoswitchs tipo ranura como se muestra en la Figura 1-2, distribuidos adecuadamente en la mesa X-Y de forma tal que se pudiese determinar la posición de inicio y fin (HOME-END) del la mesa, así como de la posición arriba y abajo (UP-DOWN) del taladro, los optoswitchs de la mesa y el taladro se encuentran interconectados a la tarjeta de control, en dicha tarjeta se encuentra la electrónica necesaria para su funcionamiento.

Figura 1-2. Sensor tipo ranura.

De está manera se puede llevar acabo el sensado de "home" y "end" tanto en la dirección X como en Y. Su función no es autónoma, sino que debe estar estrechamente en comunicación con el módulo motriz para que le indique hasta que punto se debe mover el motor la plataforma en X o Y, de manera que pueda identificar en que momento se alcanza el origen o el fin de la mesa X-Y. Físicamente esto se logra colocando los sensores en los extremos de la mesa, y es precisamente en el momento en el que se interrumpe el haz cuando el microcontrolador detecta que se ha llegado al inicio o al fin del recorrido, así también el taladro tiene dos sensores que le indican al microcontrolador cual es su posición.

I.1.4 Comunicación Serial.

El puerto serie constituye una vía mediante la cual el procesador se conecta al mundo exterior. A través del puerto serie el procesador puede recibir y enviar señales a un dispositivo estableciendo de está forma una comunicación.

El sistema de comunicación serial se encuentra en el módulo de control y se utiliza para transmitir y recibir información de la computadora al microcontrolador o viceversa. El módulo de control cuenta con las dos normas de comunicación serie, el **RS-232** y **RS-485**. Se pueden emplear cualquiera de estos dos tipos, dependiendo de la distancia en que se encuentren el microcontrolador de la computadora, si la distancia es menor de los 12 mts se puede utilizara la comunicación **RS-232**, pero en el caso contrario de que la distancia sea mayor se recomienda la comunicación **RS-485**, está interfaz proporciona inmunidad al ruido en ambientes industriales.

Cuando se quiera transmitir información vía **RS-485**, será necesario contar con una tarjeta convertidora de norma **RS-232** a **RS-485**, ya que la computadora no cuenta con un puerto serial **RS-485**.

La habilitación de uno u otro sistema está disponible en la tarjeta del sistema mínimo la cual se encuentra en el módulo de control.

I.1.5 Etapa de control.

Está etapa, se compone por un sistema de desarrollo, que contiene los siguientes elementos:

1 microcontrolador 8031.
1 memoria EPROM 2764 de 8Kb.
1 memoria RAM 6264 de 8Kb.
1 compuerta (LATCH) 74LS373.
1 compuerta (NOR) 74LS02.
1 TRANSEIVER 75176.
1 puerto paralelo PPI 8255.
1 flip-flop 74123.
1 circuito MAX232.
1 multiplexor/demultiplexor 74LS138.

El microcontrolador está compuesto básicamente por una CPU 8051, una memoria de entrada/salida, 32 líneas de entrada y salida distribuidas en 4 puertos de 8 bits, 2 TIMER/CONTADORES de 16 bits, un puerto serie de entrada/salida, un oscilador y circuitos de reloj.

Como área de memoria de programa se utiliza una EPROM 2764 de 8Kbytes, aquí se almacena el programa monitor residente que se encarga de inicializar el controlador, almacena y ejecuta los programas de aplicación.

Como área de memora de datos se utiliza la memoria RAM 6264 de 8 Kbytes, se necesita demultiplexar la parte baja del bus de datos dentro del puerto 0, para esto se utiliza una compuerta LATCH 74LS373 la cual puede atrapar 8 líneas de entrada siendo estas líneas los 8 bits de la parte baja del canal de direcciones.

Para ejecutar programas de aplicación almacenados dentro del área de memoria de externa de datos RAM se hace creer al microprocesador 8031 que está accesando áreas de memoria de programa pero realmente está seleccionando área de datos. Lo anterior se realiza colocando un pequeño circuito de selección utilizando la compuerta (NOR) 74LS02 que habilita la memoria RAM cuando se seleccionan localidades superiores a los 8 Kbyte's, es decir, para accesar a la memoria de datos se usan las direcciones a partir de la 0000H y como memoria de programa se usan las direcciones a partir de la 2000H.

El circuito 75176 se utiliza para realizar la comunicación serie dentro de la norma eléctrica RS-485 recomendada por BUS INTEL. Para efectuar la transmisión vía RS-32 se utiliza el dispositivo MAX-232 el cual nos permite convertir señales RS-232 a señales TTL/CMOS y viceversa.

Se cuenta con un sistema de monitoreo el cual se encarga de vigilar el buen funcionamientos del microcontrolador previniendo de está forma cualquier eventualidad, este sistema está implementado mediante un circuito integrado 74123 el cual es un temporizador (Multivibrador Monoestable Redisparable).

La entrada del 74123 está conectada al microcontrolador, mientras que la salida se encuentra conectada al sistema de inicializar (Reset), mediante un programa el microcontrolador genera un pulso el cual es monitoreado por el 74123, si el microcontrolador falla de alguna forma, la señal se deja de producir, en este momento el 74123 manda una señal al microcontrolador deteniendo cualquier operación del sistema.

El 8255 tiene tres puertos A, B y C de 8 bits cada uno. Todos pueden ser configurados en una amplia variedad de funciones, pero cada uno de estos puertos tiene sus propias características las cuales dan una gran flexibilidad a este dispositivo.

Los puertos A, B y C pueden ser utilizados ya sea como entradas o salidas. El puerto C tiene la característica de que puede ser dividido en dos puertos de 4 bits, el cual puede ser usado para salida y entrada de señales.

Otro elemento fundamental que se encuentra en está etapa es la tarjeta manejadora de motores a pasos o tarjeta **Driver de Motores a Pasos**, en está tarjeta llegan los pulsos generados por el microcontrolador donde son procesados para proporcionar las señales necesarias para el movimiento de los motores. A está tarjeta llegan tres señales provenientes del microcontrolador las cuales son: pulsos, giros y fase. La señal de pulso es un tren de pulsos que nos dará el número de pasos a moverse y la señal de giro puede ser un 1 o un 0, y la señal de fase es para incrementar la potencia del motor.

I.2 Funcionamiento del taladro y motores

El taladro y los motores son dos elementos adquiridos en base a los requerimientos y necesidades del equipo X-Y. Los motores se utilizan para realizar los desplazamientos en cualquiera de los ejes X-Y. Mientras que el taladro se encargara de realizar las perforaciones

I.2.1 Características de los motores.

Se utilizan dos motores a pasos para efectuar los desplazamientos en los dos distintos ejes de la mesa X-Y. El motor a pasos es un motor eléctrico que está formado por un conjunto de bobinas independientes cuyo eje gira una cantidad especifica por cada pulso de entrada que recibe, lo cual permite el control de posición, velocidad, y sentido.

Se le denomina paso al ángulo mínimo que define la distancia entre dos posiciones angulares consecutivas. En este tipo de motores, la ventaja más importante es la correspondiente entre la señal de control y la posición alcanzada sin necesidad de cerrar el ciclo realimentando la posición. Aunque el conjunto de posiciones alcanzables es limitada ya que sólo se puede alcanzar posiciones discretas, definidas por el número de pasos por vuelta. Este tipo de motores se alimentan de corriente directa y son manejados con circuiteria lógica. El tipo de motor utilizado en la mesa X-Y es de tipo de imán permanente de cuatro fases dando 200 pasos por revolución. Especificaciones de los motores a pasos:

Marca: FUJI ELECTRIC CO.LTD
Modelo: GPF-2845-2A
Voltaje DC: 1.8Volts
Corriente DC por fase: 4.8 A
Desplazamiento angular: 1.8°

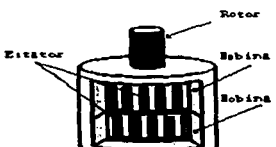


Figura 1.3 motor a pasos

I.2.2 Característica del taladro

Para realizar las perforaciones se utiliza un taladro industrial neumático el cual tiene como característica principal funcionar en forma manual y en forma automática. Para esto se cuenta con un interruptor que determinara el modo de operación, como se muestra en la Figura 1-4.

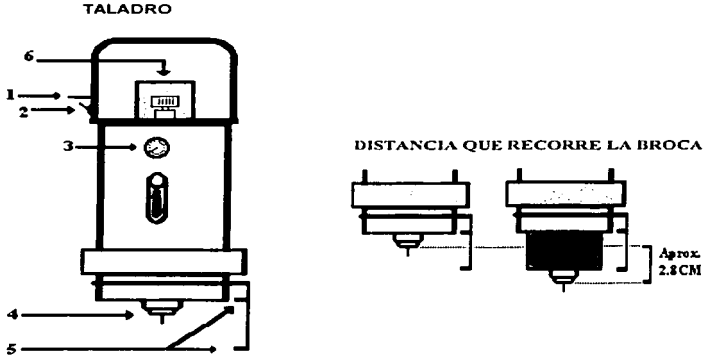


Figura 3-1. Taladro.

No.	DESCRIPCION
1	NTERRUPTOR ON / OFF
2	INTERRUPTOR DE SELECCIÓN AUTOMÁTICO / MANUAL
3	MANÓMETRO
4	MANDRIL
5	SENSORES
6	REGULADOR DE PRESIÓN

En forma manual se puede activar el taladro por medio de un interruptor tipo pushbutton, en el cual efectúa la operación de bajar y subir la broca. En forma automática, el taladro se activa mediante una señal de entrada, la cual energiza un relevador que presiona al pushbutton. Con el motor funcionando se puede regular la presión que ejerce el taladro sobre la pieza.

Entre los diferentes tipos de operaciones que se pueden realizar con el taladro se encuentran:

a) **Operación de corte** : Por medio del corte de agujeros redondos en el metal.

b) **Operación con los escariadores**: Son herramientas cortantes de precisión que se emplean para agrandar agujeros existentes con acabados fino y presión, quitando una pequeña cantidad de material.

c) **Operación de avellanado**: Es la operación de maquinar una abertura o receso de forma cónica en el extremo exterior de un agujero.

d) **Operación de refrentado**: Está elimina una pequeña cantidad de material alrededor de la superficie

Los materiales que pueden ser maquinables en este taladro son: Cobre, Níquel, Zinc, Aluminio, Acero suave, Bronce, PVC, Maderas, Tabletas fenólicas y Nylanid. Así como los diferentes tipos de Niylamid entre los que se encuentra: Nylamid N, Nylamid SL, Nylamid TS.

Especificaciones del taladro

Marca: DUMORE
Modelo: 20031
No. de rpm: 7500
Peso: 12.5 kg
Voltaje CA: 115volts-120volts
Corriente CA: 2.5 Amp
Diámetro mínimo de perforación: 3/32"
Diámetro máximo de perforación: 5/32"

I.3 Características generales del equipo X-Y.

Los datos que se proporcionan a continuación fueron obtenidos mediante el desarrollo de cálculos y programas, así como también por medio de la medición de algunas partes de la mesa X-Y, únicamente el No. de Pasos/Rev fue proporcionado por el fabricante.

Estos datos nos son de gran utilidad en los desplazamientos ya que de está menera se puede saber el número de pasos que se deben realizar para recorrer una cierta distancia.

Los datos que se muestran en la Tabla 1-2, son en base a las características originales del equipo, aunque estas pueden variar en el momento en que se cambien los motores por otros de mayor resolución.

RESOLUCIÓN DE LOS MOTORES A PASOS			
Eje	No. Pulsos/Rev	No. Pulsos Home/End	No. Vueltas Home/End
X	200	17222	86.11
Y	200	8335	41.6

TIEMPO Y VELOCIDAD DE RECORRIDO HOME/END			
	Despl. (mm)	Tiempo (seg)	Vel. Lin (mm/seg)
EJE X	435	100	4.85
EJE Y	215	105	2.05
TALADRO	29	2	2.90

VELOCIDAD ANGULAR DE LOS MOTORES A PASOS		
Eje	Resolución(Pasos/mm)	Vel. Angular (rev/s)
X	39.59	0.8611
Y	38.76	0.0416

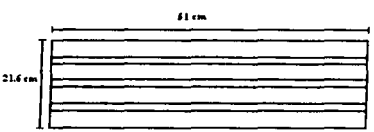
TABLA 1-2. Características generales del equipo

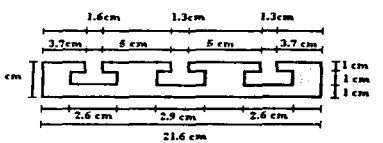
I.3.1 Características físicas de la mesa X-Y.

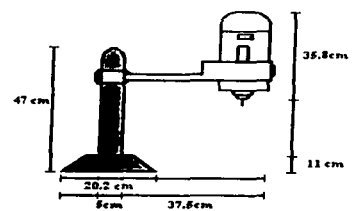
La mesa X-Y es una estructura que pesa aproximadamente 90 Kgs y que se construyó en el Centro de Investigaciones y Estudios Avanzados. El material que se utilizo para su construcción fue Aluminio y Acero, como lo muestra la Figura 1-5.

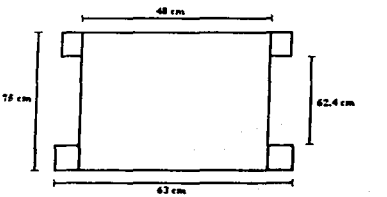
Cuenta con una base sobre la que descansan dos pares de rieles con un tornillo sin fin cada uno, de los ejes para mover una plataforma rectangular (base para colocar tarjetas o piezas) de 51 cm de largo en X y 21.6cm de ancho en Y. En estos tornillos van acoplados unos motores de 200 pasos por revolución de 4 fases cada uno, para efectuar el desplazamiento.

Además, en la base de mesa X-Y está montado sobre un mástil fijo un taladro tipo industrial semiautomático y cuya posición con respecto a la altura se puede variar.









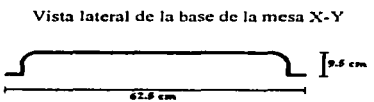


Figura 1-5. Estructura de la mesa X-Y

I.3.2 Panel de control

La unidad o etapa de potencia por la parte frontal está compuesta por un tablero el cual contiene indicadores de corto para cada fuente, también cuenta con un indicador de voltaje de línea así como un fusible tipo chasis para un fácil acceso y un interruptor de línea con que se energiza el módulo, como lo muestra la Figura 1-6.

En la parte posterior cuenta con los conectores tipo N-LOCK que alimentan a los motores a pasos. Se tiene el conector de información relativa a este módulo (DB9A) el cual es un conector tipo DB9. Se cuenta también con el conector standard para la alimentación de 120 vca.

En está parte se encuentra un ventilador extractor de aire y su función es la de proporcionar a los elementos una ventilación adecuada.

En la unidad control, por la parte frontal se compone por un tablero con tres conectores, dos de ellos son de tipo DB9 para la comunicación serie RS-232 y RS-485.

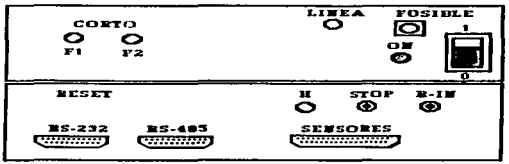


Figura 1-6. Módulos de potencia y control.

El otro conector es un DB25 y es utilizado para los sensores, también cuenta con tres pushbutton, los cuales tienen la tarea de parar el proceso (Pausa), reiniciar el proceso (Continuar) e inicializar el microcontrolador (Reset), en los procesos de pausa y continuar se encienden dos leds que indican la tarea que se está efectuando, como se muestra en la Figura 1-6. En la parte posterior se encuentran tres conectores N-LOCK, el primero de nueve pin, que se utiliza para la alimentación de la unidad de control, el segundo N-LOCK de 12 pin, se encuentran conectados los dos motores a pasos que realizan el desplazamiento de la mesa X-Y y por último el tercer N-LOCK de tres pin, que se utilizan para conectar el actuador del motor.

I.3.3 Requerimientos de Software y Hardware

Para el desarrollo y ejecución de este sistema se debe contar con los siguientes componentes:

COMPONENTE	REQUERIMIENTO
COMPUTADORA	Procesador 386 a 20Mhz mínimo 640Mb de memoria RAM. Monitor VGA,SVGA oUVGA. 1 Drivers 3½ ,5¼ o ambos. Mouse. Disco duro. Teclado en español 101 teclas. Un puerto serie. Un puerto paralelo.
SISTEMA OPERATIVO	Versión 5.0 o superior
PAQUETES	Tango Smartwork Orcad Lenguaje de programación Turbo C

CAPÍTULO

II

ANÁLISIS Y DISEÑO

¡Un extraordinario laberinto!
pero no sin solución

Alexander Pope

II Secuencia de desarrollo

El sistema que se requiere diseñar debe estar dividido en dos modalidades: **Perforado de Tarjetas** y **Maquinado de Piezas**. En la modalidad de **Perforado de Tarjetas** el sistema debe tener la capacidad de filtrar de un archivo de texto, las coordenadas de los puntos que se requieran perforar, independientemente del paquete en el cual haya sido diseñado el circuito impreso: ORCAD, SMARTWORK o TANGO. Estos puntos deben ser ordenados en forma que no se pierda demasiado tiempo en el desplazamiento entre punto y punto. Así como también se debe identificar que la tarjeta cumpla con las condiciones de diseño para poder ser reconocida por este sistema.

A partir de los puntos que se filtraron se debe dibujar la tarjeta en la pantalla y en su momento se debe de poder ampliar dicha tarjeta para identificar cual es la posición que debe guardar en la plataforma de la mesa.

Para ejecutar el modo de **Perforado de Tarjetas** se debe contar con las opciones de **Inicializar**, **Parar** el proceso, **Continuar** y **Salir** del modo de ejecución, así como también debe indicar que puntos han sido perforados y cuantos faltan.

En la modalidad de **Maquinado de Piezas** el sistema debe contar con un editor de texto en el cual se crearán programas, los cuales deben ser compilados y ejecutados en forma **Automática** e **Interactiva**.

El compilador debe revisar la sintaxis del programa y cuando este no encuentre errores debe crear un archivo objeto, en el cual se almacenarán las instrucciones a ejecutar en forma hexadecimal.

La ejecución en forma **Automática**, se llevará acabo por medio de la apertura de un archivo generado por el compilador con la extensión .HEX, el cual transmitirá los datos o bytes al microcontrolador y este ejecutará cada una de las instrucciones que le van llegando. La ejecución en forma **Interactiva**, el usuario podrá elegir mediante el teclado cualquiera de las sentencias que se encuentren en el programa y ejecutarlas. También se debe almacenar y cargar el archivo que contiene un programa en el momento que se requiera.

La ejecución en forma **Manual**, las operaciones de desplazamiento y perforación se realizaran por medio del teclado, es decir, cada movimiento, perforación, o cambio de datos se realizará a través de la pulsación conjunta de dos tecleas.

En las tres modalidades el programa debe detectar el momento en que se realice alguna operación de pausa o continuar, esto por medio de la pulsación de los interruptores que se encuentran en el Rac de control. También se debe detectar si se presenta algún corto en la etapa de potencia para detener el sistema y corregir la fallas existentes. En ambas modalidades el sistema debe poder interrumpir o suspender la ejecución de alguna operación y regresar al menú principal de dicha modalidad, aun cuando la suspención haya sido en forma externa.

II.1 Modalidad Perforado de Tarjetas.

Las partes en que se dividió esta modalidad para su desarrollo son:

a) Filtrado de datos.
b) Ordenamiento de los puntos.
c) Presentación y ampliación de la tarjeta en la pantalla.
c) Ejecución de la modalidad Perforado de Tarjetas.

II.1.1 Filtrado de datos.

Para realizar la identificación y almacenamiento de las coordenadas de cualquiera de los paquetes ORCAD. SMARTWORK o TANGO se debe contar con el archivo de texto el cual se explicará más adelante como se genera a partir del archivo fuente que tiene por extensión **.PCB**. Los datos filtrados se almacenan en dos diferentes de arreglos, identificados con el nombre de **colm** y **reng**. Los cuales pueden almacenar un máximo de 5000 datos como se define en la macro TAM.

```
#define TAM 5000
int reng[TAM],colm[TAM];
```

II.1.1.1 Paquete Orcad.

El paquete Orcad genera un archivo de texto el cual tiene la siguiente estructura:

```
C2    1000UF (90 )
1   2.600 2.100 N00003
2   2.600 2.300 N00010
U2    7905  TO220 (0 )
1   3.100 2.400 N00003
2   3.200 2.400 N00010  3
D1    DIODE (90 )
K   2.150 1.850 N00005

JP3   HEADER (270)
1   1.650 2.450 N00008
2   1.650 2.350 N00011
3   1.650 2.250 N00011
4   1.650 2.150 N00009
5   1.650 2.050 N00008
6   1.650 1.950 N00009

M***  MHOLE1 (0 )
XXXX 1.250 0.850
M***  MHOLE1 (0 )
XXXX 4.500 0.850
M***  MHOLE1 (0 )
XXXX 4.500 2.850
M***  MHOLE1 (0 )
XXXX 1.250 2.850
```

Realizando un análisis del contenido de dicho archivo, se puede establecer que para filtrar las coordenadas de los puntos X-Y, es necesario tomar en cuenta lo siguiente:

a) Todos los renglones terminan con el caracter 0x0D.
b) Las coordenadas están compuestas por una parte entera y una decimal.
c) La parte decimal consta de tres caracteres.

Partiendo del análisis anterior se desarrollo la función **arch_oread()**, la cual contiene una rutina que filtra las coordenadas y las almacena en los arreglos **colm** y **reng**, que corresponden a los ejes X y Y respectivamente.

Esta rutina analiza los caracteres contenidos en cada uno de los renglones del archivo tratando de localizar dos puntos decimales, de esta manera se puede decir que en ese renglón existe una coordenada.

La rutina cuenta con una estructura principal WHILE la cual en la parte interna se realiza la lectura e identificación de cada uno de los caracteres que se encuentran en los renglones, y también dará por terminada la lectura cuando identifique la marca de fin de archivo.

Conforme se van leyendo los caracteres se almacenan en un arreglo denominado **ar3** y también se lleva el recuento del número de puntos decimales que se encuentran en ese renglón por medio de la variable **con**.

Si al término de la revisión de cualquier renglón no se identifican los dos puntos decimales se inicializa la variable **con** y se continua con el renglón siguiente debido a que aquí no existe una coordenada.

Si por el contrario el renglón cuenta con dos puntos decimales se ingresa a una estructura FOR que localiza del arreglo **ar3**, el primer punto decimal, en ese momento se utilizan dos variables para delimitar ese dato partiendo del análisis anterior, es decir, se toman tres caracteres a la izquierda del punto decimal y cuatro a la derecha. Ese dato se guarda en un arreglo caracter por caracter al término esa cadena de caracteres se convierte en un dato tipo float y que al multiplicarla por 1000 genera un número entero el cual se guardar en el arreglo **colm**.

Después se continua con el renglón **ar3** hasta que encuentre el siguiente punto decimal y efectúa el mismo proceso que el anterior, aunque en este caso el dato resultante se guarda en el arreglo **reng** ya que este dato hace referencia al eje Y.

Este proceso se realiza con todos los renglones que contiene el archivo. A continuación se muestra la rutina que filtra las coordenadas.

```
while(ch!=EOF)
  {if(ch=='.')
    { con=con+1;}
     if (ch=='\xa')
      { if(con==2)
         {a=a-1;
          for(ki=0;ki<a;ki++)
          {n=ar3[ki];
           if(n=='.')
           {nn=ki-3;ll=ki+4; no=0;
            for(kj=nn;kj<=ll;kj++)
               {c=ar3[kj];
                switch(c)
                  {case '0':ar2[no]=c;no=no+1;break;
case '1':ar2[no]=c;no=no+1;break;
case '2':ar2[no]=c;no=no+1;break;
case '3':ar2[no]=c;no=no+1;break;
case '4':ar2[no]=c;no=no+1;break;
case '5':ar2[no]=c;no=no+1;break;
case '6':ar2[no]=c;no=no+1;break;
case '7':ar2[no]=c;no=no+1;break;
case '8':ar2[no]=c;no=no+1;break;
case '9':ar2[no]=c;no=no+1;break;
case '.':ar2[no]=c;no=no+1;break;}}
                   if(mm==1)
                    {mm=2;colm[o]=(1000*atof(ar2));o=o+1;}
                   else
                    {mm=1;reng[p]=(1000*atof(ar2));
                     p=p+1;gir=girar(gir,4);}}}
                     a=0;con=0;}
                 else
                  {a=0;con=0;}}
       else
        {ar3[a]=ch;a=a+1;}
         ch=getc(fp);}}
```

II.1.1.2 Paquete Tango.

El paquete Tango genera el siguiente archivo de texto.

```
M48
T01
X0035Y0285
X0105
T02
X0035Y0265
X0105
X0205
Y0275
Y0205
T03
X0035Y0245
X0105
T04
X0035Y0205
X0105
T05
X0035Y0145
X0105
M30
```

Analizando el contenido del archivo, se puede observar que:

a) No se repite ninguna coordenada completa, es decir, no se repite un dato X o Y en forma consecutiva.
b) El dato se encuentra predefinido por un caracter X o Y el cual establece a que parte de la coordenada corresponde.
c) En los renglones que existe una coordenada no aparece ningún otro tipo de información o caracter.
d) Todos los renglones terminan con el caracter 0x0D.

La función **arch_tango()**, emplea una rutina que filtra los datos y los guarda en los arreglos **reng** y **colm**. Esta rutina busca en los renglones la existencia de algún caracter X o Y, si no lo encuentra continua con el siguiente renglón, si por el contrario existen ambos caracteres, filtra los datos y los guarda en los arreglos correspondientes, si en dado caso de que sólo exista un dato cualquiera que este sea la coordenada se complementa con el dato que se obtuvo con anterioridad.

Para representar y almacenar los datos obtenidos en forma entera, el dato que se localizó se almacena caracter por caracter en un arreglo, de manera que se forme una cadena para luego convertirla en un dato tipo float, por último se multiplicará dicho dato por 10000, dando por resultado un valor entero. El punto decimal no aparece en el archivo de texto, este se obtiene cambiando el caracter X o Y por el punto decimal en el momento de almacenarlo en el arreglo. A continuación se muestra la rutina que filtra las coordenadas.

```
while (ch!=EOF)
  {if(ch=='\xa')
    {if (hayx==1)
      {if (hayy==1)
        {pix=1;pfx=m-1;nohay=0;
```

```
            piy=m;pfy=a-1;}
          else
           {pix=l;pfx=a-1;nohay=0;}}
         else
           {if(hayy==1)
            {piy=m;pfy=a-1;nohay=0;}
             else
              {nohay=1;}}
                if(nohay==0)
                 {char ar2[8]="          ";
                  char ar3[8]="          ";
                 if(hayx==1)
                    {cont1=0;
                     for(ki=pix;ki<=pfx;ki++)
                       {m1=ar1[ki];
switch(m1)
{ case 'X':ar2[cont1]='.';cont1=cont1+1;break;
case '1':ar2[cont1]='1';cont1=cont1+1;break;
case '2':ar2[cont1]='2';cont1=cont1+1;break;
case '3':ar2[cont1]='3';cont1=cont1+1;break;
case '4':ar2[cont1]='4';cont1=cont1+1;break;
case '5':ar2[cont1]='5';cont1=cont1+1;break;
case '6':ar2[cont1]='6';cont1=cont1+1;break;
case '7':ar2[cont1]='7';cont1=cont1+1;break;
case '8':ar2[cont1]='8';cont1=cont1+1;break;
case '9':ar2[cont1]='9';cont1=cont1+1;break;
case '0':ar2[cont1]='0';cont1=cont1+1;break;}}
                    valx=px*(10000*(atof(ar2)));
                     reng[o]=valx;valx2=valx; valx=0;o=o+1;}
              else
                 {reng[o]=valx2;o=o+1;}
                  if(hayy==1)
                     {cont2=0;
                      for(kj=piy;kj<=pfy;kj++)
                        {m2=ar1[kj];
switch(m2)
{ case 'Y':ar3[cont2]='.';cont2=cont2+1;break;
case '1':ar3[cont2]='1';cont2=cont2+1;break;
case '2':ar3[cont2]='2';cont2=cont2+1;break;
case '3':ar3[cont2]='3';cont2=cont2+1;break;
case '4':ar3[cont2]='4';cont2=cont2+1;break;
case '5':ar3[cont2]='5';cont2=cont2+1;break;
case '6':ar3[cont2]='6';cont2=cont2+1;break;
case '7':ar3[cont2]='7';cont2=cont2+1;break;
case '8':ar3[cont2]='8';cont2=cont2+1;break;
```

```
case '9':ar3[cont2]='9';cont2=cont2+1;break;
case '0':ar3[cont2]='0';cont2=cont2+1;break;}}
valy=px*(10000*(atof(ar3)));
          colm[p]=valy;valy2=valy;
          valy=0;p=p+1;gir=girar(gir,4);}
         else
          {colm[p]=valy2;p=p+1;}}
           a=0;pix=0;m=0;l=0;
           hayx=0;hayy=0;nohay=1;
           pfx=0;piy=0;pfy=0;}
      else
       {ar1[a]=ch;
         if(ch=='X')
          {l=a;hayx=1;}
         else
          {if (ch=='Y')
             {m=a;hayy=1;}}
             a=a+1;}
             ch=getc(fp);}}
```

II.1.1.3 Paquete Smartwork.

El paquete SMARTWORK genera el siguiente archivo de texto.

```
←H←H02X checkplot     10 Jul   95   16:10:28
p3.pcb
v1.4 r1 holes:   23            padmaster
approximate size:  0.65 by  0.25 inches
←H←W ←Kf   LLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLL
LLLLLLLLLLLLLLLLLLLLLLLLLLLLLL  ←←2←A
←Kf ••
                         ••←A←2←A
←Kf ••            |•µµ•¯↑      |•µµ•¯↑     |•µµ•¯|•µµ•¯↑                    |•µµ
•¯↑     |•µµ•¯↑          ••←A←2←A
←Kf ••             ÇÇÇ          ÇÇÇ |•µµ•¯↑ ÇÇÇ   ÇÇÇ |•µµ•¯↑     |•µµ•¯↑        ÇÇ
Ç        ÇÇÇ             ••←A←2←A
←Kf ••            |•µµ•¯↑             ÇÇÇ              ÇÇÇ         ÇÇÇ        |•µµ
•¯↑     |•µµ•¯↑          ••←A←2←A
←Kf                üüü          000 |•▸▸ ¯↑ 000 |•▸▸ ¯↑ 000 |•▸▸ ¯↑ 000 |•▸▸ ¯↑ üü
ü        üüü               ←←2←A
←Kf ••            |•µµ•¯↑      |•µµ•¯↑ ÇÇ  |•µµ•¯↑ ÇÇ  |•µµ•¯↑ ÇÇ  |•µµ•¯↑ ÇÇ  |•µµ
•¯↑     |•µµ•¯↑          ••←A←2←A
←Kf ••             ÇÇÇ          ÇÇÇ         ÇÇÇ         ÇÇÇ         ÇÇÇ        ÇÇ
Ç        ÇÇÇ             ••←A←2←A
←Kf ••◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆
◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆••←A←2←A
←2?
```

Examinando el contenido, se pueden establecer los siguientes puntos que nos permitan el desarrollo de una rutina que determinen las coordenadas de los puntos a perforar:

a) Una perforación está definida por los caracteres µ y þ.
b) Un renglón empieza con el caracter K.
c) El final del archivo está definido por el caracter 0x0C

La función **arch_sw()**, emplea una rutina que identifica y filtra los datos contenidos en un archivo para luego convertirlos en una coordenada, esto se lleva acabo empleando una estructura WHILE como base, en donde se realiza la lectura del archivo caracter por caracter hasta que encuentra el fin de archivo. Para determinar la coordenada donde se encuentran los puntos a perforar se utilizan dos variables enteras **r** y **c** las cuales llevan el conteo de los renglones y columnas respectivamente. La rutina incrementa el valor de la variable **c** conforme se va realizando la lectura de los caracteres, en el momento en que se encuentre el caracter K se incrementa el valor de la variable **r** y se inicializa el valor de **c**. Conforme se está realizando la lectura de los caracteres se van comparando con los caracteres que especifican la existencia de una perforación, en el momento en que la comparación sea verdadera al valor de las variables **r** y **c** se guardan en los arreglos **reng** y **colm** respectivamente. A continuación se muestra la rutina que filtra las coordenadas.

```
while (ch!=df)
    {ch=getc(fp);
     if(bkey1==1)
      {if (ch=='\x1B')
         {bkey1=2;ventana2();tipo=1;}
       else
        {tipo=0;
         elec2=mensaje(men2,elige2,1,64,0,14,15,1);
         gotoxy(20,20);
         switch(elec2)
           { case 0:ch=df;strcpy(archivo,                    ");break;}}}
         if (ch=='K')
          {r=r+1;c=0;}
         else
          {c=c+1;}
        if(vali=='µ')
          {if(ch=='µ')
            {colm[kk]=c;
             reng[kk]=r;
             kk=kk+1;
             aa=aa+1;
             gir=girar(gir,4);}}
```

```
                if (val=='þ')
                   {if(ch=='þ')
                      {colm[kk]=c;
                       reng[kk]=r;
colm[kk]=c;
kk=kk+1;
aa=aa+1;
gir=girar(gir,4);}}
            co=co+1;
            val=ch;
            vali=ch;}
        cont=aa;}
```

II.1.2 Organización de los puntos.

Cuando los datos se encuentren almacenados en los arreglos **colm** y **reng**, el siguiente paso es ordenar los datos de manera que lleven un orden ascendente y descendente.

Estos datos deben estar ordenados con respecto al eje Y en forma ascendente y con respecto al eje X se deben combinar ambas formas, es decir, el primer renglón debe estar ordenado en forma ascendente, el segundo en forma descendente y así sucesivamente hasta terminar con todos los renglones del circuito impreso de manera que se asemeje un desplazamiento en zigzag en el momento de la ejecución de esta modalidad. Para esto se desarrollo la función **ordena_zigzac()**, la cual se encarga del ordenamiento de datos. El método que se empleó es el de ordenamiento por permutación.

Este método compara el primer elemento del arreglo con cada uno de los elementos sucesivos, en el caso de que los elementos a comparar no cumplan con la condición establecida se intercambian dichos datos, si por el contrario la condición se cumple se continua con la comparación, hasta que se comparen cada uno de los elementos con los demás datos sucesivos.

Para llevar acabo el desarrollo de esta función se debe ordenar primero el arreglo **reng** que corresponde al eje Y y por último el arreglo **colm** que corresponde al eje X. Cuando se estén ordenando los elementos de los arreglos se debe tomar en cuenta que cuando se mueve un elemento del arreglo **reng** también se debe mover el elemento del arreglo **colm** al mismo lugar en que se movió el anterior elemento y viceversa para no modificar las coordenadas de los puntos a perforar. Este proceso se puede llevar acabo mediante el empleo de una estructura FOR anidada. A continuación se muestra la función que realza el ordenamiento por permutación.

```
void ordena_ziczac(int edo,int h2,int h,int kr,int z,int i,int ik,int xik,int ik2,int xik2,int yik,int
                   yik2)
          {int gir=1;
            for(ik=0;ik<cont;ik++)
              {xik=reng[ik];
               gir=girar(gir,2);
               for(ik2=0;ik2<cont;ik2++)
                 {xik2=reng[ik2];
                  if (xik<xik2)
                   {reng[ik]=xik2;
                    reng[ik2]=xik;
                    yik=colm[ik]; yik2=colm[ik2];
                    colm[ik2]=yik; colm[ik]=yik2;
                    xik=reng[ik];}}}
                    i=0;z=0;xik=reng[i];
                  for(i=0;i<cont;i++)
                    {xik2=reng[i];
                     if(xik==xik2)
                       {z++;}
                     else
                       {z=i-z;i=i-1;kr=i;
                        for(h=z;h<=kr;h++)
                          {yik=colm[h];
                            for(h2=z;h2<=kr;h2++)
                              {yik2=colm[h2];
                               if(yik<yik2)
                                 {colm[h]=yik2; colm[h2]=yik;
                                  yik=colm[h];}}}
                        i++;xik=reng[i];i--;z=0;}}
                        z=0;edo=0;i=0;xik=reng[i];
                   for(i=0;i<cont;i++)
                     {xik2=reng[i];
                      if(xik==xik2)
                        {z++;}
                      else
                        {if(edo==1)
                        {z=i-z;i--;kr=i;
                  for(h=z;h<=kr;h++)
                     {yik=colm[h];
                      for(h2=z;h2<=kr;h2++)
                       {yik2=colm[h2];
                        if(yik>yik2)
                          {colm[h]=yik2;colm[h2]=yik;
                           yik=colm[h];}}}
                           i++;xik=reng[i];
```

```
                    i--;edo=0;}
            else
              {edo=1;xik=reng[i];i--;}
          z=0;existe=1;}}}
```

Después de efectuar el ordenamiento se debe verificar que el circuito impreso cumpla con la condición de los cuatro puntos. Para esto se desarrollo la función **rut_detec()**.

La condición establece que sean cuatro puntos los que enmarquen a dicho circuito, pero sólo se verifica la existencia de dos de ellos, el primero y el último, es decir, el punto superior izquierdo y el punto inferior derecho. El punto superior izquierdo, se localiza como primer elemento del arreglo, el cual se compara en forma conjunta con todos los demás elementos del arreglo, verificando que la coordenada sea menor que cualquiera de las que existan en los arreglos. El punto inferior derecho no se puede asegurar que sea el último elemento del arreglo, debido a la forma en que se ordenaron los datos con anterioridad. Para esto se busca en forma conjunta la coordenada máxima y después se compara esta coordenada por separado con cada uno de los elementos que intervienen en los arreglos **colm** y **reng** respectivamente. En el momento en que se encuentre la coordenada máxima el valor con la cual se encontró se debe almacenar en al variable **rl** la cual se utilizará más adelante en el sistema. A continuación se muestra la función que verifica la condición de los cuatro puntos.

```
void rut_detec()
{int k=0,kk=0,ll=0,l=0,i=0,gir=0;
 k=reng[0];l=colm[0];rrd=0;
 kk=k;ll=l;
  for(i=1;i<cont;i++)
     {gir=girar(gir,1);
      if(k>reng[i]||l>colm[i])
        {rrd=1;break;}}
  if(rrd!=1)
    {for(i=1;i<cont;i++)
        {if(kk<reng[i]&&ll<colm[i])
           {kk=reng[i];ll=colm[i];}}
         for(i=1;i<cont;i++)
           {if(kk<reng[i]) rrd=1;
     if(ll<colm[i]) rrd=1;}}
   if(rrd==1)
   mensaje_tal_prin92();}
```

II.1.3 Presentar y ampliar la tarjeta.

Para mostrar y ampliar la tarjeta en la pantalla, partiendo de los datos que se encuentran en los arreglos **reng** y **colm** , se deben realizar ciertas conversiones de manera que todos los datos caigan en el área designada en la pantalla la cual esta delimitada entre 500 x 400 pixeles.

Para realizar las conversiones es necesario crear en modelo matemático que satisfaga las necesidades de conversión independientemente del paquete que se haya empleado. Este proceso de conversión es necesario a raíz de que los datos que se filtran tiene diferente estructura. El modelo empleado es:

```
x= floor(((colm[kn]- a)/b)-e);
y= floor(((reng[kn]-c)/d)-f);
```

Donde a, b, c, d, e, f son variables enteras y toman diferente valor dependiendo del paquete con el cual se esta trabajando.

En el paquete Smartwork, las variables toman los siguientes valores:

```
a=12; b=6; c-1; d=1; e=0; f=0
```

En el paquete Tango y Oread, las variables toman los siguientes valores

```
a=0; b=50; c=0; d=50; e=colm[0]/b; f=reng[0]/b;
```

El valor de cero es asignado a las variables que no son empleados en el modelo .

Los puntos que se filtraron del paquete Smartwork tienen las siguientes características: en el arreglo **colm**, los datos son múltiplos de 6 y por características del mismo archivo todos los datos tienen un incremento de 18 caracteres con respecto al dato original. Para el arreglo **reng**, sólo hay que restarle una unidad a los elementos arreglo de manera que represente el dato real con el cual fue diseñado. Con respecto a los datos que se hayan filtrado de los paquetes Oread y Tango sólo hay que dividirlos entre 50 ya que es la distancia que establecen los paquetes entre punto y punto para especificar una perforación.

El análisis anterior surge a raiz de que no se puede representar en la pantalla datos demasiado grandes ya que se encuentra con la limitante del tipo de resolución de la pantalla que es de 640 x 480 y seria imposible representar o dibujar un punto que tuviera por coordenada (4500,850) y más aun no se puede representar esta coordenada en un área de 500 x 400 pixeles.

Para una mejor comprensión de lo descrito anteriormente se toma como ejemplo el archivo de texto del paquete Orcad mostrado anteriormente. El modelo anterior debe representar un dato ascendente, es decir, si el primer punto tiene una coordenada (1250,850) el punto será X=0 y Y=0 y cuando tome la siguiente coordenada como por ejemplo, (1300, 850) el punto será X=1 y el valor de Y=0, así sucesivamente.

Para ampliar la tarjeta en la pantalla primero se debe calcular el número de veces que se puede ampliar tanto en el eje X como en el eje Y para después comparar ambos resultados y elegir el menor de ellos. Esto se lleva acabo empleando la siguiente rutina:

```
x=floor(500/(floor(((colm[rl]-a)/b)-e)));
y=floor(400/(floor(((reng[rl]-c)/d)-f)));
if(x==y) maxi=y;
else
  {if(y<x)
     maxi=y;
   else
     maxi=x;}
```

La función **pon_tarjeta_zoomO**, toma como referencia a la coordenada máxima localizada con anterioridad en el subíndice almacenado en la variable **rl**. Por ejemplo si los datos obtenidos del paquete Orcad, la coordenada máxima hubiese sido localizada en el subíndice 37 con coordenada (4500,2850), la variable **rl** sería igual a 37 y esto daría por resultado que el arreglo **colm[rl]** guarda un dato 4500 y el arreglo **reng[rl]** guarda un dato 2850, el valor de X será de 7, en tanto el valor de Y será de 10, efectuando la comparación la tarjeta se puede ampliar un máximo de 7 veces, por lo tanto **maxi** tomaría el valor de 7. La rutina que se muestra a continuación dibuja la tarjeta partiendo de los datos contenidos en los arreglos **reng** y **colm** además esta rutina puede ampliar la tarjeta dependiendo el valor del parámetro **maxima**.

```
void pon_tarjeta_zoom(int a,int b,int c,int d,int e,int f,int maxima,int resta)
{int ki,x1=0,y1=0,px=0,py=0,radio,inumx=0,inumy=0,fnumx=0,fnumy=0,i=0,j=0;
setlinestyle(SOLID_FILL,1,1);
for (ki=0;ki<cont;ki++)
  {x1=floor(((colm[ki]-a)/b)-e);
   y1=floor(((reng[ki]-c)/d)-f);
   inumx=((maxima*x1)-resta);
   numy=((maxima*y1)-resta);
   fnumx=inumx+resta;
   fnumy=inumy+resta;
   if(maxima<5)
    {for(i=inumy;i<=fnumy;i++)
        {for(j=inumx;j<=fnumx;j++)
```

```
            {py=i+50;px=j+110;
             putpixel(px,py,14);}}}
    else
     {px=(floor((inumx+fnumx)/2))+110;
      py=(floor((inumy+fnumy)/2))+50;
     radio=maxima/2;setcolor(14);
     setfillstyle(SOLID_FILL,YELLOW);
     fillellipse(px,py,radio,radio);}}}
```

II.1.4 Ejecución.

En la rutina **arch_ejecutar()**, esta diseñada de manera que de inicio a la perforación de la tarjeta por medio de la opción **EJECUTAR**, también es posible detener la operación de perforado en el momento en que se desplaza la plataforma de la mesa, esta opción se denomina **PAUSA**, la cual puede ser activada de manera externa por medio del interruptor denominado **STOP** que se encuentra en el Rac de control.

Es posible continuar con el proceso de perforación por medio de la opción denominada **CONTINUAR** o por el método externo el cual activa esta opción por medio de la pulsación del interruptor **RI-I** que se encuentra en el Rac de control.

Es posible reinicializar el sistema para empezar una nueva operación de perforado con una nueva tarjeta. La opción **SALIR** se encuentra disponible en cualquier momento en que se requiera suspender la operación. Esta rutina también detecta alguna señal de corto en la fuente de alimentación, en el momento en que suceda esta eventualidad en el sistema entra a un bucle infinito hasta que se restablezca la energía o se suspenda la ejecución de esta modalidad.

En esta rutina se utilizan las macros DMX, DMY, NVHFX, NVHFY, las cuales contienen los siguientes datos:

```
#define DMX 435      /*distancia máxima en X*/
#define DMY 215      /*distancia máxima en Y*/
#defina NVFHX 86.11  /*Número de vueltas de inicio a fin en X*/
#defina NVFHY 41.675 /*Número de vueltas de inicio a fin en Y*/
```

También se emplea la variable global denominada **motor** la cual guarda el valor de la resolución de los motores.

Cuando se diseña un circuito impreso, la pantalla se encuentra punteada dando como referencia que la distancia que existe entre punto y punto es de 0.05 pulg. ó 1.27 mm, tomando en cuenta que en un circuito integrado la distancia entre pin y pin es de 0.100 pulg. ó 2.54 mm, estos datos se deben tomar como referencia para calcular la distancia real de los datos contenidos en los arreglos **reng** y **colm**. Para esto se emplea la siguiente método:

```
sprintf(mensaje44,"%f",(float)((colm[ki]*multx)/dividex)-restx);
sprintf(mensaje55,"%f",(float)((reng[ki]*multy)/dividey)-resty);
```

Donde: **multx, multy, dividex, dividey**, toman diferentes valores dependiendo del tipo de paquete con el cual se esta trabajando.

Smartwork multx=2.54 multy=2.54 dividex=12 dividey=2

Tango y Orcad multx=2.54 multy=2.54 dividex=100 dividey=100

Los valores de **restx** y **resty** se calculan:

```
restx=(colm[0]*multx)/dividex;
resty=(reng[0]*multy)/dividey;
```

Cuando se hayan realizado los cálculos, se utiliza la función **envia()**, la cual convierte las datos **mansaje44** y **mensaje55** en su correspondiente en pasos, para después ser transmitidos. La función **envia()**, transmite la coordenada del punto a perforar al microcontrolador, pero antes se deben realizar una serie de operaciones que conviertan esos datos que se encuentran en milímetros. Para esto primero se calcula el número de pasos que se deben realizar para desplazar el eje X y el eje Y de inicio a fin, esto se realiza mediante la siguiente operación.

```
nmpasosx=ceil(motor*NVHFX);
nmpasosy=ceil(motor*NVHFY);
```

Por medio de esta relación es posible calcular la distancia en milímetros de un paso, tanto en el eje X como en eje Y:

```
sprintf (dfx, "%f",(float)DMX/nmpasosx);
upx=atof(dfx);
sprintf (dfx, "%f",(float)DMY/nmpasosy);
upy=atof (dfx);
```

Después de calcular estos datos se determina el sentido de desplazamiento de la plataforma en el eje X ya que cuando se desplace hacia la izquierda el dato que se transmita debe ser negativo y cuando se desplace hacia la derecha el dato será positivo.

Esto se logra mediante la suma de una cantidad al dato que se va ha transmitir. Esta cantidad se calcula a través del siguiente procedimiento: Primero, se toma la máxima resolución del motor que se encuentre definido por el sistema que en este caso es de 20000 pasos. Segundo, se calcula el número de pasos que se necesitan para desplazar la plataforma de inicio a fin, el resultado que se obtiene es expresado en tres formas diferentes :

$$nmpasosx=(1722200)_{10}=(1A4758)_{16}=(000110100100111010111000)_2$$
$$nmpasosy=(833500)_{10}=(0CB7DC)_{16}=(000011001011111111011100)_2$$

El resultado anterior nos muestra que para transmitir el dato de desplazamiento se necesita como mínimo tres bytes y el sentido lo determina el último bit. es decir, si el último bit es un 1 el dato será negativo, sí por el contrario es un 0 el dato será positivo. Así que, el dato a trasmitir se le tendrá que sumar la cantidad de 8388608 cuando el desplazamiento de la plataforma sea hacia la izquierda.

Para determinar que coordenadas serán negativas o positivas en el eje X es necesario comparar las coordenadas, esto es, si la coordenada actual es mayor que la coordenada anterior el desplazamiento será hacia la izquierda por lo tanto el dato que se transmita será negativo, si por el contrario la coordenada a perforar es menor el desplazamiento será hacia la derecha y el dato que se transmita al microcontrolador será positivo. Para convertir ese dato en su correspondiente en números de pasos sólo hay que dividirlo entre la distancia en milímetros de un paso, como por ejemplo:

En el caso de que el dato sea negativo.

```
sprintf(dfx,"%f",(float)dat2/upx);
dat3=atoi(dfx)+8388608;
```

En el caso de que el dato sea positivo.

```
sprintf(dfx,"%f",(float)dat2/upx);
dat3=atoi(dfx);
```

En el caso, del desplazamiento en el eje Y no es necesario establecer un sentido ya que el recorrido no varia. Por último el dato a transmitir ya sea positivo o negativo se debe enviar a las rutinas que codifiquen ese dato en hexadecimal y lo transmitan al microcontrolador.

La función de **espera()**, se emplea para que no se ejecute ninguna operación o acción hasta que se inicialice el sistema, es decir, que el equipo X-Y se encuentre en el punto considerado como inicio.

Cuando se ejecuta esta rutina el sistema entra a un bucle infinito el cual lee un dato constantemente del puerto serie, el cual lo compara con cualquiera de las tres opciones posibles: **L**, **F**, **?**.Si llegara un dato **L**, le indica al sistema que de terminada la ejecución. Si llega el dato **F**, entonces esto le indica al sistema que ha ocurrido alguna eventualidad y por lo consiguiente que envía un mensaje de error. Por último si llega el dato **?**, quiere decir que el sistema sea inicializado por completo y que puede continuar con alguna otra operación.

La función **aux_per_tarjeta_max()**, simula en la pantalla una perforación en la tarjeta cambiando de color el punto en el cual esta apuntando, después despliega en la pantalla la coordenada del siguiente punto a perforar. La función **inicial2()**, se emplea para restablecer la tarjeta y los datos a su estado inicial en la pantalla. La función de **pausa()**, se activa de manera interna o externa, es decir, por medio del teclado o a través de los interruptores que se encuentran en el Rac de control independientemente como sea invocada, esta rutina entra el un bucle infinito, el cual termina cuando recibe el dato que especifica continuar. Este dato puede llegar de manera interna o externa.

II.2 Modalidad Maquinado de Piezas.

Para el desarrollo de la modalidad **Maquinado de Piezas** , las funciones que la componen emplean ciertas constantes numéricas, las cuales se encuentran definidas por los macros: **MAXLONG, MAXRENG** y las variables globales: **final, inicial, cursor** y **apuntador**. Además es necesario emplear las macros y las variables que establecen las condiciones de la mesa X-Y y que se mencionaron con anterioridad. Estas variables se declaran de la siguiente manera:

```
# define DMX 435
# define DMY 215
# define NVHFX 86.11
# define NVHFY 41.675
# define MAXLONG 51
```

```
# define MAXRENG 17

int inicial=0 ,cursor=0,final=0 ;
int apuntador
unsigned char far * texto[1003];
```

La macro **MAXLONG**, define el número de caracteres que se podrán escribir por cada renglón. La macro **MINRENG**, establece el renglón en el cual empieza el área del editor. La macro **MAXRENG**, establece número de renglones que se podrán desplegar en el área designada para el editor.

La variable **inicial**, indica cual es el subíndice en la que se inicia el programa. La variable **final** , lleva el registro del número de líneas con las que cuenta el programa. La variable **cursor**, indica a que línea del programa esta haciendo referencia. La variable **apuntador**, indica a que línea del editor esta haciendo referencia.

También se emplea un Array de punteros definido con el nombre de **texto**, como esta variable emplea el modificador FAR, es necesario que se compile el sistema con el modelo de memoria LARGE, debido a que este tipo de modelos se emplea para aplicaciones grandes. Este Array de punteros almacena un máximo de 1000 renglones de programa y se le asigna a cada celda la memoria suficiente para almacenar 50 caracteres por renglón, mediante la función farmalloc().

II.2.1 Construcción de editor

El editor esta compuesto por una área de 17 renglones por 50 columnas, es decir, que sólo podrá desplazar y desplegar 17 renglones de un programa y en cada renglón se podrán escribir 50 caracteres como máximo. Para trabajar con el editor se cuenta con una serie de ordenes entre los cuales se encuentran:

a) Desplazamiento del cursor al inicio o fin del programa.
b) Desplazamiento del cursor al inicio o fin del área del editor.
c) Desplazamiento del cursor por medio de las flechas ←,↑,→,↓.
d) Desplazamiento del cursor por páginas.
e) Borrar una línea en la posición del cursor.
f) Insertar un renglón abajo del cursor.
g) Insertar un renglón arriba del cursor.

La función **Editar_Archivo()**, está construida por medio de la estructura de control DO-WHILE(), y en su cuerpo de sentencia cuenta con una bifurcación Switch(), la cual realiza la comparación de la tecla que se presiono y efectúa la tarea que se le haya establecido. Para esto la función **lee_alfa4()**, realiza la lectura de la tecla que se presiono, si esta tecla se encuentra comprendida entre los caracteres imprimibles, 0x21 hasta 0x7E, entonces lo escribe en la pantalla, los demás caracteres los ignora.

Pero si se presiona alguna tecla definida por dicha función en la cual se debe ejecutar una tarea entonces retorna un valor para que la bifurcación Switch(), la compara con una lista de valores enteros, si existe alguna coincidencia se ejecuta el bloque de sentencias que se asocien a esa constante.

Esta función realiza diferentes tareas en el editor para esto es necesario llevar un control de asignación de valores a las variables **inicio**, **fin**, **apuntador** y **cursor**, ya que estas variables indican el estado tanto del área del editor como del mismo programa, debido a que el programa puede superar el máximo de líneas desplegadas en el editor, para esto, es necesario llevar el control de la posición del cursor en el área del editor. La Figura 2-1 muestra la relación que existe entre el área del editor y un programa.

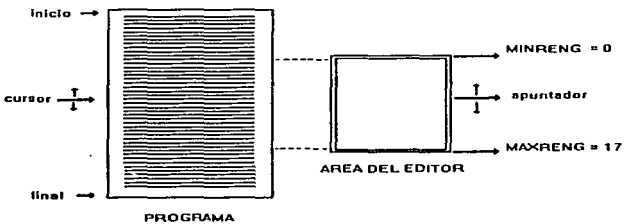


Figura 2-1. Relación del programa contra el área del editor.

La variable inicio y fin, siempre se encuentran ocupadas por las líneas especifican el inicio y fin del programa, esto quiere decir que al inicio de un nuevo programa este ya cuenta con tres líneas de texto, como se muestra en la figura 2-2.

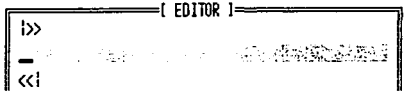


Figura 2-2. Inicio de un programa

Las rutinas que se presentan a continuación, emplean variables auxiliares entre las que se encuentran: **pfinal** para la variable **final**, **papuntador** para la variable **apuntador**, **pinicio** para la variable **inicio** y **pcursor** para la variable **cursor**.

La rutina que desplaza el cursor al inicio de la pantalla, realiza el desplazamiento de las variables **cursor** y **apuntador** al inicio, esto mediante la asignación del valor de cero a la variable **papuntador** para que se desplace al primer renglón del área del editor, y para desplazar la variable del **cursor** se debe realizar una diferencia entre el valor de la variable **cursor** y el valor que tenía la variable **apuntador** y el resultado se le asigna a la variable **cursor**. Por ejemplo: si el texto cuenta con 50 líneas y la variable **cursor** se encuentra apuntando al renglón 27 del programa y este se localiza en la línea 15 del área del editor se obtiene la diferencia de la siguiente manera:

cursor=cursor-apuntador

Colocando la variable **cursor** en el renglón 12 del programa y la variable **apuntador** en la línea 0 del editor, esto indica que el cursor se colocara al inicio del editor y del programa. Se deben tomar las siguientes consideraciones para no incurrir en algún error:

a) La variable **cursor** nunca debe apuntar al inicio o tener el un valor de cero.
b) Si ocurre el caso en el que la línea a la que se van a desplazar las variables **cursor** y **apuntador** estén ocupada por la línea del inicio del programa, entonces, se incrementan dichas variables en una unidad.
C) Si la variable **apuntador** se encuentra en la primera línea del área del editor, entonces, se debe ignorar dicha operación.

A continuación se muestra la rutina que desplaza el cursor al inicio de la pantalla.

```
pc=pcursor;
pa=papuntador;
if((pa-1)!=0 && (pc-1)!=0)
```

```
{if(pa!=0)
    {pc=pc-pa;pa=0;
     if(pa==0 && pc==0)
       {pa++;pc++;}}}
pcursor=pc;
papuntador=pa;
```

La rutina que desplaza el cursor al final de la pantalla del editor, realiza el procedimiento inverso a la rutina anterior, esto es, la variable **apuntador** se debe desplazar al final del área del editor, es decir, se iguala dicha variable con la macro **MAXRENG**, la variable **cursor** tendrá el resultado de la operación:

```
apuntador=MAXRENG
cursor=cursor+(MAXRENG-ap)
```

Esta operación se puede ejemplificar de la siguiente manera: Si el texto cuenta con 50 líneas y se encuentra en apuntando la variable **cursor** al renglón 27 y la variable **apuntador** se encuentra en la línea 15 del editor, efectuando la operación anterior la variable **cursor** apuntara al renglón 29 y la variable **apuntador** se encontrara en la línea 17. En esta rutina se tomaron las siguientes consideraciones que evitaran el mal funcionamiento de la rutina.

a) La variable **cursor** no puede apuntar la última línea del programa
b) Si la variable **apuntador** se encuentra en la última línea del editor, entonces se debe ignorar dicha tarea
c) Cuando la variable **cursor** sea mayor ó igual, que el valor de la variable **final**, entonces el desplazamiento se debe efectuar mediante los renglones que permite el programa, es decir, la variable **cursor** se iguala a la variable **final** menos una unidad y la variable **apuntador** obtiene el valor del renglón al cual debe apuntar a través de la operación :

```
apuntador=apuntador+(final-cursor-1)
```

A continuación se muestra la rutina que desplaza el cursor al final de la pantalla.

```
pc1=pcursor;pc=pc1;
pa1=papuntador;pa=pa1;
if((pa+1)!=pfinal && (pc+1)!=pfinal)
   { if(pa!=MAXRENG)
       {pc=pc+(MAXRENG-pa);
        pa=MAXRENG;
        if(pc>=pfinal)
          {pc=pfinal-1;
           pa=pa1+(pc-pc1);}}}
pcursor=pc;papuntador=pa;
```

La rutina que desplaza el cursor al inicio del programa, desplaza las variables **apuntador** y **cursor** al inicio, es decir, la variable **cursor** deberá apuntar al renglón número 1 ya que el renglón cero estará ocupado por la línea de inicio, y **apuntador** pasara al renglón número 1. Por último sólo se realiza la actualización de los datos que se encuentran en el área del editor, desplegando el programa contenido en el Array de punteros, **texto()**, y partiendo de cero hasta **MAXRENG**, mediante la estructura FOR. Para el desarrollo de esta rutina se tomaron en cuenta las siguientes consideraciones para evitar fallas.

a) Las variables **apuntador** y **cursor** no se deben encontrar en el renglón uno.
b) Cuando el programa sea menor en renglones al área del editor, sólo hay que efectuar el desplazamiento de las variables apuntador y cursor al renglón uno.

A continuación se muestra la rutina que desplaza el cursor al inicio del programa.

```
pa=papuntador;pc=pcursor;
if((pa-1)!=MINRENG && (pc-1)!=pinicial)
   {if(pfinal==MAXRENG)
     {pf=MAXRENG;}
    else
     {if(pfinal>MAXRENG)
       {pf=MAXRENG;
        for(i=0;i<=pf;i++)
            {escribe_video(3,i+MÁS,texto[i],cefondo3_3,ctexto3,ctexto3);}}
      else
       {pf=pfinal;}}}
pcursor=1;papuntador=1;
```

La rutina que desplaza el cursor al final del archivo, incrementa las variables **apuntador** y **cursor** una unidad menos que el valor de la macro **MAXRENG** y la variable **final**. Después se deben actualizar los datos contenidos en el área del editor, para esto se emplea una estructura FOR que en base a los valores que se le asignen a las variables que correspondan a la inicialización y a la condición se desplegará el contenido del Array de punteros.

Se emplean las variables **pf** asignada a la condición y **pf1** asignada a la inicialización. La variable **pf** será igual al valor de la variable **final** mientras que **pf1** será igual a la diferencia de **pfy** la macro **MAXRENG**.

Para el desarrollo de esta rutina se tomaron las siguientes consideraciones.

a) Las variables **apuntador** y **cursor** no se deben encontrar apuntando a la penúltima celda.
b) Cuando **apuntador** sea igual a **MAXRENG** y la siguiente línea del programa sea el fin de archivo se debe ignorar dicha operación.

A continuación se muestra la rutina que desplaza el cursor al final del archivo.

```
pa=papuntador;pc=pcursor;
if((pa+1)!=MAXRENG && (pc+1)!=pfinal)
   {if(pfinal==MAXRENG)
     {pf=MAXRENG;
      pf1=0;pcursor=pfinal-1;
      papuntador=pcursor;}
    else
     {if(pfinal>MAXRENG)
        {pf=pfinal;pf1=pf-MAXRENG;ii=0;
         for(i=pf1;i<=pf;i++)
            {escribe_video(3,ii+MÁS,texto[i],cefondo3_3,ctexto3,ctexto3);
             ii++;}
         pcursor=pfinal-1;papuntador=16;}
      else
        {pf=pfinal;pf1=0;
         pcursor=pfinal-1;
         papuntador=pcursor;}}}
```

La rutina que inserta un renglón en blanco en la posición de arriba del cursor, debe primero verificar que dicho renglón se pueda crear, para esto la variable **final** debe ser menor que 1000, ya que es el número máximo de renglones que se pueden escribir en un programa. Si el **final** es menor que 1000, entonces la variable se incrementa en la unidad para que se le asigne la suficiente memoria al Array de punteros para incrementar el programa en un renglón. Esto se realiza de la siguiente manera.

```
ultimo = final +1;
texto[ultimo] = (unsigned char*)farmalloc(sizeof(unsigned char)*51);
```

Después se realiza una copia del que ahora es el penúltimo renglón, que anteriormente era el último, al que ahora es el último renglón, y se realiza lo mismo con los subsecuentes renglones hasta llegar en donde se encuentra apuntado la variable **cursor**, en este lugar sustituirá la información por un renglón en blanco.

Por último se realiza una actualización de los datos en la pantalla a partir de donde se encuentra apuntado hasta la macro **MAXRENG**. En este caso el desplazamiento de los datos en el área del editor no siempre es hacia abajo, puede darse el caso que sea necesario efectuar el desplazamiento hacia arriba, esto ocurrirá cuando se inserte un renglón en el programa y la variable **apuntador** se encuentre en el último renglón en el área del editor. A continuación se muestra la rutina que inserta un renglón en blanco en la posición de arriba del cursor.

```
if (pfinal>1000)
  {alto();romper=0;}
else
  {ultimo=pfinal+1;
   texto[ultimo]=(unsigned char *)farmalloc(sizeof(unsigned char)*51);
   for(;;)
      {strcpy(texto[ultimo],texto[ultimo-1]);
       ultimo=ultimo-1;
       if (ultimo==pcursor) break;}
      strcpy(texto[pcursor],"                                        ");
      pinicial=0;pfinal=pfinal+1;
      if (papuntador<17)
        {pc=pcursor;pa=papuntador;
         for(;;)
            {escribe_video(3,pa+MÁS,texto[pc],ccfondo3_3,ctexto3,ctexto3);
             pc++;pa++;
             if(pc==pfinal)
               {escribe_video(3,pa+MÁS,texto[pc],ccfondo3_3,ctexto3,ctexto3);
                break;}
             if(pa==MAXRENG)
               {escribe_video(3,pa+MÁS,texto[pc],ccfondo3_3,ctexto3,ctexto3);
                break;}}pc=0;pa=0;}
            else
              {if (papuntador>17)
                  {papuntador--;pc=pcursor;pa=papuntador;
                   for(;;)
                      {escribe_video(3,pa+MÁS,texto[pc],ccfondo3_3,ctexto3,ctexto3);
                       pa--;pc--;
                       if(pa==MINRENG)
                        {escribe_video(3,pa+MÁS,texto[pc],ccfondo3_3,ctexto3,ctexto3);
                          break;}}
                           pc=0;pa=0;}}}
```

La rutina que borra una línea en la posición del cursor, efectúa una copia sucesiva desde la posición del cursor, hasta donde termina el programa. Esto quiere decir que, si nuestro programa cuenta con 10 líneas y se desea borrar la línea 3, se copia el contenido de la línea 4 en la línea 3, después se copia el contenido de la línea 5 en la línea 4, así sucesivamente hasta realizar la copia de la línea 10 en la línea 9, para que la memoria que ocupa la línea 10 se libere y se decremente la variable **final** en una unidad. Por último se debe realizar una actualización de datos que se encuentran en el área del editor partiendo donde se borro la línea hasta la última línea del editor. Esta rutina debe ignorar esta operación cuando sólo existan 3 líneas en el editor, es decir, la que hacen referencia a la línea de inicio, la línea uno del programa y la línea final. Debido a que la línea que se intenta borrar seria la línea uno y con esta se rompería la estructura de inicio para la escritura de un programa. A continuación se muestra la rutina que borra una línea en la posición del cursor.

```
if(papuntador==0&&(pcursor-1)==pinicial&&(pcursor+1)==pfinal)
 {strcpy(texto[1],"                                    ");
 escribe_video(3,0+MÁS,texto[pinicial],ccfondo3_3,ctexto3,ctexto3);
 escribe_video(3,1+MÁS,texto[pcursor],ccfondo3_3,ctexto3,ctexto3);
 escribe_video(3,2+MÁS,texto[pfinal],ccfondo3_3,ctexto3,ctexto3);
 pinicial=0,pfinal=2,pcursor=1,papuntador=1;}
else
 {if(papuntador==0&&(pcursor+1)==pfinal)
  {strcpy(texto[pcursor],texto[pcursor+1]);
  pcursor--;strcpy(texto[pfinal],"                                    ");
  farfree((void *)texto[pfinal]);pfinal--;
  escribe_video(3,papuntador+MÁS,texto[pcursor],ccfondo3_3,ctexto3,ctexto3);
  escribe_video(3,papuntador+MÁS+1,texto[pfinal],ccfondo3_3,ctexto3,ctexto3);}
 else
  {pf=pfinal;pfinal--;pc2=pcursor-1;
   if(pfinal==pcursor && pc2==pinicial)
    {pfinal++;strcpy(texto[pcursor],"                                    ");
    pc2=0;}
   else
    {pc=pcursor;pc1=pcursor;pa=papuntador;
    for(;;)
     {strcpy(texto[pc1],texto[pc1+1]);pc1=pc1+1;
     if (pc1==pf)
      {strcpy(texto[pc1],"                                    ");
      farfree((void *)texto[pc1]);break;}}
    pc1=0;pf=0;pinicial=0;
    if (papuntador==1 && pfinal==pcursor)
     {pcursor--;pc=pcursor;
     escribe_video(3,pa+MÁS,texto[pc],ccfondo3_3,ctexto3,ctexto3);
```

```
escribe_video(3,pa+MÁS-1,texto[pc-1],ccfondo3_3,ctexto3,ctexto3);}
else
 {if (pa==MAXRENG && pc<pfinal)
  {escribe_video(3,pa+MÁS,texto[pc],ccfondo3_3,ctexto3,ctexto3);}
 else
  {if(pfinal==pcursor)
    {papuntador--;pcursor--;}
   pc=pcursor;pa=papuntador;
   for(;;)
    {escribe_video(3,pa+MÁS,texto[pc],ccfondo3_3,ctexto3,ctexto3);
     pc++;pa++;
if(pc==(pfinal+1))
 {escribe_video(3,pa+MÁS,"                    ",ccfondo3_3,ctexto3,ctexto3);
 break;}
 if(pa>=MAXRENG)
  {escribe_video(3,pa+MÁS,texto[pc],ccfondo3_3,ctexto3,ctexto3);
  break;}}pc=0;pa=0;}}}}}
```

La rutina que inserta una línea en la parte de abajo del cursor, realiza la misma operación, que cuando se inserta un renglón en la parte de arriba del cursor. A continuación se muestra la rutina que inserta una línea en la parte de abajo del cursor

```
if (pfinal>1000)
 {alto();romper=0;}
else
 {papuntador=papuntador+1;pcursor=pcursor+1;
 ultimo=pfinal+1;
 texto[ultimo]=(unsigned char *)farmalloc(sizeof(unsigned char)*51);
 for(;;)
  {strcpy(texto[ultimo],texto[ultimo-1]);
  ultimo=ultimo-1;
  if (ultimo==pcursor) break;}
   strcpy(texto[pcursor],"                                    ");
   pinicial=0;pfinal=pfinal+1;
   if (papuntador<17)
    {pc=pcursor;pa=papuntador;
     for(;;)
      {escribe_video(3,pa+MÁS,texto[pc],ccfondo3_3,ctexto3,ctexto3);
       pc++;pa++;
       if(pc==pfinal)
        {escribe_video(3,pa+MÁS,texto[pc],ccfondo3_3,ctexto3,ctexto3);
        break;}
       if(pa==MAXRENG)
        {escribe_video(3,pa+MÁS,texto[pc],ccfondo3_3,ctexto3,ctexto3);
        break;}}
```

```
        pc=0;pa=0;}
    else
     {if (papuntador>17)
      {papuntador--;pc=pcursor;pa=papuntador;
      for(;;)
        {escribe_video(3,pa+MÁS,texto[pc],ccfondo3_3,ctexto3,ctexto3);
        pa--;pc--;
        if(pa==MINRENG)
        {escribe_video(3,pa+MÁS,texto[pc],ccfondo3_3,ctexto3,ctexto3);
        break;}}pc=0;pa=0;}}}
```

La rutina que desplaza el cursor un renglón hacia arriba, sólo decrementa el valor de la variable **apuntador** y **cursor** en una unidad. En este caso se debe tomar en consideración que cuando el cursor se encuentre en el primer renglón y por lo consiguiente **apartador** también apunte al primer renglón, cuando se presiona la tecla flecha hacia arriba se tendrá que ignorar dicha operación.

También se puede dar el caso en el que variable **apuntador** se encuentre en el primer renglón y la variable **cursor** se encuentre en el renglón 15 del programa la única variable que se decrementa será **cursor** y se tendrá que efectuar una actualización de los renglones que se encuentren en el área del editor. A continuación se muestra la rutina que desplaza el cursor un renglón hacia arriba.

```
if (pcursor!=1 && papuntador!=1)
 {if (pcursor==1 && papuntador==0)
  {pcursor--;pc=pcursor;pic=pc;pfc=pic+MAXRENG;ii=0;
  for(i=pic;i<=pfc;i++)
    {strcpy(texto2,texto[i]);
    escribe_video(3,ii+MÁS,texto2,ccfondo3_3,ctexto3,ctexto3);
    ii++;}pcursor++;papuntador++;}
 else
  {pa=papuntador;
  if(pa==0)
   {pcursor--;pc=pcursor;pic=pc;pfc=pic+MAXRENG;ii=0;
   for(i=pic;i<=pfc;i++)
   { strcpy(texto2,texto[i]);
   escribe_video(3,ii+MÁS,texto2,ccfondo3_3,ctexto3,ctexto3);
   if(strcmp(paso,texto2)==0)
     {break;}ii++;}}
   else
     {papuntador--;pcursor--;}}}romper=0;break;
```

La rutina que desplaza el cursor un renglón hacia abajo, realiza el proceso inverso a la anterior rutina ya que en este caso en vez de decrementar en una unidad las variables **apuntador** y **cursor**, se incrementaran en una unidad. En esta rutina la variable **cursor** no puede tomar el mismo valor que la variable **final** cuando se caiga en este caso se debe ignorar la ejecución de este proceso. También puede darse el caso en el que la variable **apuntador** se encuentre en el último renglón y **cursor** se encuentra en el renglón 10 de un programa que cuenta con 30 renglones entonces la única variable que se incrementa en una unidad será **cursor** mientras que **apuntador** no modifica su estado, y sólo se tendrá que efectuar la actualización de los renglones que se encuentran en el área del editor.

```
if((pcursor+1)==pfinal)
 {if((pcursor+1)==pfinal && papuntador==MAXRENG)
  {pcursor++;pc=pcursor;pfc=pc;pic=pc-MAXRENG;ii=MAXRENG;
  for(i=pfc;i>=pic;i--)
  {escribe_video(3,ii+MÁS,texto[i],cefondo3_3,ctexto3,ctexto3);
  ii--;}pcursor--;papuntador--;}}
 else
  {if((pcursor+1)!=pfinal && papuntador!=MAXRENG)
   {pa=papuntador;
    if(pa==MAXRENG)
     {pcursor++;pc=pcursor;pfc=pc;pic=pc-MAXRENG;ii=MAXRENG;
     for(i=pfc;i>=pic;i--)
     {escribe_video(3,ii+MÁS,texto[i],cefondo3_3,ctexto3,ctexto3);
    ii--;}}
 else
  {papuntador++;pcursor++;}}}
```

La operación que desplaza el cursor hacia la izquierda o hacia la derecha se encuentra integrada en la función **lee_alfa4()**. Así como también las rutinas que realizan el proceso de insertar y sustituir un caracter.

La rutina que desplaza el cursor una página hacia abajo, debe efectuar una serie de cálculos para identificar el inicio y el fin del bloque de renglones que se desplegaran en la pantalla. Para esto, partiendo de la posición que guarda la variable **papuntador** se le resta el valor de la macro **MAXRENG** y se le suma el valor de la variable **pcursor**, el resultado de esta operación indica el inicio del bloque de renglones a desplegar, el fin de este bloque se calcula mediante la suma del resultado anterior y el valor de la macro **MAXRENG**.

Para no perder la posición del cursor en la pantalla se debe mantener con el mismo valor a la variable **papuntador** y sólo se incrementa la variable **pcursor** por medio de la suma del resultado de la primera operación y el valor que tiene la variable **papuntador.**

Cuando se presente el caso en el que se requiera desplegar el cursor una pantalla hacia abajo y el renglón que especifica el fin del programa se encuentre en el área del editor, esta operación se debe ignorar. También puede darse el caso en el cual se requiera pasar a la siguiente pantalla pero el número de renglones a desplegar es menor que el área del editor, entonces, sólo hay que desplegar dichos renglones y ajustar por medio de otros cálculos y asignaciones los valores de las variables **papuntador** y **pcursor.** A continuación se muestra la rutina que desplaza el cursor una página hacia abajo

```
pc=pcursor;pa=papuntador;pf=pfinal;
if((pa+1)!=pfinal && (pc+1)!=pfinal)
 {if (pf>=MAXRENG)
  {if ((pc-pa+MAXRENG)<=pf)
   { pc1=(MAXRENG-pa)+pc;pc2=pc1+MAXRENG;pcursor=pc1+pa;
   if(pc2>=pfinal)
   {marco(1,5,54,24,64,15,0,"[ EDITOR ]");
   escribe_video(54,21,"\x1E",112,4,4); escribe_video(54,22,"_",112,4,4);
   escribe_video(54,23,"\x1F",112,4,4); escribe_video(37,24,"\x11",112,4,4);
   escribe_video(38,24,"_",112,4,4);
   escribe_video(39,24,"\x10",112,4,4);
   pc2=pfinal;
   if (pcursor>=pfinal)
     {pcursor=pc2-1;papuntador=pcursor-pc1;}}ii=0;
     for(i=pc1;i<=pc2;i++)
      {escribe_video(3,ii+MÁS,texto[i],ccfondo3_3,ctexto3,ctexto3);
       ii++;}}}}
```

La rutina que desplaza el cursor una pagina hacia arriba, se deben realizar al igual que en la rutina anterior una serie de cálculos para identificar el inicio y el fin del bloque de renglones que se deben desplegar en la pantalla del editor. Para esto partiendo de la posición que guarda la variable **pcursor** se le resta el valor de la variable **pauntador,** este resultado se toma como el último renglón del bloque a desplegar.

Para determinar el primer renglón del bloque, al resultado anterior se le resta la macro **MAXRENG**. Para no perder la posición del cursor, la variable **papuntador** no cambia de valor, mientras que la variable **pcursor** toma el valor que resulta de la suma de la segunda operación y el valor de la variable **papuntador.**

Cuando se presente el caso en el que se requiera desplegar la pantalla anterior y el primer renglón que especifique el inicio del programa se encuentre en el área del editor, esta operación se debe ignorar. Cuando se de el caso que se requiera pasar a la siguiente pantalla pero el número de renglones a desplegar sea menor que el número de renglones del editor entonces sólo se desplegara el primer renglón que especifica el inicio del programa hasta el número máximo de renglones que permita el área del editor, la variable **papuntador** no cambiará su valor, pero la variable **pcursor** se debe igualar a la variable **papuntador**. A continuación se muestra la rutina que desplaza el cursor una página hacia arriba.

```
pc=pcursor;pa=papuntador;
if((pa-1)!=pinicial && (pc-1)1=pinicial)
 {if ((pc-pa)!=pinicial)
  { pc1=pc-pa;pc2=pc1-MAXRENG;
  if(pc2<=pinicial)
   {pcursor=0+pa;}
  else
   {pcursor=pc2+pa;}
  if(pc2<=pinicial)
   {pc2=pinicial;pc1=MAXRENG;
    if (pcursor<=pinicial)
      {pcursor=pc2+1;papuntador=pcursor;pc1=MAXRENG;}}ii=0;
       for(i=pc2;i<=pc1;i++)
        {escribe_video(3,ii+MÁS,texto[i],cefondo3_3,ctexto3,ctexto3);
        ii++;}}}
```

II.2.2 Construcción del compilador.

El compilador se encuentra dividido en dos módulos. El primer módulo, se encarga de revisar que cada una de las sentencias que conforman al programa cumpla con las reglas precisas de formación. Así como, también debe llevar acabo el conteo del número de errores que se van generando en dicha revisión. El segundo módulo se encarga de realizar una codificación de las sentencias contenidas en el programa, en un sistema numérico hexadecimal, dichos datos son los que recibe el microcontrolador para efectuar una tarea. Durante la codificación, los datos se deben almacenar en un archivo que tenga por extensión **.HEX**, para su uso posterior.

Las sentencias que se muestran en la Tabla 2-1 son las que se emplean para diseñar programas y sobre estas se realiza la revisión de sintaxis y la conversión de los mismos en hexadecimal.

VALOR	SENTENCIA	DESCRIPCIÓN	PARÁMETROS
1	ANG(X,Y)	ACELERACIÓN ANGULAR	0.01< X >999.99 0.01< Y >999.99
2	ALN(X,Y)	ACELERACIÓN LINEAL	0.01< X >999.99 0.01< Y >999.99
3	RNG(±X,±Y)	RECORRIDO ANGULAR	-20000< X >20000 -20000< Y >20000
4	RLN(±X,±Y)	RECORRIDO LINEAL	-435< X >435 -215< Y >215
5	MVA(±X,±Y)	MOVIMIENTO ABSOLUTO	-20000< X >20000 -20000< Y >20000
6	MVR(±X,±Y)	MOVIMIENTO RELATIVO	-20000< X >20000 -20000< Y >20000
7	RMV()	REPITE MOVIMIENTO	
8	CDI()	CAMBIA DIRECCIÓN EN X y Y	
9	CDX()	CAMBIA DIRECCIÓN EN X	
10	CDY()	CAMBIA DIRECCIÓN EN Y	
11	AHF(N)	ACELERACIÓN A HOME	0.01< N >999.99
12	VHF(N)	VELOCIDAD A HOME	0.001< N >20.00
13	GHM()	IR A HOME	
14	GEN()	IR A FIN	
15	VAN(X,Y)	VELOCIDAD ANGULAR	0.001< X >20.00 0.001< Y >20.00
16	VLN(X,Y)	VELOCIDAD LINEAL	0.001< X >20.00 0.001< Y >20.00
17	GAN()	EJECUTA MOVIMIENTO ANGULAR	
18	GLN()	EJECUTA MOVIMIENTO LINEAL	
19	MAN(±X,±Y)	EJECUTA MOVIMIENTO ANGULAR	-20000< X >20000 -20000< Y >20000
20	NOP()	NO OPERACIÓN	
21	RMD(#)	RESOLUCIÓN DEL MOTOR	50< # >20000
22	RIP()	INICIALIZAR PROCESO	
23	EFT()	ENCIENDE FUENTE	
24	AFT()	APAGA FUENTE	
	GSB(#)	BRINCA A SUBRUTINA	0 < # > 100
	SUB(#)	SUBRUTINA	0 < # > 100
	RTE()	RETORNO DE SUBRUTINA	
	ILCI(#,N)	INICIO DE CICLO	0 < # > 100 0 < N > 1000
	FLP(#)	FIN DE CICLO	0 < # > 100
25	LBF()	LIMPIA BUFFER	
26	ILX(#,±N)	MOV. LINEAL 'X' EN INCREMENTOS	0 < # > 100 -20000 < N > 20000
27	ILY(#,±N)	MOV. LINEAL 'Y' EN INCREMENTOS	0 < # > 100 -20000 < N > 20000
28	ITL()	INTERRUPCION TOTAL	
29	IPS()	PAUSA	
30	CNP()	CONTINUA	
31	ATL()	ACTIVA TALADRO	
32	IPX(#,±N)	MOV. PASOS 'X' EN INCREMENTOS	0 < # > 100 -20000 < N > 20000
33	IPY(#,±N)	MOV. PASOS 'Y' EN INCREMENTOS	0 < # > 100 -20000 < N > 20000
34	BEG()	INICIO DE PROGRAMA	
35	END()	FIN DE PROGRAMA	
37	PEA()	POSICIÓN ABS. CODIFICADOR X,Y	
38	PER()	POSICION REL. CODIFICADOR X,Y	
39	INP(#)	ENTRADA BINARIA	# = BYTE
40	OUT(#)	SALIDA BINARIA	# = BYTE
41	LHX()	LIMITE HOME X	
42	LHY()	LIMITE HOME Y	
43	LFX()	LIMITE END X	
44	LFY()	LIMITE END Y	
45	EPP()	ENVIA POSICION PASOS	
46	WIN(#)	ESPERA ENTRADA BINARIA	# = BYTE
47	MPT(#)	MUEVETE AL PUNTO	0 < # > 1000
48	MAP(#)	MUEVETE A TRAVES DEL PUNTO	0 < # > 1000
49	PNT(#)	PUNTO	0 < # > 1000
50	DLY(#)	RETARDO	
51	ESC(#)	ESCALA	1 < # > 999.999
52	ATC().	ACTIVA TALADRO CONTINUAMENTE	0.001 < # > 1000
53	PTL()	APAGE TALADRO	
54	CMX()	CALACULA MOVIMIENTO 'X' EN BASE A VELOCIDAD, ACELERACIÓN Y POSICION	
55	CMY()	CALCULA MOVIMIENTO 'Y' EN BASE A VELOCIDAD, ACELERACION Y POSICION	
56	MVL(±X,±Y)	EJECUTA MOVIMIENTO LINEAL	-20000< X >20000 -20000< Y >20000

Tabla 2-1. Sumario de sentencias.

II.2.2.1 Revisión de sintaxis.

Para llevar acabo la revisión de sintaxis del programa, se emplean las funciones, **sintaxis0**, **sintaxis2()** y **sintaxis3()**, las cuales a su vez se auxilian de otras funciones, que se mencionan más adelante. La función de **sintaxis0**, lleva acabo la revisión del programa en base a las siguientes condiciones:

a) Las sentencias se deben escribir con letras mayúsculas.
b) Las sentencias deben estar precedidas por un paréntesis
c) Todas las sentencias deben terminar con un punto y coma.
d) El argumento, si esta compuesto por dos datos, debe estar separado por una coma.
e) En un renglón puede haber más de una sentencia.
f) En una sentencia no pueden existir espacios en blanco.

Lo anterior se lleva acabo efectuando la revisión renglón por renglón y caracter por caracter. Para esto, se copia el segundo renglón del programa en una variable declarada como una cadena de caracteres, después se analizan todos los caracteres que se encuentran en esa cadena, llevando el conteo de las veces que se presentan y en que punto de la cadena se encontraron los caracteres, corchetes ([ ]), paréntesis(( )), coma(,), punto y coma (;), así como también el caracter que hace referencia a los espacios en blanco. Para llevar el registro de las sentencias que aparecen en un renglón se detecta el caracter (, se retroceden tres caracteres y estos se convierten en una cadena para después efectuar la comparación de dicha cadena con un Array de punteros que fue inicializado con todas las sentencias que se emplean para el desarrollo de programas.

Si existe alguna coincidencia se detiene el proceso de comparación y se incrementa la variable que lleva el conteo de las sentencias así como, en el punto en el que se encuentra dicha sentencia. Si por el contrario no existe ninguna coincidencia, entonces se continua con la revisión de los demás caracteres.

Cuando se termine de efectuar dicha tarea, se debe comparar si la variable que lleva el conteo del número de espacios en blanco es igual a 50, si la repuesta es verdadera entonces se continua con el siguiente renglón, si por el contrario la respuesta es falsa entonces se prosigue con la revisión del siguiente renglón debido a que en ese renglón no existe ninguna sentencia. Después se efectúan la siguiente comparaciones.

a) El número de corchetes que abren debe ser igual al número de corchetes que cierran.
b) El número de paréntesis que abren debe ser igual al número de paréntesis que cierran.
c) El número de sentencias debe ser igual al número de paréntesis que abren e igual al número de puntos y comas existan.

Si las anteriores condiciones se cumplen entonces, se revisa que después de cada paréntesis que abre exista un paréntesis que cierre seguido de un punto y coma. Después se revisa que el argumento que se encuentre delimitado por los paréntesis, dicho argumento debe estar compuesto por los dígitos del 0 al 9. Puede darse el caso en el que aparezcan los caracteres: menos(-), punto(.) y coma(,). Debido a que estos datos pueden ser negativos y fraccionarios, así que en el momento en que aparezcan cualquiera de estos caracteres se debe revisar que cumplan con las siguientes condiciones.

a) No pueden existir más de una coma.
b) La coma debe estar antes que el caracter menos.
c) El punto debe estar precedido por un dígito.

Para finalizar se realiza el recuento de los caracteres que se analizaron, este recuento se lleva acabo sumando los caracteres que se emplearon en las sentencias más los caracteres de los comentarios, así como también los espacios en blanco.

Cuando alguna de las anteriores condiciones no se cumpla se crea un archivo para guardar los diferentes mensajes de error que se detectaron, así como también en que línea se presentaron. Hasta que se hayan analizado todos los renglones se detiene el proceso de revisión de sintaxis y se procede a realizar las correcciones en el programa, en base a los mensajes de error que se hayan almacenado en el archivo. El tipo de error esta determinado en base a cualquier condición que no se haya cumplido, por ejemplo: Cuando la suma de los caracteres sea diferente a 50 se almacena en el archivo el mensaje CARAC. INDEF.

Cuando el argumento no cumpla con las condiciones entonces se almacena el mensaje de error ARG(?). Cuando sea mayor el número de corchetes que abren de los que cierran o viceversa se almacena el mensaje MENSAJE[]. Se almacena el mensaje ? INSTRUC, cuando exista un error en los delimitadores del argumento, en la escritura de la sentencia, así como también en el punto y coma .

En dicho archivo sólo se podrán almacenar 50 mensajes de error, esto quiere decir que si en esta primera fase el programa tiene más de 50 errores se detendrá el proceso para corregir dichos errores.

La función **sintaxis2()**, revisa el argumento de cada una de las sentencias que intervienen en el programa de manera que el o los datos que lo conforman, no sobrepasen los límites establecidos. Para llevar acabo esta revisión se emplea las funciones: **checa()**, **cantidad_cero()**,**cantidad_uno()** y **cantidad_dos()**.

La función **sintaxis2()** al igual que la función **sintaxis ()**, realiza la misma exploración del programa, aunque en este caso se lleva sólo el registro de la posición que ocupan los paréntesis, corchetes y comas. En este caso no se guardar la posición en que empieza la sentencia si no que se debe guardar en un arreglo bidimencional la palabra que hace referencia a la sentencia, para después compararlas con el contenido del arreglo de punteros que se declarado al inicio el cual se inicializo con todas las sentencias que se pueden emplear para diseñar un programa, de esta comparación se obtiene el resultado de la posición en que fueron ordenadas las sentencias, este resultado es un valor entero, el cual al ser comparado con una lista de constantes enteras contenidas en una bifurcación SWITCH() y al encontrar la coincidencia se ejecutan el bloque de sentencia que se encuentran asociadas a dicha constante y que hacen referencia a las condiciones que rige el dicho argumento.

Las sentencias pueden estar formadas por uno, dos o ningún dato, para esto primero es necesario identificar el número de comas que se encuentran en el argumento, esto lo realiza la función **checa()**, que en base a los parámetros transmitidos en el momento que se invoco a dicha función retorna un valor cero cuando no exista ninguna coma esto indica que el argumento puede estar compuesto por uno o ningún dato. Si retorna un uno entonces el argumento esta compuesto por dos datos.

Las funciones **cantidad_cero ()**, **cantidad_Uno()** y **cantidad_cos()**, se encuentran distribuidas en cada una de las opciones de la bifurcación SWITCH (), en base a las condiciones de los argumentos establecidos para cada sentencia la cual se puede observar en la Tabla 2-1. Las sentencias que no emplean un argumento, la revisión se llevara acabo por la función **cantidad_cero()** ya que sólo revisa que la posición del paréntesis que abre, más una unidad sea igual al valor de la posición del paréntesis que cierra. Esta función sólo emplea los parámetros que se registraron con anterioridad, la posición de los paréntesis y del número de sentencias por renglón.

Las sentencias que emplean un argumento compuesto por un dato se emplea la función **cantidad_uno()**, esta función es necesario que se le transmitan al momento de ser invocada, los parámetros de la posición de los paréntesis, el número de la sentencia que ocupa en el renglón, las cantidades que delimitan al dato y el tipo de dato que se debe revisar.

El dato se guarda caracter por caracter en un arreglo para conformar una cadena de caracteres y al final se debe convertir en un dato tipo float para después verificar que el dato se encuentre entre las cantidades que delimitan al dato. Si entre los caracteres que esta guardando se encuentra un punto, entonces se realiza la revisión del tipo de dato que debe contener la sentencia ya que puede ser entera o fraccionaria. Como se puede observar en la Tabla 2-1.

Cuando la sentencia emplee un dato en el argumento y este deba ser un dato binario se emplea la función **cantidad_uno_bin()**, la cual utiliza los mismos parámetros que la anterior función con excepción del parámetro que especifica el tipo de dato. Como un dato binario sólo admite unos y ceros, entonces sólo revisa que los caracteres que se encuentren entre los paréntesis sean unos y ceros.

Las sentencias que emplean el argumento compuesto por dos datos utilizan la función **cantidad_dos()**, esta función emplea los mismos parámetros que la función **cantidad_uno()**, aunque estos parámetros se deben especificar para cada uno de estos, también se emplea el parámetro que guarda la posición de la coma. En esta función la rutina de revisión para cada uno de los datos es la misma que se empleo en la función **cantidad_uno()**, aunque para este caso el primer dato se encuentra entre el paréntesis que abre y la coma, y por lo consiguiente el segundo dato se encuentra entre la coma y el paréntesis que cierra.

En el caso que existiera algún error en el argumento independiente mente de lo que sea, las funciones retornan un cero a la función **sintaxis2()**, y esta a su vez abre el archivo de mensajes de error en el cual se almacena el mensaje RANGO INDEF, y la linea en la que ocurrió dicho error. Este procedimiento de revisión continua hasta que termina con la última sentencia por lo tanto se continua almacenando más errores conforme estos sigan apareciendo. Si por el contrario los argumentos no tienen ningún error entonces se retorna un uno a la función **sintaxis2()**, que indica que no existe ningún error y por lo consiguiente continua examinando los demás renglones del programa. La revisión de la estructura del programa queda a cargo de la función **sintaxis3()**, esta función realiza el análisis del programa renglón por renglón y caracter por caracter, al igual que las anteriores funciones **sintaxis ()**, **sintaxis2()**.

Esta función genera un registro de la posición y el número de sentencias que se encuentran en el programa pero sólo aquellas que forman parte de la estructura de control JLP() y FLP(), las sentencias que limitan el cuerpo del programa BEG(), END() y las sentencias que se emplean para declarar las subrutinas SUB() y RTE(). Además se debe llevar el registro de la sentencia GSB(), que es la que efectúa las llamadas de las subrutinas. Para diseñar esta función se debe tomar en cuenta las siguientes condiciones.

a) No existe recursión.
b) Se puede anidar bucles.
c) No puede existir bucles con el mismo número de clasificación.
d) No puede existir subrutinas con el mismo agrupamiento.
e) No se puede llamar a una subrutina que no exista.
f) Las sentencias anteriormente mencionadas pueden estar precedidas por un mensaje nunca por otra sentencias.

La primera tarea que debe realizar es la búsqueda del caracter que define al paréntesis que abre, cuando lo encuentra se regresa tres caracteres y los convierte en una cadena para después compararla con las sentencias que se mencionaron con anterioridad. Conforme se vayan revisando los renglones del programa se debe de ir registrando el número de veces que aparece cualquiera de las anteriores sentencias, así como también el renglón en donde apareció y en los casos en que aparezcan las sentencias JLP(), FLP(), SUB() y GSB(), se registrara el argumento que contienen.

Después se lleva acabo una serie de comparaciones que comprueban la estructura del programa en base a las condiciones establecidas con anterioridad.

Para empezar la variable que lleva el registro del número de veces que aparecieron las sentencias BEG() y END() debe ser igual a uno y el renglón donde apareció la sentencia BEG() debe ser menor que el renglón donde apareció la sentencia END(). Después revisa que el número de sentencias SUB() sea igual al número de sentencias RTE() y que el número de sentencias JLP() sea igual al número de sentencias FLP().

La función **estructura()**, revisa que se encuentren bien escrita que el orden en que aparecieron las sentencias JLP() y FLP(), para crear una estructura de bucles anidados. Para esto se emplean dos arreglos **paso2** y **paso**, en donde se almacenan, los datos consecutivamente. Para esto el arreglo **paso2** contiene los argumentos de las funciones JLP() y FLP() en el orden que aparecen en el programa. Este almacenamiento se efectúo dentro de la función **sintaxis3()**. La Figura 2-3 muestra el método en que se revisa la estructura de los bucles anidados.

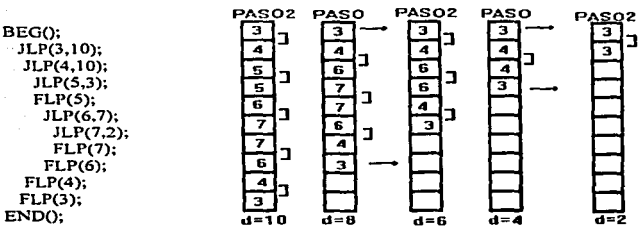


Figura 2-3. Revisión de bucles anidados

También se emplea una variable **K** que se iguala a una variable **d** la cual lleva el contenido de los elementos de los arreglos. El procedimiento se puede resumir en una serie de pasos que se describen a continuación:

1) Se comparan los elementos del arreglo **paso2** por parejas en forma ascendente, es decir, el uno con el dos, el tres con el cuatro, el cuatro con el cinco, etcétera.
2) Los elementos idénticos se discriminan y se reduce el valor de **d** en dos unidades.
3) Los elementos diferentes se almacenan en el arreglo **paso** en ese orden.
4) Si el valor de **d** es igual a cero entonces se da por terminada la revisión.
5) Si el valor de **d** es diferente de cero, se toma el primer elemento del arreglo **paso** y lo guarda en el arreglo **paso2**.
6) Se comparan los elementos del arreglo **paso**, por parejas, pero sólo aquellos elementos que se encuentren entre el primero y el último elemento del arreglo.
7) Si los elementos son diferentes se almacenan en el arreglo **paso2** en ese orden.
8) Si los elementos son idénticos se discrimina y se reduce el valor de **d** en dos unidades.
9) Al término de la comparación se almacena el último elemento de la regla **paso** en el arreglo **paso2**.
10) Si la variable **d** es diferente de cero se repite el proceso desde el primer inciso.
11) Si la variable **d** es igual a cero se da por terminada la revisión retornando un valor uno a la función **sintaxis3()** indicando que la estructura este bien escrita.
12) Cuando se repite esta secuencia y la variable **d** no llegue hacer cero entonces se retorna un valor cero indicando que existe algún error en la estructura.

Cuando exista alguna diferencia en las comparaciones hechas en las funciones **sintaxis()**, **sintaxis2()** y **sintaxis3()**, así como también en cualquiera de las funciones en las que se auxilia para realizar la revisión de sintaxis del programa, se abre un archivo en donde se almacena el tipo de error y la línea en que se presenta. Para esto cuando se suspenda la operación de revisión de sintaxis se debe mostrar en la pantalla el contenido de dicho archivo en el área de mensajes, este procedimiento queda a cargo de la función **men_error()**.

II.2.2.2 Creación del archivo objeto

Para realizar la codificación de las sentencias que se encuentran en el programa, en datos en hexadecimal y después guardarlos en un archivo se emplean las funciones **gen_hex()**, **gen_hex_tot()** y **gen_hex_beg_end()**.

La función **gen_hex()**, genera la cadena con el nombre del archivo en donde se almacenan los datos codificados. Para generar dicha cadena primero se copia la cadena que contiene la ruta, el nombre y extensión del archivo fuente, en una cadena con el nombre **linet**, después se busca el caracter punto para sustituir los tres caracteres siguientes por los caracteres H, E, X, de esta manera se modifica la extensión DAT por HEX del la cadena **linet**.

Cuando se cuente con el nombre y la extensión del archivo se procede a revisar que el archivo no exista en la ruta especificada, si el archivo existe entonces se elimina debido que los datos que se almacenaran serán diferentes a los datos existentes en este archivo.

La función **gen_hex_tot()**, genera el registro de las sentencias BEG(), END(), SUB(), RTE(), JLP().y FLP() en base al número de veces que se presenta cualquiera de estos comandos, también el renglón en que se apareció y el argumento que contiene dicha sentencia, pero sólo en los casos en que se presenten las sentencias SUB(), JLP() y FLP(). Esta función realiza la misma operación de generar el registro que la función **sintaxis3()**. La función **gen_hex_beg_end()**, le son transmitidos los siguientes parámetros:

1) LGEB2, Contiene el renglón donde empieza el programa principal.
2) LDNE2, Contiene el renglón donde termina el programa principal.
3) LINET, Es el nombre del archivo donde se guardara el archivo en hexadecimal.
4) BUS, Contiene el número de subrutinas que se encuentra en el programa.
5) LBUS, Contiene los renglones en que se encuentran las subrutinas .
6) LLBUS, Contiene el argumento de las subrutinas.
7) LTER, Contiene los renglones donde terminan las subrutinas.
8) PLJ, Contiene el número de bucles que se encuentran en el programa.
9) LPLJ, Contiene el renglón donde inicia el bucle.
10)LPLJ, Contiene el renglón donde termina el bucle.
11)LLPLF, Contiene el argumento de que define al número de bucles.

La función **gen_hex_beg_end()**, se auxilia de las funciones **dos_vel_ace()**, **dos_dos()**, **cero()**, **uno_vel_ace_esc()**, **uno()** y **uno_bin()**, las cuales se encargan de realizar la conversión de las sentencias en un dato en hexadecimal.

Esta función empieza la conversión en el renglón donde se encuentra la sentencia BEG() y termina en la sentencia END(). Para efectuar la conversión se copia el primer renglón en una cadena y esta se explora caracter por caracter, cuando encuentre el caracter que define al paréntesis que abre retrocede tres caracteres y estos los convierte en una cadena para después compararla con un arreglo de punteros que fue declarado al inicio e inicializado con todas las sentencias que se emplean para diseñar un programa.

Esta comparación da por resultado una posición en base a un valor entero el cual se compara con una lista de constantes numéricas y donde se encuentre una coincidencia se ejecuta la función respectiva que efectúe la conversión de la sentencia en un dato en hexadecimal. Si en el renglón existe más de una sentencia entonces se realiza la conversión de estas y después cambia al siguiente renglón.

Esta función, cuando aparezca la sentencia GSB(), toma el argumento de esta sentencia y determina a que sentencia SUB() de las que se encuentran en el programa se esta invocando, además se determina en que renglón empieza y en que renglón termina. Esto se lleva acabo mediante la conversión de los caracteres que se encuentran en el argumento en una cadena para después convertirlo en un valor entero, después se localiza la subrutina por medio del parámetro de la comparación del valor calculado y el parámetro que contiene los argumentos de las sentencias. En donde exista una coincidencia entonces la posición de ese elemento se tomará como base para encontrar el renglón de donde inicia y donde termina dicha subrutina. Para después almacenarlos en las variables **in** y **ou**.

Debido a que la función **gen_hex_beg_end()**, sólo realiza la conversión de los renglones que se encuentran en el intervalo definido por los parámetros **lbeg2** y **ldne2**. Es posible realizar operación de recursividad, es decir, se vuelve a llamar a la función, **gen_hex_beg_end()**, para sólo convertir las sentencias que se encuentran en la subrutina y después regresar a la siguiente sentencia que se encuentra en el programa principal, en su correspondiente dato en hexadecimal, con los mismos valores de los parámetros aunque en este caso sólo se sustituye el parámetro **lgeb2** por el valor de la variable **in** y el parámetro **ldle2** por el valor de la variable **ou**. Los parámetros que se transmiten cuando se realice la llamada a la función **gen_hex_beg_end()**, serán los mismos, tomando en cuenta la substitución del primero el segundo parámetro.

Se utilizo el mismo criterio de recursividad para diseñar la rutina que realiza la conversión de las sentencias hexadecimal que se encuentran dentro de la estructura de control o bucle. En la sentencia JLP(#,N), contiene dos parámetros el primero, #, es el que hace referencia al número del bucle y N, hace referencia al número de repeticiones que se deben realizar de las sentencias que se encuentran en el bucle. La sentencia con que se termina el bucle es FLP(#), y el parámetro # hace referencia a que número de bucle se refiere. Esto es importante debido a que en un bucle pueden contener otros bucles, a esto se le llama anidamiento de bucles.

La rutina que se emplea cada vez que se aparece la sentencia JLP(), debe almacenar los parámetros que se encuentran entre los paréntesis y que están separados por una coma, en las variables **p** y **p2**.

El valor de **p** se toma como base para localizar el inicio y el fin del bucle por medio de la comparación de los argumentos con esta variable. Al igual que en la rutina anterior cuando se localizan estos datos se guardan en las variables y que posteriormente se sustituye en los parámetros **lbeg2** y **ldne2**. El proceso de recursón se llevara acabo en la base al número de repeticiones que se encuentra especificado en la variable **p2**, esto quiere decir que se llama a la función **gen_hex_beg_end()**, tanta veces como lo indique la variable **p2**.

Para efectuar el almacenamiento de los datos codificados en hexadecimal, partiendo de las sentencias contenidas en el programa, es necesario llevar una secuencia lógica de almacenamiento, es decir, los datos deben estar separados por caracteres que indique a que dato se esta haciendo referencia.

Estos caracteres son: &, # y S. El caracter &, indica que el dato que le precede es el que hace referencia a alguna sentencia. El caracter #, indica que el dato que le precede corresponde a la parte entera de un número. El caracter S, indica que el dato que le precede corresponde a la parte decimal de un número. Para esto a cada sentencia le corresponde un valor entero que se le asigna cuando en la bifurcación SWITCH(), exista una coincidencia entre la lista de constantes numéricas que corresponde a cada sentencia. para después en base al argumento que le fue asignado a dicha sentencia ejecutar cualquiera de las sentencias: **des_vel_ace()**, **dos_dos()**, **dos()**, **cero()**, **uno_vel_ace_esc()**, **uno()** y **uno_bin ()**.

Todas estas funciones realizan la misma operación de conversión y almacenamiento, ya que están basados en el siguiente procedimiento. Primero convertir el valor asignado a esa sentencia en un valor en hexadecimal y almacenarlo conjuntamente con el caracter &. Como se muestra a continuación.

```
fprintf(fp,"& %02X",co);
```

Donde **co**, contiene el valor decimal de las sentencia. La cadena de control se encuentra definida por: "&%02X", y es en la que se especifica el formato en que se almacena los datos, para este caso el contenido de **co** se codifica en un valor en hexadecimal. Cuando la sentencia cuente con un argumento se le incluirá a la cadena el caracter #. Como se muestra a continuación.

```
fprintf(fp,"&%02x#",co);
```

Segundo examina el argumento de manera que se pueda estar seguro de que tipo de dato es, un valor entero positivo negativo, o un valor decimal positivo o negativo.

Esto se puede llevar a cabo debido a que en dicho argumento aparecen los caracteres punto y menos. Cuando el dato sea un entero positivo se convierte el argumento en un dato entero y se almacena en el archivo en hexadecimal. Como se muestra a continuación:

```
fprintf (fp,"%04x",p);
```

Donde **p**, contiene el dato del argumento. En este caso la cadena de control convertirá el valor de **p** en un valor hexadecimal empleando para esto cuatro caracteres debido a que se quiso tener un criterio general para todas las funciones dado que el dato máximo que se puede expresar es 20000 y que al convertirlo en hexadecimal 4E20. Se puede observar que se requieren cuatro caracteres para representar lo en ese sistema. Cuando el argumento cuente con una parte entera y una decimal se efectúa la conversión y se almacena primero la parte entera y después la parte decimal.

```
fprintf(fp,"%04X",p)
fprintf(fp,"%04XS",p)
```

Cuando el argumento sea negativo se ignora el signo menos y se toma sólo los dígitos, para después convertirlos en un valor entero, después se les suma la cantidad 32768 que en haxadecimal es 8000. Esto se lleva acabo debido a que el último bit hará referencia al signo de dicha cantidad.

Por ejemplo, si se desea almacenar el dato 3784 en decimal su correspondiente seria 0EC8 y en binario es 0000111011001000. El último bit es cero por lo tanto el dato es positivo. Pero si se desea almacenar el dato -3784 en decimal su correspondiente en hexadecimal seria 0EC8 más 8000 por lo tanto se almacenara el dato 8EC8 que en binario es 1000111011001000. Como el último bit es uno, el microcontrolador sabrá que el dato es negativo.

```
P=P+32768
fprinf (fp,"%04X",p);
```

Si el argumento cuenta con una parte entera y una parte decimal la cantidad de 32768 se le sumarán a los datos filtrados del argumento y después se almacenan. El microcontrolador debe reconocer los tres caracteres que anteriormente se mencionaron ya que al momento de ejecutar el programa le serán transmitidos y en base a esto se podrá ejecutar una tarea. Como el microcontrolador cuenta con un programa en el cual se encarga de recibir la información y manipular conforme a la lógica de su diseño de su programa. Esto quiere decir que cuando reciba el microcontrolador el caracter **&** y posteriormente identifique el tipo de tarea que deberá efectuar esta sabrá si tiene que esperar un parámetro o no y en que momento ejecutar dicha tarea.

La función **cero()**, se emplea para aquellas sentencias que no cuenten con un argumento. La función **uno()**, se emplea en aquellas sentencias que cuentan con un argumento compuesto por un número entero. La función **dos()**, se emplea en aquellas sentencias que cuentan con un argumento compuesto por dos números enteros. La función **uno_bin()**, se emplea para aquellas sentencias que cuanta con un argumento compuesto por un dato binario. La función **dos_dos()**, se emplea para aquellas sentencias que cuentan con un argumento compuesto por dos números enteros y estos se encuentren expresados en milímetros.

Debido a que el movimiento de la plataforma sólo se puede llevar acabo en base a pasos, esta función tendrá que efectuar la conversión de dicho número en pasos para después efectuar la conversión en hexadecimal. Para convertir el dato en pasos se efectúa por medio de la siguiente instrucción:

```
p=(p*motorxy*NVHFY)/DMX  /* Para el eje X*/
p=(p*motorxy*NVHFY)/DMY/*Para el eje Y*/
```

Donde motorxy contiene la resolución de los motores, NVHFX=86.11; DMX=435, NVHFY=41.675, DMY=215.

La función **uno_vel_ace_esc()**, se emplea para aquellas sentencias que cuenten con un argumento compuesto por un número decimal. La función **dos_vel_ace()**, se emplea para aquellas sentencias que cuenten con un argumento compuesto por dos números decimales.

II.2.3 Operaciones de entrada y salida

Las operaciones de entrada y salida están definidas por las funciones: **salvar_archivo()**, **leer_archivo()**, **cerrar_archivo()**, **nuevo_archivo()**.

La función **salvar_archivo()**, almacena el contenido del editor que se encuentra en la memoria, en un sistema de memoria intermedia. Esta función puede realizar dos diferentes tipos de operaciones de almacenamiento. La primera, se denomina sobreescribir, en la cual el almacenamiento se realiza a través de los cambios hechos a un archivo existente. El segundo método, se efectúa por medio de la creación de un archivo donde se ha de almacenar la información. La identificación del tipo de almacenamiento que debe efectuar se lleva a cabo empleando dos punteros tipo FILE y la función fopen().

Primero, se realiza la lectura de la cadena de caracteres la cuál define el nombre y la ruta de el archivo, después se abre el archivo para su lectura utilizando la función fopen(), esta función si retorna un NULL, entonces, el archivo no existe, y por lo tanto se debe abrir un archivo para su escritura, donde se ha de almacenar la información de el editor. Si por el contrario devuelve la función fopen() algo diferente a la macro NULL entonces se efectuará el proceso de sobreescribir. Antes de sobreescribir se despliega en la pantalla una ventana que le de al usuario la opción de poder modificar el nombre de el archivo al igual que la ruta.

```
void salvar_archivo()
{FILE *gen_archivo;
FILE *arch;
int nncd=0,j=0,elec1,romper=0;
char nonbre[]="paso002.xym";
int ncd2=0;
char opc;
int pinicial=0,pfinal=0,pcursor=0;
pinicial=inicial;pfinal=final;pcursor=cursor;
salva_video(5,12,47,18,nonbre);
marco(5,12,46,16,80,15,1,"Ruta Nombre y Extenciòn");
escribe_video(60,25,"OPCIÓN SALVAR      ",112,+128,4);
escribe_video(10,16," [ESC] Cerrar ",80,15,15);
do
 {nncd=lee_alfa(nv_arch,40,6,14,40,64);
  switch(nncd){
    case -2:romper=0;salvar=0;break;
    case -1:coloca_video(5,12,47,18,nonbre);
            romper=1;salvar=0;break;
    case 0:opc=nv_arch[0];
           ncd2=disco2(opc,0);
           if (ncd2==6)
             {ncd2=disco2(opc,0);}
              if(ncd2==0)
                {gen_archivo=fopen(nv_arch,"r");
                 if (gen_archivo==NULL)
                  {fclose(gen_archivo);
                   marco(5,12,46,16,112,0,1,"");
                   escribe_video(6,14,"          SALVANDO ",112,128,15);
                   arch=fopen(nv_arch,"w");
                   for (j=pinicial;j<=pfinal;j++)
                      {fprintf(arch,"%s\n",texto[j]);}
                       fclose(arch);
                       coloca_video(5,12,47,18,nonbre);
```

```
                romper=1;salvar=1;
                escribe_video(31,25,nv_arch,112,0,0);}
            else
             {fclose(gen_archivo);
              elec1=mensaje_salvar();
              switch(elec1){
              case 0:marco(5,12,46,16,112,0,1,"");
                     escribe_video(6,14,"      REMPLAZANDO ",112,128,15);
                     arch=fopen(nv_arch,"w");
                     for (j=inicial;j<=final;j++)
                         {fprintf(arch,"%s\n",texto[j]);}
                          fclose(arch);
                          coloca_video(5,12,47,18,nonbre);
                          escribe_video(31,25,nv_arch,112,0,0);
                          romper=1;salvar=1;} break;
              case 1:romper=0;break;
              case 2:coloca_video(5,12,47,18,nonbre);
                     romper=1;salvar=0;break;}}break;}
                      } while(romper==0);
escribe_video(60,25,"                ",112,0,0);
escribe_video(65,25,"Esc",112,4,4);
escribe_video(69,25,"Cerrar     ",112,0,0);}
```

La función **leer_archivo** (),abre un archivo para cargarlo en la memoria. Para esto es necesario, utilizar la función fopen() en la cual, después de que se lea la cadena de caracteres que define el nombre y la ruta de el archivo, se abre el archivo para su lectura, si retorna un NULL, entonces, el sistema indica que el archivo no existe. Si por el contrario retorna algo diferente entonces se inicia la lectura del archivo.

La lectura se lleva a cabo, primero identificando si el archivo pertenece a los creados por este sistema, mediante la comparación de la primera cadena de caracteres que se leyó con una cadena que se encuentra definida como el inicio de el archivo. Si no son iguales entonces el sistema desplegará un mensaje en la pantalla indicando que el archivo no pertenece a los creados por este sistema.

Si la cadena es igual entonces, se le asigna la suficiente memoria al Array de punteros para guardar dicha información, después salta tres caracteres y se realiza la lectura de los siguientes 50 caracteres que conforman la cadena del siguiente renglón y se compara esta cadena con otra cadena que especifica el fin de archivo, si no son iguales se le asigna al Array de punteros la memoria suficiente para almacenar la cadena, si por el contrario dicha cadena es igual entonces también se almacena la información y se termina la lectura del archivo.

```
void leer_archivo()
{FILE *gen_arch;
struct ffblk z;
char ch,opc;
int bas=0,pasos=0;
char paso[51]= "<<|                                                  ";
char paso1[51]="|>>                                                  ";
char basura[3];
int nncd=0,elec11,romper=0,ncd2;
register int done, atr=0,xy=5,yx=17,g=0;
int kiki=0;
char bw[40],bw2[40];
int i=0,pinicial=0,pfinal=0,pcursor=0,f=0,h=0,pf=0;
pinicial=inicial;pfinal=final;pcursor=cursor;
marco(5,12,46,16,80,15,1,"Ruta Nombre y Extenciòn");
escribe_video(25,16," [F6] Directorio ",80,15,15);
escribe_video(60,25,"OPCIÓN LEER          ",112,+128,4);
escribe_video(10,16," [ESC] Cerrar ",80,15,15);
pasos=0;
strcpy(linet2,linet3);
do
{nncd=lee_alfa(nv_arch,40,6,14,40,64);
 switch(nncd){
 case -3:opc=nv_arch[0];
         ncd2=disco2(opc,1);
         if (ncd2==6)
            {ncd2=disco2(opc,1);}
            if(ncd2==0)
              {pasos=1;g=0;
               marco(55,5,80,24,75,15,0,"");
               yx=7;xy=58;
               done=findfirst(nv_arch, &z, atr);
               while (!done)
                  {sprintf(bw,"%s",z.ff_name);
                  escribe_video(xy,yx,bw,80,15,15);
                  done = findnext(&z);
                  yx++;
                  if (yx==23)
                    {escribe_video(72,24," MÁS[\x19]",80,15,15);
                     escribe_video(57,24,"[ESC]Regresar ",80,15,15);
                     for(;;)
                       {kiki=lee_dato_simple(75,24);
                        if(kiki==2)
                          {break;}
```

```
                        if(kiki==-1)
                          {break;}}
                          switch(kiki)
                          {case -1:g=1;break;
                           case 2:g=0;yx=7;marco(55,5,80,24,75,15,0,"");break;}
                                  if(g==1)
                                    {break;}}}
                           romper=0;}
                          else
                            {ncd2=20;romper=0;} break;
case -2:if (pasos==1)
        {marco(55,5,80,24,64,15,0,"");}
         pasos=0;romper=0;break;
case -1:if (pasos==1)
       {marco(55,5,80,24,64,15,0,"");}
       char_simple(5,12,47,18,255,4,64);
       pasos=0;romper=1;existe=0;break;
case 0:opc=nv_arch[0];
       ncd2=disco2(opc,1);
       if (ncd2==6)
         {ncd2=disco2(opc,1);}
          if(ncd2==0)
           {if (pasos==1)
              {marco(55,5,80,24,64,15,0,"");pasos=0;}
               pasos=0;gen_arch=fopen(nv_arch,"r");
               if (gen_arch==NULL)
                  {elec11=mensaje_leer();
                   switch(elec11)
                   {case 0:romper=0;break;
                    case 1:fclose(gen_arch);
                           char_simple(5,12,47,18,255,4,64);
                           romper=1;existe=0;salvar=0;break;}}
             else
             { h=0;texto[h]=(unsigned char *)farmalloc(sizeof(unsigned char)*51);
               fgets(texto[h],MAXLONG,gen_arch);
               fgets(basura,3,gen_arch);
               if(strcmp(paso1,texto[h])==0)
                {marco(5,12,46,16,112,0,1,"");
                 escribe_video(6,14,"              LEYENDO ",112,128,15);
                 f=0;h=1;
                for(;;)
                 {texto[h]=(unsigned char *)farmalloc(sizeof(unsigned char)*51);
                    fgets(texto[h],MAXLONG,gen_arch);
                    if(strcmp(paso,texto[h])==0)
                      {f=h;break;}
```

```
                    else
                     {fgets(basura,3,gen_arch);h++;}}
                      fclose(gen_arch);
                      marco(1,5,54,24,64,15,0,"[ EDITOR ]");
                      escribe_video(54,21,"\x1E",112,4,4);
                      escribe_video(54,22,"_",112,4,4);
                      escribe_video(54,23,"\x1F",112,4,4);
                      escribe_video(37,24,"\x11",112,4,4);
                      escribe_video(38,24,"_",112,4,4);
                      escribe_video(39,24,"\x10",112,4,4);
                      inicial=0;final=f;cursor=1;
                      if(final==MAXRENG)
                       {pf=MAXRENG;}
                      else
                       {if(final>MAXRENG)
                          {pf=MAXRENG;}
                           else
                             {pf=final;}}
                  for(i=0;i<=pf;i++)
                 {escribe_video(3,i+MÁS,texto[i],cefondo3_3,ctexto3,ctexto3);}
                    romper=1;existe=1;compilar=1;salvar=1;apuntador=1;
                    escribe_video(31,25,nv_arch,112,0,0);editar_archivo();}
              else
               { fclose(gen_arch);
                farfree((void *)texto[h]);
                h=0;bas=mensaje_leer2();
                char_simple(5,12,47,18,255,4,64);
                romper=1;existe=0;salvar=0;apuntador=1;}}}break;}
  }while(romper==0);
```

La función **cerrar_archivo()**, se encarga de liberar el contenido de la memoria para dejarla disponible para alguna otra tarea.

```
void cerrar_archivo()
{int j=0;
int pinicial=0,pfinal=0,pcursor=0;
pinicial=inicial;pfinal=final;pcursor=cursor;
escribe_video(60,25,"OPCIÓN CERRAR          ",112,+128,4);
for (j=pinicial;j<=pfinal;j++)
  {farfree((void *)texto[j]);}
  inicial=0;final=0;cursor=0,apuntador=0;
  existe=0;salvar=0;
  escribe_video(60,25,"                    ",112,0,0);
  strcpy(nv_arch,"NINGUNO.DAT                          ");
  escribe_video(31,25,nv_arch,112,0,0);
```

```
escribe_video(65,25,"Esc",112,4,4);
escribe_video(69,25,"Cerrar     ",112,0,0);
marco(1,5,54,24,64,15,0,"[ EDITOR ]");
compilar=0;}
```

La función **nuevo_archivo()**, se encarga de asignar la memoria suficiente a tres renglones, los cuales definen el primero y el tercero como inicio y fin del archivo, dejando el segundo como línea de texto para el programa.

```
void nuevo_archivo()
{int pinicial=inicial,pfinal=final,pcursor=cursor,papuntador=apuntador,j;
pinicial=0,pfinal=2,pcursor=1,papuntador=1;
texto[pinicial]=(unsigned char *)farmalloc(sizeof(unsigned char)*51);
texto[pfinal]=(unsigned char *)farmalloc(sizeof(unsigned char)*51);
texto[pcursor]=(unsigned char *)farmalloc(sizeof(unsigned char)*51);
strcpy(texto[pinicial],"|>>                                    ");
strcpy(texto[pcursor],"                                        ");
strcpy(texto[pfinal],"<<|                                      ");
marco(1,5,54,24,64,15,0,"[ EDITOR ]");
escribe_video(54,21,"\x1E",112,4,4);
escribe_video(54,22,"_",112,4,4);
escribe_video(54,23,"\x1F",112,4,4);
escribe_video(37,24,"\x11",112,4,4);
escribe_video(38,24,"_",112,4,4);
escribe_video(39,24,"\x10",112,4,4);
escribe_video(3,0+MÁS,texto[pinicial],ccfondo3_3,ctexto3,ctexto3);
escribe_video(3,1+MÁS,texto[pcursor],ccfondo3_3,ctexto3,ctexto3);
escribe_video(3,2+MÁS,texto[pfinal],ccfondo3_3,ctexto3,ctexto3);
inicial=pinicial,final=pfinal,cursor=pcursor,apuntador=papuntador;
existe=1;salvar=1;compilar=0;
escribe_video(60,25,"OPCIÓN NUEVO     ",112,+128,12);
escribe_video(60,25,"                 ",112,0,0);
strcpy(nv_arch,"NINGUNO.DAT                    ");
escribe_video(31,25,nv_arch,112,0,0);
escribe_video(65,25,"Esc",112,4,4);
escribe_video(69,25,"Cerrar     ",112,0,0);
```

La función **disco2()** en la modalidad **Maquinado de Piezas** y **disco()** en la modalidad **Perforado de Tarjetas**, tiene la tarea de detectar la presencia de algún error en la unidad de disco, esto se lleva acabo mediante el empleo de la función de biblioteca de Turbo "C", **biosdisk()**. Esta función da por resultado una serie de valores que corresponden a los diferentes tipos de errores que se pueden presentar, entre los cuales se encuentran los siguientes:

VALOR EN HEX	VALOR EN DEC	DESCRIPCIÓN
03H	3	DISCO PROTEGIDO
20H	32	FALLA EN LA UNIDAD
80H	128	NO HAY RESPUESTA

Por lo tanto cuando esta función entregue cualquiera de estos resultados debe mostrar en la pantalla un mensaje de error. Si por el contrario entrega un resultado diferente se continua con el proceso de lectura o escritura.

Esta función hace referencia a cualquiera de las dos unidades disponibles por medio de uno de sus parámetros. Para esto es necesario contar con el primer caracter de la cadena que contiene la ruta, el nombre y la extensión del archivo, ya que el primer caracter es el que especifica la unidad a la cual se desea realizar alguna operación. Las unidades a las cuales se puede accesar son: a: y b:, por lo consiguiente si en la ruta se especifica cualquier otra unidad, simplemente se da por terminada esta ejecución.

```
disco2(char opc,int que)
{ int ncd2=0,result,disco;
 char buffer[512].unidad='f';
switch(opc)
       {case 'a':disco=0;  unidad='A';break;
        case 'b':disco=1;  unidad='B';break;
        default :unidad='C';break;}
        if (unidad=='C')
         {ncd2=0;
          return(ncd2);}
         else
         {result=biosdisk(4,disco,0,0,0,1,buffer);
         if(result==6)
          {return (ncd2=6);}
         if (result==128||result==3)
           {marco(5,12,46,16,112,15,15,"MENSAJE DEL SISTEMA");
            escribe_video(8,13,"ERROR DE LECTURA EN EL DRIVE",112,0);
            gotoxy(38,13);
            printf("%c:",unidad);
            escribe_video(30,15,"<<REGRESAR>>",32,15);
            bioskey(0);
            marco(5,12,46,16,80,15,1,"Ruta Nombre y Extenciòn");
            if(que==1)
            {escribe_video(25,16," [F6] Directorio ",80,15,15);}
            escribe_video(10,16," [ESC] Cerrar ",80,15,15);
            ncd2=20;
```

```
return(ncd2);}
else
{ncd2=0;
 return(ncd2);}}}
```

II.2.4 Ejecución en forma Automática

Para llevar acabo la ejecución en forma Automática, se emplea la función **eject_auto()**, esta función lee el contenido del archivo que se creo al momento de compilar el programa. Esta función efectúa la lectura del archivo caracter por caracter y la compara con los caracteres **&**, **#**, **S**, que se encuentran dentro de la bifurcación SWITCH(), cuando exista una coincidencia ejecuta el bloque de instrucciones que se encuentren asociados a dicho caracter.

Cuando el dato leído corresponda con el caracter **&**, se procede a leer los dos siguientes caracteres para después convertirlos en una cadena la cual será transmitida por la función **enviar_caracter_arg_auto2()**, la cual efectúa las operaciones necesarias para enviar este dato al microcontrolador, el cual a su vez retorna una respuesta que le indique la función deberá efectuar. La respuesta que le transmite el microcontrolador puede ser cualquiera de los siguientes caracteres:

1) **z**: Este caracter que indica que se originó un corto en la fuente.
2) **>** :Este caracter que indica que el microcontrolador transmitirá información al sistema.
3) **?**: Este caracter que indica que puede enviar la siguiente información.
4) **@**: Este caracter que indica que se ejecutará una tarea.
5) **L**: Este caracter que indica que se suspenda la ejecución en modo Automático.
6) **F**: Este caracter que indica que ocurrió algún error.

En este bloque de instrucciones también se efectúa la comparación del tipo de sentencia que se envió, contra una serie de sentencias que corresponde a las que pueden efectuar un desplazamiento y que permiten ejecutar una pausa y por lo consiguiente un continuar.

Si estas sentencias no corresponde con ninguna que pueda efectuar un movimiento, es posible que sólo realice una operación de almacenamiento de datos, o bien que efectúe una operación de perforado en la cual seria imposible realizar alguna pausa y por lo consiguiente el sistema tendrá que mantenerse en espera del caracter que indique que puede continuar con la transmisión de información.

En el caso de que la lectura de los caracteres se presente el caracter #, la función tendrá que leer, los siguientes cuatro caracteres para convertirlos en una cadena y luego serán transmitidos por la función **envia_caracter_arg_auto4()**, la cual efectúa las operaciones necesarias para transmitir el dato al microcontrolador el cual retorna cualquiera de los caracteres que se mencionaron con anterioridad.

Por último cuando se lea el caracter **S** se recibirá la misma operación que se efectúa cuando se presenta el caracter #. Cuando se ejecute alguna tarea de movimiento y se efectúe una pausa el sistema se detendrá debido a que se accesara a un bucle infinito el cual se podrá romper cuando presione el pushbutton que se encuentra en el Rac de control denominado **R-IN**, para lo cual el microcontrolador transmitirá el caracter de continuar. Este tipo de caracteres se aplicará más adelante, así como también el modo de operación de las funciones **envia_caracter_arg_auto2()** y **envia_araeter_arg_auto4()**.

II.2.5 Ejecución en forma Interactiva

El método Interactivo se emplean las funciones: **ejecutar_pasos()**, **gen_hex_interactivo()**, **gen_hex2()** y **eject_auto()**.

Debido a que en esta modalidad la secuencia de ejecución de las sentencias no llevan un orden, la secuencia de ejecución queda a cargo del usuario el cual podrá elegir cualquiera de las sentencia del programa y ejecutarla por medio de un Enter. Para esto fue necesario tomar la parte de la función **editar_archivo()**, pero sólo aquellas rutinas que desplazan el cursor y sólo se modifico la rutina que reconoce el Enter, para que cuando se presione esta tecla se ejecute la función **gen_hex2()** y después **eject_auto()**.

Las rutinas que desplazan el cursor se emplean para desplazarse a las sentencias que se desean ejecutar. No es necesario explicar el modo de operación de la función **ejecutar_pasos()**, ya que tiene la misma lógica de diseño que se empleó en la función **editar_arcjhivo()**.

la función **gen_hex2()**, realiza la misma operación que la función **gen_hex()**, aunque en este caso la extensión del archivo será HDC. Después ejecutará la función **gen_hex_interactivio()**. La función **gen_hex_interactivio()**, tiene la tarea de efectuar la conversión de las sentencias que se encuentran en el renglón en un dato en hexadecimal.

Para esto se obtuvo una copia a la función **gen_hex_beg_end()** con el nombre **gen_hex_interactivio()** y sólo se elimino la estructura FOR en la que se especifica el intervalo de los renglones que se codificarán ya que en este caso no se emplearan los parámetros **lgeb2, ldne2** y mucho menos los demás parámetros que llevan el registro de las sentencias JLP(), FLP(), RTE() y GSB().

En esta sentencia los únicos parámetros que necesita es la cadena que especifica la ruta, el nombre y la extensión del archivo. así como la variable que contiene el renglón que se desea ejecutar. Debido a que no se pueden ejecutar cualquiera de estas sentencias, cuando se elija alguna el sistema simplemente lo debe ignorar. Por último se emplea nuevamente la función **eject_auto()**, debido a que esta función cuando realiza la lectura del archivo caracter por caracter va revisando el fin de archivo cuando lo encuentra termina la ejecución de esta función. Por lo tanto puede ser empleada esta función para ejecutar cualquiera de las dos formas, **Automática** o **Interactiva**. Cabe recalcar que cuando se ejecute la modalidad Interactiva antes se debió compilar el archivo en su totalidad. A continuación se muestra la Figura 2-4, la cual muestra la forma en que se guarda un programa en hexadecimal.

```
SUB(4);
ANG(11.0,12.0);
RTE();          &22&17&04#033F#06D1&01#000A$0000#000A$0000&01#000
                1$0000#0003$0000
SUB(10);        &1F&1F&02#0000$000A#0002$0190&1F&14
ALN(0.01,2.40); &01#000B$0000#000C$0000&04#018C#0184&01#0001$0000#0
                003$0000
ATL();          &1F&1F&02#0000$000A#0002$0190&1F&14
NOP();          &01#000B$0000#000C$0000&04#018C#0184&01#0001$0000#0
GSB(4);         003$0000
RTE();          &1F&1F&02#0000$000A#0002$0190&1F&14
                &01#000B$0000#000C$0000&04#018C#0184&01#0001$0000#0
BEG();          003$0000
EFT();RLN(21,45); &1F&1F&02#0000$000A#0002$0190&1F&14
ANG(10.0,10.0); &01#000B$0000#000C$0000&04#018C#0184&01#0001$0000#0
JLP(12,5);      003$0000
ANG(1,3);       &1F&1F&02#0000$000A#0002$0190&1F&14
JLP(3,2);       &01#000B$0000#000C$0000&04#018C#0184
ATL();          &18&23
FLP(3);
GSB(10);
RLN(10,10);
FLP(12);
AFT();
END();
```

Figura 2-4 . El programa se muestra en el lado izquierdo y su codificación en el lado derecho.

II.2.6 Ejecución en forma Manual.

En esta modalidad al igual que en las otras dos modalidades se emplean las marcos DMX, DMY, NUHFX, NVHFY, con sus respectivos valores:

```
# define DMX 435 /* Distancia máxima en milímetros para el eje X */
# define DMY 215 /* Distancia máxima en milímetros para el eje Y*/
# define NVHFX 86.11 /*Número de vueltas de inicio a fin en el eje X */
# define NVHFY 41.675 /*Número de vueltas de inicio a fin en el eje Y */
```

También se emplea la variable **mot** la cual guarda la resolución del motor y que se utiliza para calcular el máximo número de pasos que se deben efectuar para desplazar tanto al eje X como al eje Y, de inicio a fin. También se calcula la distancia en milímetro que se desplaza la plataforma cuando esta se mueve un paso en ambos ejes. Los cálculos se realizan:

```
nmpasosx=ceil (mot*NVHFX);
nmpasosy=ceil (mot*NVHFY);
sprintf(dfx,"% f",(float) (DMX/nmpasos x));
upx=atof(dfx);
sprintf(dfx,"%f",(float)(DMY) numpasos y ));
upy=atof(dfx);
```

En esta modalidad se emplean las funciones, **manual()** y **protocolo()**. La función **manual()**.esta construida por una estructura DO-WHILE() la cual cuenta en su bloque de sentencias con una bifurcación múltiple llamada SWITCH(). La pulsación de una o dos teclas da por resultado la asignación de un valor a una variable la cual se compara a una lista de constantes u opciones contenidas en la bifurcación y cada una realiza una tarea de asignación o de ejecución.

Las operaciones de ejecución pueden efectuar las tareas de desplazamiento de la plataforma X-Y, activación o desactivación de la fuente que se encuentra en el Rac de potencia y que se emplea para suministrar voltaje a los motores y por último la tarea de perforación.

Las opciones de desplazamiento se ejecutan cuando se presionan conjuntamente cualquiera de las teclas que se especifican a continuación:

a) ←,→,↑,↓ Desplazamiento según la dirección
b) ALT D Desplázate al punto

En esta modalidad no es necesario definir algún desplazamiento negativo o positivo ya que sólo con presionar cualquiera de las flechas de desplazamiento el microcontrolador sabe en que sentido debe girar su eje.

Para cada opción existe una rutina, aunque en algunos casos son similares, ya que sólo cambian en las condiciones de operación. Antes de efectuar algún desplazamiento se debe comprobar que la fuente se encuentre encendida, ya que en caso contrario no se efectuaría dicha tarea y desplegará en la pantalla el porque no se puede realizar dicha tarea. Después verifica que el movimiento que se requiere sea posible llevarlo acabo, para esto se toma en cuenta la posición que guarda la plataforma al inicio de cualquier modalidad. Como se muestra en la Figura 2-5.

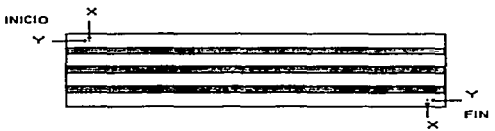


Figura 2.5. Inicio y fin de la plataforma.

La figura anterior se puede observar que cuando este en el inicio la plataforma no se podrá efectuar un movimiento hacia la izquierda o hacia la arriba.

En el caso en que la plataforma se encuentre en el final, no se podrá realizar un desplazamiento hacia la derecha o hacia abajo, esto debido a que el sistema cuenta con una variable que lleva el registro del número de pasos que se han de efectuar, y lo compara con la variable que contiene el máximo número de pasos permitidos en cualquiera de los ejes X o Y. Estas rutinas emplean las variables **aa1** y **ff1**, las cuales llevan el conteo del número de pasos que se han realizado tanto en el eje X como en el eje Y respectivamente, por lo tanto las variables **aa1** y **ff1** no pueden ser menores a cero ni mayores que las variables **nmpasosx** y **nmpasosy** respectivamente. Cuando las condiciones anteriores se hayan superado entonces se entabla la comunicación entre el microcontrolador y la rutina de desplazamiento para que se efectúe dicha tarea. Estas rutinas cuentan además con una rutina integrada de **Pausa**, la cual se puede activar presionando la tecla de función **F7** o bien por medio del interruptor que se encuentra en el Rac de control.

Cuando se efectúa una **Pausa** la rutina propone dos opciones: **Cancelar** y **Continuar**. Si se elige la opción **Cancelar**, se suspende la tarea de desplazamiento, pero el microcontrolador debe proporcionar al sistema las coordenadas en las cuales se encuentra la plataforma para que actualicen sus datos en la pantalla así como en las variables.

Si por el contrario se elige la opción **Continuar**, entonces se continua con el desplazamiento de la plataforma hasta que llegue al punto elegido. La rutina de **Pausa** esta construida mediante un bucle infinito, esto quiere decir que cuando se ingrese a dicho bucle la única salida será por medio de la elección de cualquiera de las dos opciones anteriores. Recordemos que la opción **Continuar** puede ejectuar pulsando el botón **R-IN** que se encuentran en el Rac de control.

Esta rutina desplaza la plataforma el número de pasos que especifique las variables **pasosx** y **pasosy**, es decir al presionar la flecha de desplazamiento a la derecha a la variable **aa1** se le incrementará el valor de **pasosx**. Si por contrario se presiona la flecha a la izquierda se le incrementará a la variable **aa1** el valor de **pasosx**. Esto también sucede cuando se empleen a las otras dos teclas de desplazamiento. En la rutina desplázate al punto, lo único que verifica, es que la fuente este encendida ya que para desplazarse a un punto, previamente se debió haber estado en el. Esta rutina no desplaza la plataforma de una cantidad de pasos especificadas en las variables **pasosx** y pasosy si no que se desplaza a una coordenada que anteriormente fue grabada, la rutina de grabar un punto se explica más adelante.

La otra opción de ejecución se emplea para indicarle al microcontrolador que encienda o apague la fuente. En esta rutina se emplea una variable **FTE** la cual cambia su valor de 0 a 1 cuando se enciende y de 1 a 0 cuando se ejecute el proceso de apagar la fuente. Esta poción se activa mediante la pulsación conjunta de las teclas **ALT-F**. La última opción se emplea para efectuar una perforación, esta rutina se activa presionando las teclas **ALT-P**. Esta rutina le envía al microcontrolador la orden de perforar y cuando esta concluya el microcontrolador debe informar a la rutina que la tarea fue efectuada satisfactoriamente. Dentro de la función **manual()**, las rutinas de asignación tiene como tarea de modificar las condiciones de operación en dicha modalidad. Esta tipo de operaciones son:

a) La rutina de regresar al punto (ALT-R).
b) La rutina de cambiar al punto (ALT-C).
c) La rutina de grabar un punto (ALT-G).
d) La rutina que modifica el número de pasos en X (ALT-X).
e) La rutina que modifica el número de pasos en Y (ALT-Y).
f) La rutina que modifica la velocidad lineal X (ALT-L).
g) La rutina que modifica la velocidad lineal Y (ALT-V).
h) La rutina que modifica la aceleración lineal en X (ALT-N).
i) La rutina que modifica la aceleración lineal en Y (ALT-A).
j) La rutina que transmite un dato (ALT-0).
k) La rutina que modifica la resolución del motor (ALT-M).

La rutina que graba un punto, lo que en realidad realiza, es almacenar la coordenada en la cual se encuentra la plataforma en dos arreglos **gppx** y **gppy**. Estos arreglos pueden almacenar como máximo mil puntos. Par grabar un punto se utiliza un variable punto como subíndice de los arreglos y que determinarán en que celda se han de grabar dichos datos. Cuando se elija esta opción y se almacenan dichos datos en los arreglos, al microcontrolador sólo se le debe enviar el valor de la variable **punto** para que guarde en algún registro el valor de la coordenada que le llegó, y que corresponde a la posición que guarda en ese momento la plataforma, ya que también el microcontrolador lleva el conteo del número de pasos efectuados tanto en el eje X como en el eje Y.

La rutina que cambia de punto sólo incrementa en una unidad el valor del subíndice de la variable **punto**. La rutina que regresa al punto decrementa en una unidad el valor del subíndice de la variable **punto**. Las rutinas que modifican el número de pasos en X y en Y, se emplean para almacenar el número de pasos que se efectuaran cuando se presione cualquier tecla de desplazamiento (←,→,↓,↑).

Estas rutinas antes de enviar al microcontrolador algún dato, primero deben verificar que el dato a transmitir no sobre pase el número el valor de **pasosx** o **NMPASOSY**, si esto ocurriera el dato que se escribió se ajustará al valor de cualquiera de las variables **NMPASOSX** o **NMPASOSY**, esto depende de que eje se requiere modificar.

Las rutinas que modifican la velocidad y aceleración lineal en los ejes X y Y tienen las mismas características de las rutinas anteriores, aunque en estos casos se deben ajustar los valores a un intervalo de operación, es decir, el dato que se escriba para modificar la **Velocidad** lineal no debe de ser menor que 0.001 ni mayor que 20.00. Así como también el dato para la **Aceleración** no debe de ser menor que 0.001 ni mayor que 999.999.

En la rutina que transmite un dato binario, solamente se escribe el dato en forma binaria y que se le envía al microcontrolador. En la rutina que modifica la resolución del motor, primero, verifica que dicha resolución no sobre pase la máxima resolución estipulada que en este caso es de 20000 pasos por revolución. Después se debe inicializar todo el sistema ya que al modificar la resolución las condiciones cambian en base al número máximo de pasos.

Dentro de la estructura DO-WHILE se emplea una estructura WHILE la cual revisa constantemente el puerto serie y el puerto paralelo asignado para el Mouse. El puerto del Mouse se revisa por que también se puede elegir una opción por medio de este periférico. El puerto serie se revisa ya que por medio de este se efectúa la comunicación entre el microcontrolador y el sistema. Por el puerto serie, el microcontrolador puede enviarle al sistema la siguiente información:

a) Comunicación establecida.
b) Solicitud para transmitir un dato.
c) Suspención.
d) Fuente en corto.
e) Error de comunicación.

En esta modalidad al igual que en las otras dos es necesario que el microcontrolador envíe al sistema un dato que indique que se encuentra funcionando y que la comunicación se encuentra establecida. Cuando el microcontrolador le envía el dato al sistema que especifica que va ha transmitir un dato y el lo acepta, llama a la función **protocolo()** la cual es la que se encarga de recibir todos los datos que envía el microcontrolador. En esta rutina sólo se puede recibir un dato que modifique algún valor tanto en las variables como en la pantalla, si el microcontrolador necesita transmitir más de un dato será necesario solicitar la transmisión varias veces. En esta función la estructura principal es la bifurcación SWITCH(). En base al dato que reciba elige alguna de las siguientes opciones:

a) Cambia la posición de inicio en el eje X.
b) Cambia la posición de fin en el eje X.
c) Cambia la posición de inicio en el eje Y.
d) Cambia la posición de fin en el eje Y.
e) Cambia el parámetro del codificador en X.
f) Cambia el parámetro del codificador en Y.
g) Modifica el dato de salida.
h) Cambia la coordenada en X y Y.

Las primeras son empleadas para indicarle al sistema y al usuario cuando un eje se encuentra en el inicio o en el fin del recorrido, es decir, si el microcontrolador transmite un cero en la posición del inicio del eje y entonces la plataforma se encuentra fuera del origen. Cuando se transmita un 1 entonces indica que se encuentra en el origen. El mismo criterio es para las demás operaciones. La suspención, indica que en el Rac de control se presiono el interruptor Reset, y por lo tanto se debe salir de esa modalidad y debe regresar al menú anterior.

Cuando se apague la fuente que suministra voltaje a los motores paso a paso entonces cualquier tarea subsecuente se debe ignorar hasta que la falla se haya corregido, o simplemente se suspenda la ejecución de esta modalidad. En esta rutina se ingresa a la función **fte_corto()**, y esta debe permanecer ahí hasta que retorne el caracter que especifica que la comunicación se a restablecido.

Por último cuando envíe un caracter que no se encuentra definido en el sistema el caracter de error, entonces el sistema realizará el mismo proceso que la función **fte_corto()**, se detendrá ya que esta indica que la comunicación se rompió por alguna eventualidad.

II.4 Comunicación

La comunicación entre el microcontrolador y la PC queda a cargo del protocolo de comunicación, el cual se diseño mediante un juego de reglas y convenciones que regulan la transmisión de datos.

El protocolo de comunicación esta formado por una serie de funciones que realizan el intercambio de caracteres de un modo uniforme de manera que cuando se transmita un caracter (8 bits a la vez) debe esperar por parte del receptor una respuesta que indique que llego dicho caracter, antes de enviar el siguiente.

Las funciones se dividen en dos modos de operación, transmisión y recepción. Estas funciones cuentan con una rutina de tiempo de espera respuesta si en ese lapso de tiempo no llega la respuesta esperada, se asume que ocurrió un error de comunicación o de ejecución, esto en base al tipo de operación que se haya estado realizando.

Cuando se transmite una cantidad de la PC al microcontrolador, se debe efectuar la conversión de dicha cantidad en su correspondiente en hexadecimal, después se separa dicho resultado en parejas de caracteres, las cuales se codifican en correspondiente en decimal dando por resultado un valor comprendido entre 0 y 255 que corresponderá a un caracter del código ASCII, estos datos son los que se transmitirán al microcontrolador.

El procedimiento de transmisión del microcontrolador a la PC es muy similar al que se acaba de describir con anterioridad, aunque en este caso cuando se hayan recibido los caracteres enviados por el microcontrolador, empleando la función de biblioteca de Turbo C, **sprintf()**, se realizara la conversión de dichos caracteres en su correspondiente en hexadecimal de manera global, para después realizar la conversión de dicho resultado en decimal. En la Tabla 2.2. Se muestra conjunto de caracteres de control que corresponden a las sentencias empleadas para el diseño de programas.

DEC	HEX	INSTRUCCIÓN	DESCRIPCIÓN
1	01	ANG	Caracter de aceleración angular
2	02	ALN	Caracter de aceleración lineal
3	03	RNG	Caracter de recorrido angular
4	04	RLN	Caracter de recorrido lineal
5	05	MVA	Caracter de movimiento absoluto
6	06	MVR	Caracter de movimiento relativo
7	07	RMV	Caracter que repite movimiento
8	08	CDI	Caracter que cambia la dirección en X y en Y
9	09	CDX	Caracter que cambia la dirección en X
10	0A	CDY	Caracter que cambia la dirección en Y
11	0B	AHF	Caracter de aceleración de inicio a fin
12	0C	VHF	Caracter de velocidad de inicio a fin
13	0D	GHM	Caracter de ve ha inicio
14	0E	GEN	Caracter de ve ha fin
15	0F	VAN	Caracter de velocidad angular
16	10	VLN	Caracter de velocidad lineal
17	11	GAN	Caracter que ejecuta el movimiento angular
18	12	GLN	Caracter que ejecuta el movimiento lineal
19	13	MAN	Caracter que ejecuta movimiento angular
20	14	NOP	Caracter de no operación
21	15	RMD	Caracter de resolución del motor
22	16	RIP	Caracter que inicializa el proceso
23	17	EFT	Caracter que enciende fuente
24	18	AFT	Caracter que apaga fuente
25	19	LBF	Caracter que limpia el buffer
26	1A	ILX	Caracter que ejecuta movimiento lineal en X con incrementos
27	1B	ILY	Caracter que ejecuta movimiento lineal en Y con incrementos
28	1C	ITL	Caracter de interrupción total
29	1D	IPS	Caracter de pausa
30	1E	CNP	Caracter de continua
31	1F	ATL	Caracter que activa el taladro
32	20	IPX	Caracter ejecuta movimiento en X, en pasos con incrementos
33	21	IPY	Caracter ejecuta movimiento en Y en pasos con incrementos
34	22	BEG	Caracter de Inicio del programa
35	23	END	Caracter de Fin del programa
37	25	PEA	Caracter de posición del codificador en forma absoluta en X y Y
38	26	PER	Caracter de posición del codificador en forma relativa en X y Y
39	27	INP	Caracter de entrada binaria
40	28	OUT	Caracter de salida binaria
41	29	LHX	Caracter de limite inicio en X
42	2A	LHY	Caracter de limite inicio en Y
43	2B	LFX	Caracter de limite fin en X
44	2C	LFY	Caracter de Limite fin en Y
45	2D	EPP	Caracter que envía posición en pasos de X y Y
46	2E	WIN	Caracter de espera una entrada

Tabla 2.2

DEC	HEX	INSTRUCCIÓN	DESCRIPCIÓN
48	30	MPT	Caracter de muévete al punto
49	31	MAP	Caracter de muévete a través del punto
50	32	PNT	Caracter de punto
51	33	DLY	Caracter de retardo
52	34	ESC	Caracter de escala
53	35	ATC	Caracter que activa el taladro continuamente
54	36	PTL	Caracter que apaga el taladro
55	37	CMX	Caracter que calcula movimiento X en base a velocidad, aceleración y posición
56	38	CMY	Caracter que calcula movimiento Y en base a velocidad, aceleración y posición
57	39	MVL	Caracter que ejecuta movimiento lineal

Continuación de la Tabla 2.2.

En la Tabla 2.3. Se muestran los caracteresde control empleados para la transmisión de datos entre el microcontrolador y la PC y viceversa.

DEC	HEX	CAR	DESCRIPCIÓN
75	4B	K	Llego el dato transmite el siguiente byte
79	4F	O	Llego el dato transmite el siguiente byte
82	52	R	Llego el dato transmite el siguiente byte
83	53	S	Llego el dato transmite el siguiente dato
81	51	Q	Llego el dato transmite el siguiente byte
84	54	T	Llego el dato transmite el siguiente byte

Tabla 2.3.

En la Tabla 2.4. Se muestran los caracteresde control que pueden ser transmitidos por la PC para solicitar al microcontrolador la ejecución de alguna tarea.

DEC	HEX	CAR	DESCRIPCIÓN
21	15	§	Caracter de resolución del motor
23	17	↨	Caracter que enciende fuente
24	18	↑	Caracter que apaga fuente
29	1D	↔	Caracter de pausa
30	1E	▲	Caracter de continua
31	1F	▼	Caracter que activa el taladro
40	28	(	Caracter de salida binaria
48	30	0	Caracter de muévete al punto
58	3A	:	Caracter de ejecución en el modo Manual
59	3B	;	Caracter de ejecución en el modo Automático
60	3C	<	Caracter de ejecución en el modo Interactivo
61	3D	=	Caracter de ejecución en el modo Perforado de tarjetas
63	3F	?	Listo para transmitir o recibir/Entabla comunicación
66	42	B	Caracter que desplaza el eje X a la derecha
67	43	C	Caracter que desplaza el eje Y hacia abajo
68	44	D	Caracter que desplaza el eje Y hacia arriba
69	45	E	Caracter que desplaza el eje X a la izquierda
71	47	G	Caracter que graba el punto
72	48	H	Caracter de velocidad lineal en X

Tabla 2.4.

DEC	HEX	CAR	DESCRIPCIÓN
73	49	I	Caracter de aceleración lineal en Y
74	4A	J	Caracter que cambia los incrementos del eje X
75	4B	K	Caracter que cambia los incrementos del eje Y
76	4C	L	Caracter para salir del modo
78	4E	N	Caracter que pide restablecer la comunicación
85	55	U	Caracter que inicializa el proceso
87	57	W	Caracter de espera
89	59	Y	Caracter de aceleración lineal en Y
90	5A	Z	Caracter de velocidad lineal en X
106	6A	j	Caracter del parámetro en X
107	6B	k	Caracter del parámetro en Y
123	7B	{	Caracter de cancelar la operación

Continuación de la Tabla 2.4.

En la Tabla 2.5. Se muestran los caracteresde control que pueden ser transmitidos por el microcontrolador a la PC para solicitar la ejecución de alguna tarea.

DEC	HEX	CAR	DESCRIPCIÓN
62	3E	>	Caracter que solicita transmitir datos
63	3F	?	Listo para transmitir o recibir/Entabla comunicación
64	40	@	Caracter de ejecutando el proceso
76	4C	L	Caracter que pide salir del modo
70	46	F	Caracter que indica que ocurrió un error
87	57	W	Caracter de espera
97	61	a	Caracter que cambia la posición de inicio en X
98	62	b	Caracter que cambia la posición de inicio en Y
99	63	c	Caracter que cambia la posición de fin en X
100	64	d	Caracter que cambia la posición de fin en Y
101	65	e	Caracter que cambia el parámetro del codificador en X
102	66	f	Caracter que cambia el parámetro del codificador en Y
103	67	g	Caracter que cambia la salida
104	68	h	Caracter que cambia la entrada
105	69	i	Caracter que apaga o enciende la fuente
106	6A	j	Caracter que apaga o enciende el taladro
107	6B	k	Caracter que modifica la resolución del motor
108	6C	l	Caracter que modifica la posición del eje X en pasos
109	6D	m	Caracter que modifica la posición del eje Y en pasos
110	6E	n	Caracter que modifica los incrementos en X
111	6F	o	Caracter que modifica los incrementos en Y
112	70	p	Caracter que modifica la velocidad lineal en X y en Y
113	71	q	Caracter que modifica la aceleración lineal en X y en Y
114	72	r	Caracter que modifica el recorrido angular en X y en Y
115	73	s	Caracter que modifica el recorrido lineal en X y en Y
116	74	t	Caracter que modifica la velocidad angular en X y en Y
117	75	u	Caracter que modifica la aceleración angular en X y en Y
120	78	x	Caracter que actualiza las coordenadas en X y en Y
122	7A	z	Caracter que indica un corto en la fuente

Tabla 2.5.

A continuación se muestran una serie de tablas que indican la secuencia de transmisión de los caracteres para la ejecución: **Manual, Automático, Interactivo** y **Perforado de tarjetas**. Para efecto de una mejor comprensión de lo que representa un **BYTE**, este se considera como un valor transmitido en hexadecimal y que representa un caracter ASCII.

a) Secuencia de instrucciones para el acceso y funcionamiento de todas las operaciones posibles para la ejecución **Manual.**

Realiza una perforación					
PC		**—**	**µC**		**DESCRIPCIÓN**
DEC	**CAR**		**DEC**	**CAR**	
31	▼	→			Transmitiendo el caracter que activa el taladro
		←	64	@	Transmitiendo el caracter de ejecutando el proceso
		←	63	?	Listo para transmitir o recibir/Entabla comunicación
63	?	→			Listo para transmitir o recibir/Entabla comunicación

Acceso al modo Manual					
PC		**—**	**µC**		**DESCRIPCIÓN**
DEC	**CAR**		**DEC**	**CAR**	
63	?	→			Listo para transmitir o recibir/Entabla comunicación
		←	63	?	Listo para transmitir o recibir/Entabla comunicación
58	:	→			Transmitiendo el caracter de ejecución en el modo Manual
		←	75	K	Llego el dato transmite el siguiente byte
21	§	→			Transmitiendo el caracter de resolución del motor
		←	79	O	Llego el dato transmite el siguiente byte
BYTE1		→			Se transmite el 1er byte que corresponde la resolución del motor
		←	83	S	Llego el dato transmite el siguiente byte
BYTE2		→			Se transmite el 2do byte que corresponde la resolución del motor
		←	63	?	Listo para transmitir o recibir/Entabla comunicación
63	?	→			Listo para transmitir o recibir/Entabla comunicación
		←	87	W	Transmitiendo el caracter de espera
		←	63	?	Listo para transmitir o recibir/Entabla comunicación

Realiza un desplazamiento en X o Y							
PC				**—**	**µC**		**DESCRIPCIÓN**
DEC		**CAR**			**DEC**	**CAR**	
68	66	D	B	→			Transmitiendo cualquiera de los caracteres que desplaza el eje X o Y a la derecha, izquierda, arriba o abajo
67	69	C	E				
				←	64	@	Transmitiendo el caracter de ejecutando el proceso
				←	63	?	Listo para transmitir o recibir/Entabla comunicación
63		?		→			Listo para transmitir o recibir/Entabla comunicación

Salida de un dato binario

PC		—	µC		DESCRIPCIÓN
DEC	CAR		DEC	CAR	
40	(	→			Transmitiendo el caracter de salida binaria
		←	79	O	Llego el dato transmite el 1er byte
BYTE 1		→			Se transmite el 1er byte que corresponde a la salida binaria
		←	83	S	Llego el dato transmite el 2do byte
BYTE 2		→			Se transmite el 2do byte que corresponde a la salida binaria
		←	63	?	Listo para transmitir o recibir/Entabla comunicación
63	?	→			Listo para transmitir o recibir/Entabla comunicación

Graba un punto

PC		—	µC		DESCRIPCIÓN
DEC	CAR		DEC	CAR	
71	G	→			Transmitiendo el caracter que graba el punto
		←	79	O	Llego el dato transmite el 1er byte
BYTE 1		→			Se transmite el 1er byte del punto que se quiere grabar
		←	83	S	Llego el dato transmite el 2do byte
BYTE 2		→			Se transmite el 2do byte del punto que se quiere grabar
		←	63	?	Listo para transmitir o recibir/Entabla comunicación
63	?	→			Listo para transmitir o recibir/Entabla comunicación

Desplázate al punto

PC		—	µC		DESCRIPCIÓN
DEC	CAR		DEC	CAR	
48	0	→			Transmitiendo el caracter de muévete al punto
		←	79	O	Llego el dato transmite el 1er byte
BYTE 1		→			Se transmite el 1er byte del punto donde se requiere desplazar
		←	83	S	Llego el dato transmite el 2do byte
BYTE 2		→			Se transmite el 2do byte del punto donde se requiere desplazar
		←	64	@	Transmitiendo el caracter de ejecutando el proceso
		←	63	?	Listo para transmitir o recibir/Entabla comunicación
63	?	→			Listo para transmitir o recibir/Entabla comunicación

Modifica los incrementos en X o Y

PC				—	µC		DESCRIPCIÓN
DEC		CAR			DEC	CAR	
74	75	J	K	→			Transmitiendo cualquiera de los el caracteres que cambian los incrementos del eje X o Y
				←	79	O	Llego el dato transmite el 1er byte
BYTE 1				→			Se transmite el 1er byte correspondiente a los incrementos
				←	83	S	Llego el dato transmite el 2do byte
BYTE 2				→			Se transmite el 2do byte correspondiente a los incrementos
				←	79	O	Llego el dato transmite el 3er byte
BYTE 3				→			Se transmite el 3er byte correspondiente a los incrementos
				←	63	?	Listo para transmitir o recibir/Entabla comunicación
63		?		→			Listo para transmitir o recibir/Entabla comunicación

Modifica la resolución del motor

PC		—	µC		DESCRIPCIÓN
DEC	CAR		DEC	CAR	
58	:	→			Transmitiendo el caracter de ejecución en el modo Manual
		←	75	K	Llego el dato transmite el siguiente byte
21	§	→			Transmitiendo el caracter de resolución del motor
		←	79	O	Llego el dato transmite el 1er byte
BYTE2		→			Se transmite el 1er byte que corresponde la resolución del motor
		←	83	S	Llego el dato transmite el 2do byte
BYTE2		→			Se transmite el 2do byte que corresponde la resolución del motor
		←	63	?	Listo para transmitir o recibir/Entabla comunicación
63	?	→			Listo para transmitir o recibir/Entabla comunicación
		←	87	W	Transmitiendo el caracter de espera
		←	63	?	Listo para transmitir o recibir/Entabla comunicación
63	?	→			Listo para transmitir o recibir/Entabla comunicación

Modifica la aceleración o velocidad lineal X o Y

PC				—	µC		DESCRIPCIÓN
DEC		CAR			DEC	CAR	
73	89	I	Y	→			Transmitiendo cualquiera de los el caracteres que cambian la
72	90	J	Z				velocidad o aceleración en X o Y
				←	79	O	Llego el dato transmite el 1er byte
BYTE 1				→			Se transmite el 1er byte correspondiente a la parte entera
				←	83	S	Llego el dato transmite el 2do byte
BYTE 2				→			Se transmite el 2do byte correspondiente a la parte entera
				←	79	O	Llego el dato transmite el 3er byte
BYTE 3				→			Se transmite el 3er byte correspondiente a la parte decimal
				←	83	S	Llego el dato transmite el 4to byte
BYTE 4				→			Se transmite el 4to byte correspondiente a la parte decimal
				←	63	?	Listo para transmitir o recibir/Entabla comunicación
63		?		→			Listo para transmitir o recibir/Entabla comunicación

Modifica el estado de la mesa (Inicio/ Fin) en la pantalla

PC		—	µC				DESCRIPCIÓN
DEC	CAR		DEC		CAR		
		←	62		>		Transmitiendo el caracter que solicita transmitir datos
83	S	→					Llego el dato transmite el siguiente byte
		←	97	99	a	c	Transmitiendo el caracter que cambia la posición de inicio y fin
			98	100	b	d	en X y en Y
81	Q	→					Llego el dato transmite el siguiente byte
		←	BYTE				Se transmite el byte(30,31) correspondiente a un 0 o un 1
78	N	→					Transmitiendo el caracter que pide restablecer la comunicación
		←	63		?		Listo para transmitir o recibir/Entabla comunicación
63	?	→					Listo para transmitir o recibir/Entabla comunicación

Modifica el parámetro de entrada en la pantalla

PC		—	µC		DESCRIPCIÓN
DEC	CAR		DEC	CAR	
		←	62	>	Transmitiendo el caracter que solicita transmitir datos
83	S	→			Llego el dato transmite el siguiente byte
		←	104	h	Transmitiendo el caracter que cambia la entrada
81	Q	→			Llego el dato transmite el siguiente byte
		←	BYTE		Se transmite el byte que corresponde a la entrada binaria
78	N	→			Transmitiendo el caracter que pide restablecer la comunicación
		←	63	?	Listo para transmitir o recibir/Entabla comunicación
63	?	→			Listo para transmitir o recibir/Entabla comunicación

Modifica los parámetros del codificador X o Y en la pantalla

PC		—	µC				DESCRIPCIÓN
DEC	CAR		DEC		CAR		
		←	62		>		Transmitiendo el caracter que solicita transmitir datos
83	S	→					Llego el dato transmite el siguiente byte
		←	101	102	e	f	Transmitiendo el caracter que cambia el codificador en X o en Y
81	Q	→					Llego el dato transmite el 1er byte
		←	BYTE 1				Se transmite el 1er byte que corresponde al codificador en X o Y
82	R	→					Llego el dato transmite el 2do byte
		←	BYTE 2				Se transmite el 2do byte que corresponde al codificador en X o Y
78	N	→					Transmitiendo el caracter que pide restablecer la comunicación
		←	63		?		Listo para transmitir o recibir/Entabla comunicación
63	?	→					Listo para transmitir o recibir/Entabla comunicación

Cancela un desplazamiento en X o en Y y actualiza la posición de la mesa en la pantalla

PC		—	µC		DESCRIPCIÓN
DEC	CAR		DEC	CAR	
123	{				Transmitiendo el caracter de cancelar la operación
		←	63	?	Listo para transmitir o recibir/Entabla comunicación
63	?	→			Listo para transmitir o recibir/Entabla comunicación
		←	62	>	Transmitiendo el caracter que solicita transmitir datos
83	S	→			Llego el dato transmite el siguiente byte
		←	120	x	Transmitiendo el caracter que actualiza los puntos en X y en Y
81	Q	→			Llego el dato transmite el 1er byte
		←	BYTE 1		Se transmite el 1er byte correspondiente a X
82	R	→			Llego el dato transmite el 2do byte
		←	BYTE 2		Se transmite el 2do byte correspondiente a X
84	T	→			Llego el dato transmite el 3er byte
		←	BYTE 3		Se transmite el 3er byte correspondiente a X
81	Q	→			Llego el dato transmite el 1er byte
		←	BYTE 1		Se transmite el 1er byte correspondiente a Y
82	R	→			Llego el dato transmite el 2do byte
		←	BYTE 2		Se transmite el 2do byte correspondiente a Y
84	T	→			Llego el dato transmite el 3er byte
		←	BYTE 3		Se transmite el 3er byte correspondiente a Y
78	N	→			Transmitiendo el caracter que pide restablecer la comunicación
		←	63	?	Listo para transmitir o recibir/Entabla comunicación
63	?	→			Listo para transmitir o recibir/Entabla comunicación

Enciende o apaga la fuente

PC			μC		DESCRIPCIÓN
DEC	CAR		DEC	CAR	
24 | 23	↑ | ↓	→			Transmitiendo cualquiera de los caracteres que enciende o paga la fuente.
		←	79	O	Llego el dato transmite el siguiente byte
63	?	→			Listo para transmitir o recibir/Entabla comunicación
		←	63	?	Listo para transmitir o recibir/Entabla comunicación

b) Secuencia de instrucciones para la ejecución **Automático** e **Interactivo.**

Acceso al modo Automático o Interactivo

PC			μC		DESCRIPCIÓN
DEC	CAR		DEC	CAR	
63	?	→			Listo para transmitir o recibir/Entabla comunicación
		←	63	?	Listo para transmitir o recibir/Entabla comunicación
59 | 60	: | <	→			Transmitiendo el caracter de ejecución en el modo Automático o Interactivo
		←	75	K	Llego el dato transmite el siguiente byte
21	§	→			Transmitiendo el caracter de resolución del motor
		←	79	O	Llego el dato transmite el 1er byte
BYTE1		→			Se transmite el 1er byte que corresponde la resolución del motor
		←	83	S	Llego el dato transmite el 2do byte
BYTE2		→			Se transmite el 2do byte que corresponde la resolución del motor
		←	63	?	Listo para transmitir o recibir/Entabla comunicación
72	H	→			Transmitiendo el caracter de velocidad lineal en X
		←	79	O	Llego el dato transmite el 1er byte
BYTE 1		→			Se transmite el 1er byte correspondiente a la parte entera
		←	83	S	Llego el dato transmite el 2do byte
BYTE 2		→			Se transmite el 2do byte correspondiente a la parte entera
		←	79	O	Llego el dato transmite el 3er byte
BYTE 3		→			Se transmite el 3er byte correspondiente a la parte decimal
		←	83	S	Llego el dato transmite el 4to byte
BYTE 4		→			Se transmite el 4to byte correspondiente a la parte decimal
		←	63	?	Listo para transmitir o recibir/Entabla comunicación
90	Z	→			Transmitiendo el caracter de aceleración lineal en X
		←	79	O	Llego el dato transmite el 1er byte
BYTE 1		→			Se transmite el 1er byte correspondiente a la parte entera
		←	83	S	Llego el dato transmite el 2do byte
BYTE 2		→			Se transmite el 2do byte correspondiente a la parte entera
		←	79	O	Llego el dato transmite el 3er byte
BYTE 3		→			Se transmite el 3er byte correspondiente a la parte decimal
		←	83	S	Llego el dato transmite el 4to byte
BYTE 4		→			Se transmite el 4to byte correspondiente a la parte decimal
		←	63	?	Listo para transmitir o recibir/Entabla comunicación
73	I	→			Transmitiendo el caracter de velocidad lineal en Y
		←	79	O	Llego el dato transmite el 1er byte
BYTE 1		→			Se transmite el 1er byte correspondiente a la parte entera
	•	←	83	S	Llego el dato transmite el 2do byte
BYTE 2		→			Se transmite el 2do byte correspondiente a la parte entera
		←	79	O	Llego el dato transmite el 3er byte

Continuación de la tabla anterior

PC		—	µC		DESCRIPCIÓN
DEC	CAR		DEC	CAR	
BYTE 3		→			Se transmite el 3er byte correspondiente a la parte decimal
		←	83	S	Llego el dato transmite el 4to byte
BYTE 4		→			Se transmite el 4to byte correspondiente a la parte decimal
		←	63	?	Listo para transmitir o recibir/Entabla comunicación
89	Y	→			Transmitiendo el caracter de aceleración lineal en Y
		←	79	O	Llego el dato transmite el 1er byte
BYTE 1		→			Se transmite el 1er byte correspondiente a la parte entera
		←	83	S	Llego el dato transmite el 2do byte
BYTE 2		→			Se transmite el 2do byte correspondiente a la parte entera
		←	79	O	Llego el dato transmite el 3er byte
BYTE 3		→			Se transmite el 3er byte correspondiente a la parte decimal
		←	83	S	Llego el dato transmite el 4to byte
BYTE 4		→			Se transmite el 4to byte correspondiente a la parte decimal
		←	63	?	Listo para transmitir o recibir/Entabla comunicación
63	?	→			Listo para transmitir o recibir/Entabla comunicación
		←	87	W	Transmitiendo el caracter de espera
		←	63	?	Listo para transmitir o recibir/Entabla comunicación
63	?	→			Listo para transmitir o recibir/Entabla comunicación

Transmisión de una sentencia

PC		—	µC		DESCRIPCIÓN
DEC	CAR		DEC	CAR	
63	?	→			Listo para transmitir o recibir/Entabla comunicación
		←	63	?	Listo para transmitir o recibir/Entabla comunicación
	BYTE	→			Se transmite el byte correspondiente a la sentencia
		←		S	Llego el dato transmite el siguiente byte
84	T	→			Llego el dato transmite el siguiente byte
		←	63	?	Listo para transmitir o recibir/Entabla comunicación
63	?	→			Listo para transmitir o recibir/Entabla comunicación

Transmisión del argumento

PC		—	µC		DESCRIPCIÓN
DEC	CAR		DEC	CAR	
63	?	→			Listo para transmitir o recibir/Entabla comunicación
		←	63	?	Listo para transmitir o recibir/Entabla comunicación
	BYTE	→			Se transmite el 1er byte correspondiente a la parte entera o decimal para X o Y.
		←		S	Llego el dato transmite el 2do byte
	BYTE	→			Se transmite el 2do byte correspondiente a la parte entera o decimal para X o Y.
		←	63	?	Listo para transmitir o recibir/Entabla comunicación
63	?	→			Listo para transmitir o recibir/Entabla comunicación

Modifica el estado de la mesa (Inicio/ Fin) en la pantalla

PC		—	µC				DESCRIPCIÓN
DEC	CAR		DEC		CAR		
		←	62		>		Transmitiendo el caracter que solicita transmitir datos
83	S	→					Llego el dato transmite el siguiente byte
		←	97 98	99 100	a b	c d	Transmitiendo el caracter que cambia la posición de inicio y fin en X y en Y
81	Q	→					Llego el dato transmite el siguiente byte
		←	BYTE				Se transmite el byte (30 o31) correspondiente a un 0 o un 1
78	K	→					Transmitiendo el caracter que pide restablecer la comunicación
		←	63		?		Listo para transmitir o recibir/Entabla comunicación
63	?	→					Listo para transmitir o recibir/Entabla comunicación

Modifica el estado del codificador en la pantalla

PC		—	µC				DESCRIPCIÓN
DEC	CAR		DEC		CAR		
		←	62		>		Transmitiendo el caracter que solicita transmitir datos
83	S	→					Llego el dato transmite el siguiente byte
		←	101	102	e	f	Transmitiendo caracter que cambian los datos del codificador en X y Y en forma absoluta o relativa
81	Q	→					Llego el dato transmite el 1er byte
		←	BYTE				Se transmite el 1er byte que corresponde al codificador en X
82	R	→					Llego el dato transmite el 2do byte
		←	BYTE				Se transmite el 2do byte que corresponde al codificador en X
81	Q	→					Llego el dato transmite el 1er byte
		←	BYTE				Se transmite el 1er byte que corresponde al codificador en Y
82	R	→					Llego el dato transmite el 2do byte
		←	BYTE				Se transmite el 2do byte que corresponde al codificador en Y
78	N	→					Transmitiendo el caracter que pide restablecer la comunicación
		←	63		?		Listo para transmitir o recibir/Entabla comunicación
63	?	→					Listo para transmitir o recibir/Entabla comunicación

Modifica el estado de la fuente en la pantalla

PC		—	µC		DESCRIPCIÓN
DEC	CAR		DEC	CAR	
		←	62	>	Transmitiendo el caracter que solicita transmitir datos
83	S	→			Llego el dato transmite el siguiente byte
		←	105	i	Transmitiendo caracter que modifica el estado de la fuente.
81	Q	→			Llego el dato transmite el siguiente byte
		←	BYTE		Se transmite el byte (30 o 31) que corresponde a un (0) o un (1)
78	K	→			Transmitiendo el caracter que pide restablecer la comunicación
		←	63	?	Listo para transmitir o recibir/Entabla comunicación
63	?	→			Listo para transmitir o recibir/Entabla comunicación

Modifica el estado del taladro en la pantalla

PC		—	µC		DESCRIPCIÓN
DEC	CAR		DEC	CAR	
		←	62	>	Transmitiendo el caracter que solicita transmitir datos
83	S	→			Llego el dato transmite el siguiente byte
		←	106	j	Transmitiendo caracter que modifica el estado del taladro.
81	Q	→			Llego el dato transmite el siguiente byte
		←	BYTE		Se transmite el byte(30 o 31) correspondiente a un (0) o un (1)
78	K	→			Transmitiendo el caracter que pide restablecer la comunicación
		←	63	?	Listo para transmitir o recibir/Entabla comunicación
63	?	→			Listo para transmitir o recibir/Entabla comunicación

Modifica la resolución del motor en la pantalla

PC		—	µC		DESCRIPCIÓN
DEC	CAR		DEC	CAR	
		←	62	>	Transmitiendo el caracter que solicita transmitir datos
83	S	→			Llego el dato transmite el siguiente byte
		←	107	k	Transmitiendo caracter que modifica la resolución del motor
81	Q	→			Llego el dato transmite el 1er byte
		←	BYTE		Se transmite el 1er byte correspondiente a la resolución
82	R	→			Llego el dato transmite el 2do byte
		←	BYTE		Se transmite el 2do byte correspondiente a la resolución
84	T	→			Llego el dato transmite el 3er byte
		←	BYTE		Se transmite el 3er byte correspondiente a la resolución
78	K	→			Transmitiendo el caracter que pide restablecer la comunicación
		←	63	?	Listo para transmitir o recibir/Entabla comunicación
63	?	→			Listo para transmitir o recibir/Entabla comunicación

Modifica la posición en pasos en X o Y en la pantalla

PC		—	µC		DESCRIPCIÓN
DEC	CAR		DEC	CAR	
		←	62	>	Transmitiendo el caracter que solicita transmitir datos
83	S	→			Llego el dato transmite el siguiente byte
		←	108 | 109	l | m	Transmitiendo caracter que modifica los pasos en X o en Y
81	Q	→			Llego el dato transmite el 1er byte
		←	BYTE		Se transmite el 1er byte correspondiente a los pasos en X o en Y
82	R	→			Llego el dato transmite el 2do byte
		←	BYTE		Se transmite el 2do byte correspondiente a los pasos en X o en Y
84	T	→			Llego el dato transmite el 3er byte
		←	BYTE		Se transmite el 3er byte correspondiente a los pasos en X o en Y
78	K	→			Transmitiendo el caracter que pide restablecer la comunicación
		←	63	?	Listo para transmitir o recibir/Entabla comunicación

Modifica la posición en pasos en X o Y en la pantalla

PC		—	μC		DESCRIPCIÓN
DEC	CAR		DEC	CAR	
		←	62	>	Transmitiendo el caracter que solicita transmitir datos
83	S	→			Llego el dato transmite el siguiente byte
		←	108 | 109	1 | m	Transmitiendo caracter que modifica los pasos en X o en Y
81	Q	→			Llego el dato transmite el 1er byte
		←	BYTE		Se transmite el 1er byte correspondiente a los pasos en X o en Y
82	R	→			Llego el dato transmite el 2do byte
		←	BYTE		Se transmite el 2do byte correspondiente a los pasos en X o en Y
84	T	→			Llego el dato transmite el 3er byte
		←	BYTE		Se transmite el 3er byte correspondiente a los pasos en X o en Y
78	K	→			Transmitiendo el caracter que pide restablecer la comunicación
		←	63	?	Listo para transmitir o recibir/Entabla comunicación
63	?	→			Listo para transmitir o recibir/Entabla comunicación

Modifica los incrementos en X o Y en la pantalla

PC		—	μC		DESCRIPCIÓN
DEC	CAR		DEC	CAR	
		←	62	>	Transmitiendo el caracter que solicita transmitir datos
83	S	→			Llego el dato transmite el siguiente byte
		←	110 | 111	n | o	Transmitiendo caracter que modifica los a incrementos en X o en Y
81	Q	→			Llego el dato transmite el 1er byte
		←	BYTE		Se transmite el 1er byte correspondiente a incrementos en X o en Y
82	R	→			Llego el dato transmite el 2do byte
		←	BYTE		Se transmite el 2do byte correspondiente a incrementos en X o en Y
84	T	→			Llego el dato transmite el 3er byte
		←	BYTE		Se transmite el 3er byte correspondiente a incrementos en X o en Y
78	K	→			Transmitiendo el caracter que pide restablecer la comunicación
		←	63	?	Listo para transmitir o recibir/Entabla comunicación
63	?	→			Listo para transmitir o recibir/Entabla comunicación

Modifica la velocidad o aceleración lineal o angular en X y Y en la pantalla

PC		—	μC		DESCRIPCIÓN
DEC	CAR		DEC	CAR	
		←	62	>	Transmitiendo el caracter que solicita transmitir datos
83	S	→			Llego el dato transmite el siguiente byte
		←	112 | 113 116 | 117	p | q t | u	Transmitiendo caracter que modifica la vel. o ace. lineal o angular en X y Y
81	Q	→			Llego el dato transmite el 1er byte
		←	BYTE		Se transmite el 1er byte correspondiente a la parte entera de X
82	R	→			Llego el dato transmite el 2do byte
		←	BYTE		Se transmite el 2do byte correspondiente a la parte entera de X
84	T	→			Llego el dato transmite el 3er byte
		←	BYTE		Se transmite el 3er byte correspondiente a la parte decimal X
82	R	→			Llego el dato transmite el 4to byte

Continuación de la tabla anterio

PC		—	µC		DESCRIPCIÓN
DEC	CAR		DEC	CAR	
		←.	BYTE		Se transmite el 4to byte correspondiente a la parte decimal de X
81	Q	→			Llego el dato transmite el 1er byte
		←	BYTE		Se transmite el 1er byte correspondiente a la parte entera de Y
82	R	→			Llego el dato transmite el 2do byte
		←	BYTE		Se transmite el 2do byte correspondiente a la parte entera de Y
84	T	→			Llego el dato transmite el 3er byte
		←	BYTE		Se transmite el 3er byte correspondiente a la parte decimal Y
82	R	→			Llego el dato transmite el 4to byte
		←	BYTE		Se transmite el 4to byte correspondiente a la parte decimal de Y
78	K	→			Transmitiendo el caracter que pide restablecer la comunicación
		←	63	?	Listo para transmitir o recibir/Entabla comunicación
63	?	→			Listo para transmitir o recibir/Entabla comunicación

Modifica el recorrido en X en la pantalla

PC		—	µC		DESCRIPCIÓN
DEC	CAR		DEC	CAR	
		←	62	>	Transmitiendo el caracter que solicita transmitir datos
83	S	→			Llego el dato transmite el siguiente byte
		←	114 115	r s	Transmitiendo caracter que modifica el recorrido en X o en Y
81	Q	→			Llego el dato transmite el 1er byte
		←	BYTE		Se transmite el 1er byte correspondiente al recorrido en X o en Y
82	R	→			Llego el dato transmite el 2do byte
		←	BYTE		Se transmite el 2do byte correspondiente al recorrido en X o en Y
84	T	→			Llego el dato transmite el 3er byte
		←	BYTE		Se transmite el 3er byte correspondiente al recorrido en X o en Y
78	K	→			Transmitiendo el caracter que pide restablecer la comunicación
		←	63	?	Listo para transmitir o recibir/Entabla comunicación
63	?	→			Listo para transmitir o recibir/Entabla comunicación

c) Secuencia de instrucciones para la ejecución **Perforado de tarjetas.**

Acceso al modo Perforado de circuitos impresos

PC		—	µC		DESCRIPCIÓN
DEC	CAR		DEC	CAR	
63	?	→			Listo para transmitir o recibir/Entabla comunicación
		←	63	?	Listo para transmitir o recibir/Entabla comunicación
61	:	→			Transmitiendo el caracter de ejecución en el modo Perforado de tarjetas.
		←	75	K	Llego el dato transmite el siguiente byte
21	§	→			Transmitiendo el caracter de resolución del motor
		←	79	O	Llego el dato transmite el 1er byte
BYTE1		→			Se transmite el 1er byte que corresponde la resolución del motor
		←	83	S	Llego el dato transmite el 2do byte
BYTE2		→			Se transmite el 2do byte que corresponde la resolución del motor

Continuación de la tabla anterior

PC		—	µC		DESCRIPCIÓN
DEC	CAR		DEC	CAR	
		←	63	?	Listo para transmitir o recibir/Entabla comunicación
72	H	→			Transmitiendo el caracteres de la velocidad en X
		←	79	O	Llego el dato transmite el 1er byte
BYTE 1		→			Se transmite el 1er byte correspondiente a la parte entera
		←	83	S	Llego el dato transmite el 2do byte
BYTE 2		→			Se transmite el 2do byte correspondiente a la parte entera
		←	79	O	Llego el dato transmite el 3er byte
BYTE 3		→			Se transmite el 3er byte correspondiente a la parte decimal
		←	83	S	Llego el dato transmite el 4to byte
BYTE 4		→			Se transmite el 4to byte correspondiente a la parte decimal
		←	63	?	Listo para transmitir o recibir/Entabla comunicación
73	I	→			Transmitiendo el caracteres de la velocidad en Y
		←	79	O	Llego el dato transmite el 1er byte
BYTE 1		→			Se transmite el 1er byte correspondiente a la parte entera
		←	83	S	Llego el dato transmite el 2do byte
BYTE 2		→			Se transmite el 2do byte correspondiente a la parte entera
		←	79	O	Llego el dato transmite el 3er byte
BYTE 3		→			Se transmite el 3er byte correspondiente a la parte decimal
		←	83	S	Llego el dato transmite el 4to byte
BYTE 4		→			Se transmite el 4to byte correspondiente a la parte decimal
		←	63	?	Listo para transmitir o recibir/Entabla comunicación
23	I	→			Transmitiendo el caracter que apaga fuente
		←	79	O	Llego el dato transmite el siguiente byte
		←	63	?	Listo para transmitir o recibir/Entabla comunicación
63	?	→			Listo para transmitir o recibir/Entabla comunicación
		←	87	W	Transmitiendo el caracter de espera
		←	63	?	Listo para transmitir o recibir/Entabla comunicación

Inicializa el proceso de perforado

PC		AL	µC		DESCRIPCIÓN
DEC	CAR		DEC	CAR	
85	U	→			Transmitiendo el caracter que inicializa el proceso
		←	87	W	Transmitiendo el caracter de espera
63	?	→			Listo para transmitir o recibir/Entabla comunicación
		←	63	?	Listo para transmitir o recibir/Entabla comunicación

Transmisión de la coordenada X y Y

PC		—	µC		DESCRIPCIÓN
DEC	CAR		DEC	CAR	
106	j	→			Transmitiendo punto en X
		←	79	O	Llego el dato transmite el 1er byte
BYTE1		→			Transmitiendo el 1er byte correspondiente al punto en X
		←	83	S	Llego el dato transmite el 2do byte
BYTE 2		→			Transmitiendo el 2do byte correspondiente al punto en X
		←	79	O	Llego el dato transmite el 3er byte
BYTE3		→			Transmitiendo el 3er byte correspondiente al punto en X

Continuación de la tabla anterior

PC		—	µC		DESCRIPCIÓN
DEC	CAR		DEC	CAR	
		←	63	?	Listo para transmitir o recibir/Entabla comunicación
107	k	→			Transmitiendo punto en Y
		←	79	O	Llego el dato transmite el 1er byte
BYTE1		→			Transmitiendo el 1er byte correspondiente al punto en Y
		←	83	S	Llego el dato transmite el 2do byte
BYTE2		→			Transmitiendo el 2do byte correspondiente al punto en Y
		←	79	O	Llego el dato transmite el 3er byte
BYTE3		→			Transmitiendo el 3er byte correspondiente al punto en Y
		←	64	@	Transmitiendo el caracter de ejecutando el proceso
		←	63	?	Listo para transmitir o recibir/Entabla comunicación
31	▼	→			Transmitiendo el caracter que activa el taladro
		←	64	@	Transmitiendo el caracter de ejecutando el proceso
		←	63	?	Listo para transmitir o recibir/Entabla comunicación
63	?	→			Listo para transmitir o recibir/Entabla comunicación

Las siguientes tablas de transmisión de caracteres se emplean en cualquiera de la cuatro modos de ejecución.

Pausa por medio del teclado

PC		—	µC		DESCRIPCIÓN
DEC	CAR		DEC	CAR	
29	↔	→			Transmitiendo el caracter de pausa
		←	29	↔	Transmitiendo el caracter de pausa

Continuar por medio del teclado

PC		—	µC		DESCRIPCIÓN
DEC	CAR		DEC	CAR	
30	▲	→			Transmitiendo el caracter de continua
		←	64	@	Transmitiendo el caracter de ejecutando el proceso

Pausa por medio del rac de control

PC		—	µC		DESCRIPCIÓN
DEC	CAR		DEC	CAR	
		←	29	↔	Transmitiendo el caracter de pausa
29	↔	→			Transmitiendo el caracter de pausa

Continuar por medio del rac de control

PC		—	µC		DESCRIPCIÓN
DEC	CAR		DEC	CAR	
		←	64	@	Transmitiendo el caracter de ejecutando el proceso
63	?	→			Listo para transmitir o recibir/Entabla comunicación

Suspender por medio del teclado					
PC		—	µC		DESCRIPCIÓN
DEC	CAR		DEC	CAR	
76	L	→			
		←	63	?	Listo para transmitir o recibir/Entabla comunicación
63	?	→			Listo para transmitir o recibir/Entabla comunicación

Suspender por medio del rac de control					
PC		—	µC		DESCRIPCIÓN
DEC	CAR		DEC	CAR	
		←	76	L	
63	?	→			Listo para transmitir o recibir/Entabla comunicación
		←	63	?	Listo para transmitir o recibir/Entabla comunicación

Corto en la fuente					
PC		—	µC		DESCRIPCIÓN
DEC	CAR		DEC	CAR	
		←	122	z	Transmitiendo el caracter que indica un corto en la fuente
78	N	→			Transmitiendo el caracter que pide restablecer la comunicación

Ocurrencia de un error					
PC		—	µC		DESCRIPCIÓN
DEC	CAR		DEC	CAR	
		←	70	F	Transmitiendo el caracter que indica que ocurrió un error
78	N	→			Transmitiendo el caracter que pide restablecer la comunicación

Para finalizar, además de las funciones que llevan el control de la transmisión de los caracteres, se emplean las funciones **Tbyte()**, y **Rbyte()**, las cuales tienen la tarea de transmitir y recibir los caracteres o byte's por el puerto serie. Estas rutinas se diseñaron para probar la funcionalidad del sistema 8031, también debido a que dicho sistema se diseño con una velocidad de transmisión en baudios muy diferente a las establecidas para la comunicación vía serie.

CAPÍTULO

III

DESCRIPCIÓN DEL SISTEMA

No hay nada que dé tanta vergüenza, como ver a alguien hacer algo que uno dijo que era imposible hacer.

Sam Edwing

III Menú principal.

El menú principal está formado por tres opciones, en donde dos de las cuales hacen referencia a las dos modalidades u opciones en las que puede operar este sistema: **Perforado de Tarjetas Impresas** y **Maquinado de Piezas**. La tercera opción es la de salida. Está pantalla aparecerá como se muestra en la Figura 3-1.

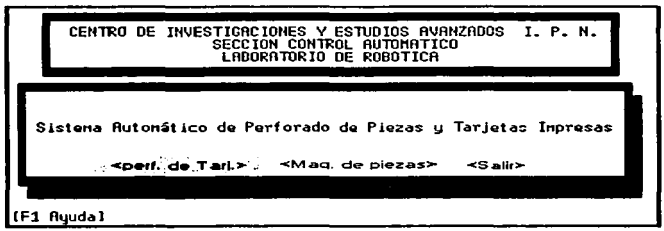


Figura 3-1. Menú principal

En este menú las opciones se pueden seleccionar de tres formas: primero, se pueden utilizar las flechas de desplazamiento para mover la barra iluminada a la opción que se requiere y entonces pulsar ENTER. Segundo, simplemente presione la letra mayúscula que aparece resaltada con el nombre de la opción. La tercera opción es pulsando un CLIC con el botón derecho del Mouse, colocando el puntero en la opción a elegir.

La ayuda en este menú se activa pulsando la tecla de función F1. En está opción la ayuda contempla tres ventanas, las cuales presentan dos opciones. **Mas y Regresar.** La opción **Mas** presenta ventana de ayuda y la opción **Regresar** retorna al menú principal. No es posible regresar a la ventana de la ayuda anterior, sólo se puede continuar hasta la última ventana de ayuda.

III.1 Menú Perforado de Tarjetas.

La pantalla del módulo de perforado de tarjetas está compuesta por un menú horizontal que cuenta con tres opciones: Archivo, Ejecutar y Ayuda. La opción Archivo abre un menú vertical con una serie opciones las cuales pueden ser elegidas de la misma forma que se realiza en el menú principal. La Tabla 3-1, muestra el contenido de cada opción del menú Perforado de Tarjetas.

MENÚ	OPCIONES
Archivo	Leer_Achivo, Información, Regresar al Menú Principal, Salir a Dos.
Ejecutar	
Ayuda	

Tabla 3-1. Sumario de los elementos del menú Perforado de Tarjetas.

En está modalidad las opciones Información y Ejecutar no estarán disponibles hasta que se cargue un archivo en la memoria.

Para cargar un archivo se debe accesar a la opción Leer_Archivo que se encuentra dentro del menú principal en la opción Archivo, Como se muestra en la Figura 3-2. Está opción abre una ventana en la cual se debe seleccionar por medio de un ENTER, el tipo de paquete que se utilizó para crear el circuito impreso, está ventana contiene tres opciones las cuales son: Smartwork, Tango y Orcad y son las únicas que reconoce el sistema.

El método para crear los archivos de texto de los anteriores paquetes se explicará más adelante.

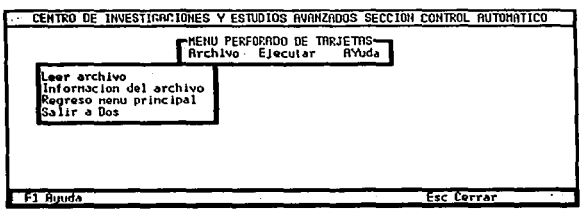


Figura 3-2. Menú desplegable de archivo.

Al elegir una de las tres opciones de la ventana anterior se abre un recuadro donde se escribirá la ruta y el nombre del archivo después presione ENTER, en está área sólo se admiten cuarenta caracteres. Está ventana se muestra en la Figura 3-3.

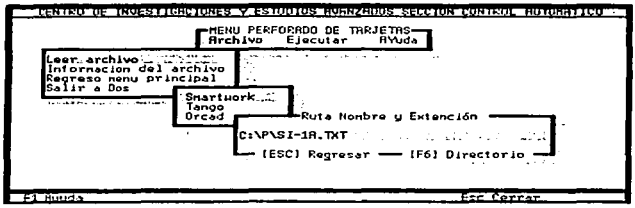


Figura 3-3. Pantalla de lectura de un archivo.

En está ventana se presentan dos opciones [Esc]Regresar y [F6]Directorio. Esc se utiliza para cancelar la operación de lectura. La opción [F6]Directorio, se emplea para localizar un archivo, sólo hay que escribir la posible ruta enseguida la instrucción *.TXT, con esto se abrirá una ventana que mostrará todos los archivos con la extensión .TXT que se encuentren en ese directorio. Cuando los archivos sobre pasan el limite de la ventana, aparecen en el recuadro de la ventana las opciones [Esc]Regresa y [↓]Más. La opción Esc regresa a la ventana anterior. Al pulsar la tecla [↓] se mostrará el siguiente bloque de archivos. Como se muestra en la Figura 3-4.

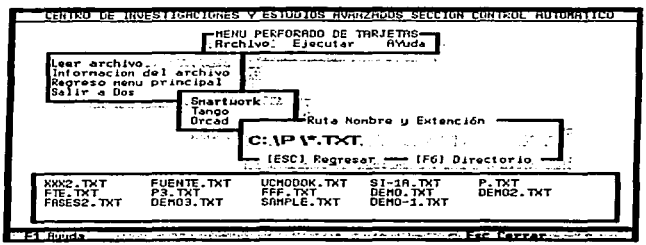


Figura 3-4. Pantalla que muestra el directorio

Si la ruta y el nombre del archivo son los correctos se cargará y regresará al menú de la opción **Archivo**, si existe algún error, entonces el sistema abrirá una ventana, indicando el tipo de error que se presento. Estos errores se pueden generar debido a que la ruta o el nombre del archivo estén erróneos, que el archivo especificado no pertenezca al establecido en la ventana anterior, o bien que el archivo no cumpla con la condiciones establecidas por este sistema.

La opción **Información**, accesa a una pantalla como se muestra en la Figura 3-5. La cual se divide en dos partes: Recuadros de información y Área de presentación de la tarjeta.

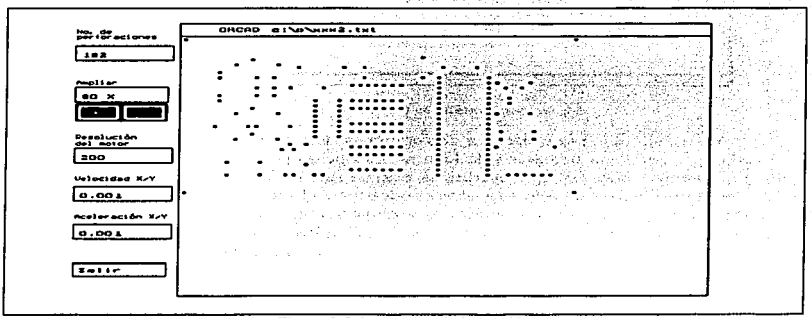


Figura 3-5. Pantalla de información.

En la parte izquierda de la pantalla se encuentran unos recuadros que muestran las condiciones iniciales en las cuales se va a ejecutar el sistema. El primer recuadro indica el número de perforaciones del circuito. El segundo recuadro, se utiliza para ampliar el tamaño de la tarjeta para su mejor visualización, con sólo presionar las teclas [+] o [-] se puede incrementar o decrementar el tamaño de la tarjeta respectivamente. En el recuadro superior se mostrará el porcentaje de ampliación de la tarjeta.

Los siguientes recuadros proporcionan la información de las condiciones en las cuales el sistema se va a ejecutar. Se pueden modificar estas condiciones con sólo presionar la tecla ALT y la letra mayúscula resaltada o por medio del Mouse.

El recuadro de **Resolución del motor**, indica el número de pasos de los motores, cabe resaltar que los motores de cada eje deben contar con la misma resolución. Los recuadros de **Velocidad** y **Aceleración**, establecen la velocidad y la aceleración con la que se han de desplazar los ejes X-Y. El recuadro de **Salir** regresa al menú de perforado de tarjetas. El área de presentación de la tarjeta muestra la posición que debe guardar la misma en la plataforma de la mesa X-Y. En la parte superior muestra el tipo de paquete que se empleó y el lugar donde se encuentra el archivo. La opción **Regresar al menú principal**, se utiliza para salir de la modalidad Perforado de Tarjetas y regresar al menú principal. La opción **Salir a Dos**, suspende la ejecución del sistema y regresa al sistema operativo sin pasar por el menú principal.

III.1.1 Ejecución modo Perforado de Tarjetas.

Cuando se hayan establecido las condiciones en las cuales el sistema a de operar, se accesa a la opción ejecutar, está pantalla se muestra en la Figura 3-6.

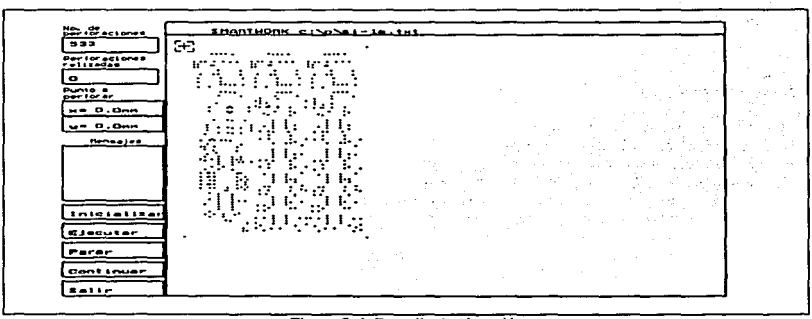


Figura 3-6. Pantalla de ejecución.

En la pantalla se encuentra el área de posición de la tarjeta, el área de mensajes y los recuadros de información, además se cuenta con las opciones: **Inicializar, Ejecutar, Parar, Continuar, Salir**. Estas opciones están disponibles en el momento de la ejecución del sistema.

Para seleccionar una opción se presionan conjuntamente la tecla ALT y la letra mayúscula resaltada, también se puede seleccionar por medio del Mouse.

Para iniciar con el proceso de perforado se elige la opción ejecutar, al ingresar a está modalidad el sistema mandará un mensaje que indica que se está inicializando, es decir, desplazando la mesa al inicio. Cuando termina desaparece ese mensaje en ese momento se puede iniciar con la perforación de la tarjeta impresa.

Para iniciar la perforación de tarjeta se elige la opción **Ejecutar**. El sistema se encarga de enviarle al microcontrolador las coordenadas de los puntos a perforar, por lo consiguiente el microcontrolador le envía al sistema las señales que especifican el desplazamiento y la perforación.

Los recuadros en la pantalla muestran la siguiente información: **Número de perforaciones, Número de perforaciones realizadas,** y el **Punto a perforar.**

El **Número de perforaciones realizadas** se incrementa conforme se realicen las perforaciones. El **Punto a perforar** indicará a que punto se va a desplazar la plataforma de la mesa para realizar la perforación.

El sistema cuenta con un indicador de posición gráfico el cual muestra el punto que va ha perforar, cuando se realice una perforación cambiará de color el punto, por medio de este sistema se puede saber como se están realizando las perforaciones.

Las opciones se encontrarán disponibles cuando la letra mayúscula estén en rojo. La opción de **Pausa** sólo puede ser usada en el momento en que se está desplazando la mesa, está opción puede activarse por medio del teclado o mediante el interruptor que se encuentra en el Rac de control. La opción **Continuar** puede ser activada de la misma manera. La opción de **Pausa** y **Continuar** pueden ser elegidas por alguna eventualidad que ocurra o simplemente para cambiar una broca.

Cuando se inicializa el sistema se regresa a las condiciones iniciales, esto quiere decir, que desplazará la plataforma de la mesa al inicio de los eje X y Y, en está opción es posible suspender la ejecución sin tener que salir del sistema.

Debido a que el rac de control cuenta con un detector de corto circuito en la fuente, cuando se presente está eventualidad se detendrá el proceso de perforado para corregir la falla.

La opción **Salir** se encuentra disponible en todo momento, ya que si ocurriera alguna eventualidad en el microcontrolador, se podría suspender o regresar al menú de **Perforado de tarjetas**. La opción de **Ayuda** cuenta con un sistema de ayuda en línea, la cual se activa pulsando la tecla de función **F1**. Es sensible al contexto, lo que significa que visualiza la información que está relacionada con lo que se está haciendo en ese momento. Para salir del sistema de ayuda se debe pulsar la tecla ESC o Cancelar.

III.2 Método para crear el archivo de texto.

Antes de explicar como se crea el archivo de texto primero se deben de contemplar las siguientes condiciones de diseño: Primero, el circuito se debe de encontrar enmarcado por un área de cuatro puntos, formando un cuadrado o un rectángulo, independientemente del paquete que se utilice para el diseño del circuito impreso (Orcad, Tango, Smartwork), como lo muestra la Figura 3-7. Segundo. El archivo fuente se debe guardar con la extensión [.PCB]. Tercero, El archivo de texto debe tener la extensión [.TXT].

Figura 3-7. Circuito Impreso enmarcado por los cuatro puntos.

Para cada paquete el método difiere, así que se recomienda seguir los pasos correctamente que se describen a continuación.

El archivo de texto de **SMARTWORK** se genera mediante los siguientes pasos:

1.- Contar con el archivo que contiene el circuito impreso.
2.- Ejecutar el archivo DOT.EXE.
3.- Elegir la opción (2) 2x DOT-MATRIX CHECKPLOT
4.- Introducir número de columnas.
5.- Escribir el nombre del archivo fuente con la extensión [.PCB].
6.- Borrar la instrucción Ltp1
7.- Escribir el nombre del archivo de salida con la extensión [.TXT].
8.- Elegir la opción (4) PADMASTER.

El archivo de texto de **TANGO** se genera mediante los siguientes pasos:

1.- Contar con el archivo que contiene el circuito.
2.- Ejecutar el archivo GPLOT.EXE para que accese a TANGO-PCB PHOTO-PLOTTER SYSTEM.
3.- Cargar el archivo que contiene al circuito y que tiene por extensión [.PCB].
4.- Elegir la opción N.C DRILL FILE GENERATION.

El archivo de texto de **ORCAD** se genera mediante los siguientes pasos:

1.- Contar con el archivo que contiene el circuito.
2.- Contar el archivo MODLOC.EXE. Para generar el archivo de texto.
3.- escriba la siguiente instrucción:

MODLOC [Unidad:][ruta][NOMBRE.PCB] [NOMBRE.TXT] VERBOSE

III.3 Maquinado de Piezas.

La pantalla del módulo de **Maquinado de Piezas** está compuesta por un menú horizontal de cuatro opciones: **Archivo, Editar, Ejecutar** y **Opciones**. Cada opción abre un menú vertical en donde las opciones pueden ser elegidas de la misma forma en que sea descrito con anterioridad. La Tabla 3-2, muestra el contenido del menú **Maquinado de Piezas**.

MENÚ	OPCIONES
Archivo	Nuevo, Leer Archivo, Salvar Archivo, Cerrar Archivo, Regresar al Menú Principal, Salir a Dos.
Editar	Compilar Archivo, Editar Archivo.
Ejecutar	Automático, Interactivo, Manual.
Opciones	Ayuda, Mensajes.

Tabla 3-2. Sumario de los elementos del menú Maquinado de Piezas.

La opción **Nuevo** permite la edición de un programa. Si existe un archivo presente en la pantalla del editor desplegará un mensaje con tres opciones: **Continua, Salvar** o **Cancelar**. **Continuar**, cierra el archivo activo y abre una ventana nueva. **Salvar**, almacena la información contenida en editor. **Cancelar**, suspende la ejecución de edición de un nuevo archivo.

La opción **Leer_Archivo**, se utiliza para cargar en memoria un archivo de un disco o de algún directorio del disco duro. Cuando se elige este comando se abre una ventana en donde se solicita la ruta y el nombre del archivo, después se debe presiona ENTER. Como se muestra en la Figura 3-8.

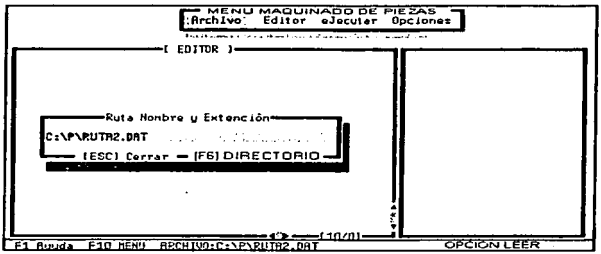


Figura 3-8 Pantalla de lectura de un archivo.

En la ventana existen dos opciones **ESC** que se utiliza para cerrar una ventana. La opción **[F6]** Directorio, le ayuda a localizar un archivo, sólo hay que escribir la posible ruta y el nombre, después se escribe la instrucción ***.DAT**, esto mostrará todos los archivos con la extensión .DAT que se encuentren en ese directorio.

Si en está opción ocurriera algún error en el archivo especificado, se abrirá una ventana, indicando el tipo de error, dándonos por opciones **Repetir** o **Cancelar**. La opción **Repetir**, regresa a la opción para corregir la ruta o el nombre del archivo según sea el caso y la opción **Cancelar**, suspende la operación de lectura.

La opción **Salvar_Archivo**, almacena lo que está actualmente en el editor en un archivo que contenga el nombre mostrado en la línea de estado. Cuando se activa está opción abre una ventana, permitiéndonos introducir el nombre del archivo en que se desea guardar el contenido actual del editor, si no se desea modificar el nombre sólo hay que presionar ENTER. En caso contrario modifique la ruta, el nombre y presione ENTER.

Si no se modificó el nombre continuación, se abrirá una ventana indicando que el nombre del archivo ya existe y da tres opciones a elegir: **Modificar, Salvar** o **Cancelar.**

La opción **Modificar**, regresará a la ventana anterior para cambiar de nombre. La opción **Salvar**, realizará la operación de sobreescribir el contenido anterior con el actual. La opción **Cancelar**, suspenderá la operación de salvar, retornando al menú principal. Cuando se almacene un programa, el archivo debe tener la extensión **.DAT**, para que después pueda ser cargado en memoria.

La opción **Cerrar_Archivo** se usa para despejar el contenido del editor. Antes de cerrar el archivo, verifica que el archivo haya sido salvado, si no, se sugiere que se guarden los cambios realizados, antes de cerrarlo. Si un archivo se cierra sin ser almacenado con anterioridad, se perderán los cambios efectuados desde la última vez que se guardó. Las opciones **Regresar al menú principal** y **Salir a Dos** realizan la misma función que las opciones que se encuentran en la modalidad Perforado de Tarjetas. Este menú de opciones se muestra en la Figura 3-9.

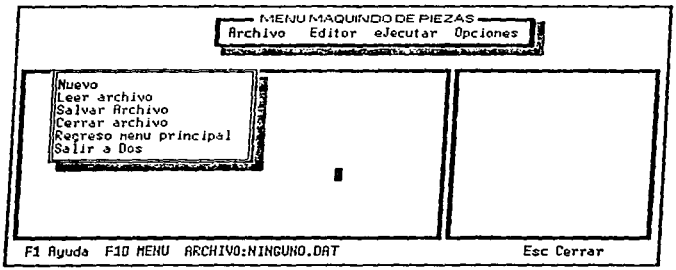


Figura 3-9. Menú desplegable de archivo.

La opción **Compilar_Archivo** permite codificar el archivo que está actualmente en el editor en un archivo en hexadecimal Este archivo se utilizará para enviar la información al microcontrolador y que de está manera se pueda ejecutar el programa en forma Automática o Interactiva.

Cuando se compila un archivo se abre una ventana indicando el número de líneas que contiene el programa, la línea que se está compilando, total de líneas compiladas y el número de errores que se presentaron, como se muestra en la Figura 3-10.

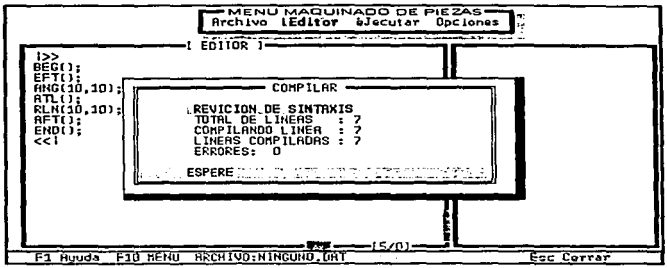


Figura 3-10. Ventana de compilación.

El compilador primero revisa la sintaxis del programa, si no hay ningún error se genera el archivo en hexadecimal. Si por el contrario existen errores se especificará el número de errores que se presentaron. Después mostrará en la pantalla de mensajes la línea y el tipo error que se presentó. Antes de crear el archivo en hexadecimal el compilador revisa que los últimos cambios realizados estén almacenados, si no lo están abrirá una ventana dando por opciones **Salvar** o **Cancelar**. La opción **Salvar** almacena el programa y genera el hexadecimal. La opción **Cancelar** suspende la ejecución compilar.

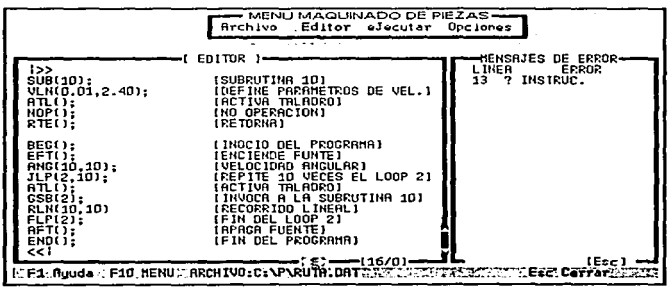


Figura 3-11. Pantalla del editor y mensajes.

La opción **Editar_Archivo** permitirá continuar o iniciar un nuevo programa. El editor está compuesto por el área de texto, barras de desplazamiento, línea de estado, indicador de posición del cursor y un área de mensajes. Como se muestra en la Figura 3-11.

En la línea de estado del editor se encuentran establecidas las condiciones en las cuales se está editando un programa. Está línea contiene:

a) Modo de inserción.
b) Las teclas de secuencia de funciones.
c) Nombre del archivo activo

El modo de inserción opera de la siguiente manera: INS indica que el editor está en modo insertar; es decir, que al introducir texto lo insertará en medio de lo que está ya en el editor, el modo opuesto es SUB, en este modo de operación, el texto nuevo sustituye al existente.

Las teclas de secuencia de función están compuestas por **F1** y **F10**. La tecla de función **F10**, regresa al menú principal, y la tecla de función **F1**, activa el sistema de ayuda en línea.

El nombre del archivo activo estará compuesto por la ruta y nombre con la extensión .DAT. Las barras de desplazamiento, se utilizan para desplazar el cursor en el área de texto por medio del Mouse.

El indicador de posición muestra en que línea y en que columna se encuentra el curso. El área de mensajes se localiza a un costado del editor y se usa para visualizar errores de sintaxis que ocurrieron en el momento de la compilar.

También en está ventana se utiliza visualizar los archivos contenidos en un directorio, está opción se encuentra disponible en la opción **Leer Archivo**.

El Editor reconoce secuencias o combinaciones de teclas utilizadas para facilitar la edición de texto, semejantes a la de otros editores de texto como por ejemplo: TURBO PASCAL, TURBO C/C++, etc. En la Tabla 3-3 se muestran todas las ordenes que se pueden emplear en el editor.

ORDEN	ACCIÓN
Cursor	**Cursor**
[Pg-Down]	MUEVE EL CURSOR UNA PÁGINA ABAJO
[Pg-Up]	MUEVE EL CURSOR UNA PÁGINA ARRIBA
[CTRL-HOME]	MUEVE EL CURSOR AL PRINCIPIO DEL ARCHIVO
[CTRL-END]	MUEVE EL CURSOR AL FINAL DEL ARCHIVO
[ALT-P]	MUEVE EL CURSOR A LA PARTE SUPERIOR DE LA PANTALLA
[ALT-F]	MUEVE EL CURSOR A LA PARTE INFERIOR DE LA PANTALLA
[←,↑,→,↓]	FLECHAS DE DESPLAZAMIENTO
[HOME]	MUEVE EL CURSOR AL COMIENZO DE LA LÍNEA
[END]	MUEVE EL CURSOR AL FINAL DE LA LÍNEA
Insertar	**Insertar**
[INS]	INSERTAR/SUBSTITUIR CARACTERES
[ENTER]	CREA UNA LÍNEA EN BLANCO
[ALT-I]	INSERTA UNA LÍNEA EN LA POSICIÓN DEL CURSOR
Borrar	**Borrar**
[BACKSPACE]	BORRA UN CARACTER A LA IZQUIERDA DEL CURSOR
[ALT-B]	BORRA UNA LÍNEA EN LA POSICIÓN DEL CURSOR
[DEL]	BORRA UN CARACTER QUE ESTE ENCIMA DEL CURSOR

Tabla 3-3. Ordenes del editor por categorías.

La opción **Ejecutar**, está compuesta por tres formas de ejecución: **Automática**, **Interactiva** o **Manual**. Estas opciones se explicará su funcionamiento más adelante

La opción **Mensajes** muestra los errores de sintaxis que ocurrieron en la última compilación, como se muestra en la figura 3-11. Si el número de errores sobre pasa la capacidad del área de mensajes es posible desplazarse a la siguiente ventana utilizando las flechas de desplazamiento ↓,↑y para salir presione ESC.

En está modalidad se cuenta también con un sistema de ayuda en línea, la cual se activa pulsando la tecla de función F1. Es sensible al contexto, lo que significa que visualiza la información que está relacionada con lo que se está haciendo en ese momento. Para salir del sistema de ayuda se debe pulsar la tecla ESC o Cancelar.

III.3.1 Ejecución modo Automático e Interactivo.

Cuando se haya concluido con la escritura del programa y después de la compilación el programa se encuentre excento de errores es posible ejecutar lo de dos maneras, en forma Automática o Interactiva. La Figura 3-12 muestra la pantalla de ejecución en modo Automático. Para el modo Interactivo la pantalla de ejecución es similar y la barra del editor donde se encuentra el cursor se encuentra activada.

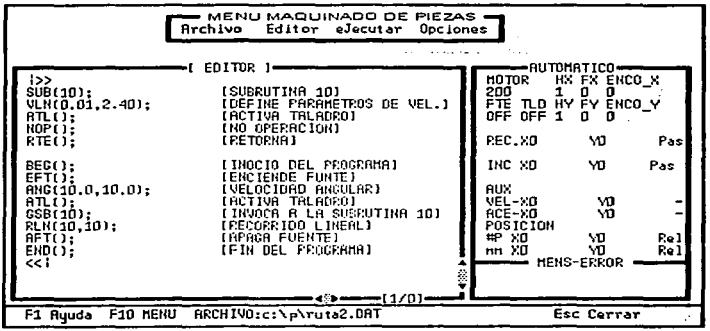


Figura 3-12. Pantalla de ejecución en modo Automático.

El modo **Automático** ejecuta el programa conforme se encuentren las instrucciones en el mismo, es decir empezará con la sentencia BEG() y terminará con END(). Mientras que por el método **Interactivo** la primera sentencia que se debe ejecutar es la de inicio BEG(), las demás instrucciones se pueden ejecutar en forma indistinta, es decir con sólo seleccionar la sentencia por medio de las flechas de desplazamiento y presionar ENTER se ejecutará esa sentencia. En estos dos métodos, algunas instrucciones pueden enviar información la cual se podrá visualizar en el área de mensajes.

En el modo **Automático e Interactivo** es posible ejecutar una pausa mediante una instrucción en el programa o a través del interruptor de paro que se encuentra en el rac de control. Para reanudar el movimiento puede presionar conjuntamente las teclas ALT-C o por medio del interruptor que se encuentra en el rac de control que dice R-IN.

En forma Interactiva se puede continuar el movimiento por medio de la instrucción CNT(), ya que en este caso las instrucciones se van ejecutando conforme lo requiera el usuario. En todo momento se encuentra activada la opción suspender por medio de las teclas ALT-S. Cuando se termina de ejecutar un programa en forma **Automática** regresa al menú principal. Mientras que para terminar con la ejecución de un programa en forma **Interactiva** se tiene que ejecutar le última sentencia del programa END() y regresa al menú principal.

III.3.2. Ejecución modo Manual.

Cuando se requiera maquinar una pieza y no se desee hacerlo por medio de un programa, se puede ejecutar el sistema en forma **Manual**. La opción **Manual** accesa a una pantalla como lo muestra la Figura 3-13, en la cual existen opciones que se pueden modificar con sólo presionar la tecla ALT y la letra mayúscula resaltada o por medio del Mouse. Está pantalla está compuesta por: **Simulador de operación del taladro, Simulador de desplazamiento de los ejes, Área de mensajes, Ventanas de información y Ventanas para modificar los datos.**

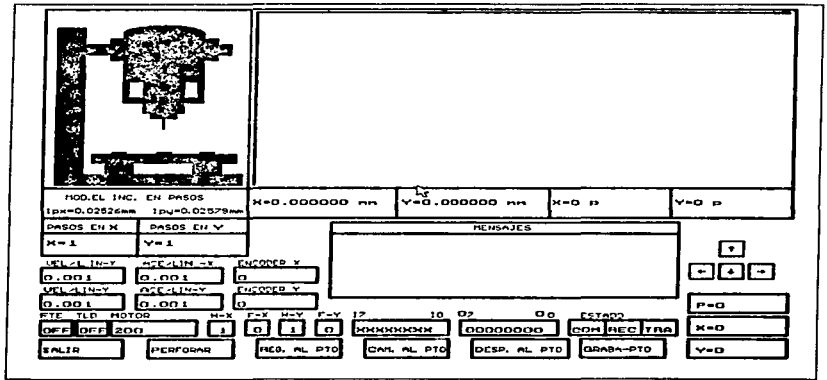


Figura 3-13. Pantalla de ejecución en modo Manual.

Al ingresar a está modalidad el sistema mandará un mensaje que indica que está inicializando la mesa, es decir, desplaza la plataforma al inicio. Cuando termina desaparece el mensaje y en ese momento se puede iniciar con la ejecución.

Antes de desplazar la plataforma de la mesa se tiene que encender la fuente. Cuando se efectúa algún desplazamiento se dan a elegir dos opciones en el área de mensajes: **Pausa** y **Cancelar**. Las cuales están definidas por las tecla de función **F7** y **ESC** respectivamente. Si se presiona la tecla de función **F7** se detiene la mesa para continuar con el desplazamiento sólo basta con presionar la tecla de función **F8** que corresponde a la opción **Continuar** y que se muestra en el área de mensajes en sustitución de la opción pausa, la tecla **ESC** se emplea para cancelar el desplazamiento de la plataforma de la mesa. Las opciones de **Parar** y **Continuar** pueden activarse también por medio del interruptor que se encuentra en el Rac de control. La Tabla 3-4, muestra las opciones ejecutar y modificar los datos.

TECLAS	OPERACIÓN
Ordenes de ejecución.	
ALT-P	PERFORA
ALT-S	SALIR
ALT-G	GRABAR PUNTO
ALT-D	DESPLÁZATE AL PUNTO
ALT-F	ENCIENDE O APAGA LA FUENTE
ALT-C	CAMBIA AL SIGUIENTE PUNTO
ALT-R	REGRESA AL PUNTO ANTERIOR
Ordenes de modificación de datos.	
ALT-M	MODIFICA LA RESOLUCIÓN DEL MOTOR
ALT-L	MODIFICA LA VELOCIDAD LINEAL EN X
ALT-V	MODIFICA LA VELOCIDAD LINEAL EN Y
ALT-N	MODIFICA LA ACELERACIÓN LINEAL EN Y
ALT-V	MODIFICA LA ACELERACIÓN LINEAL EN Y
ALT-X	MODIFICA EL NÚMERO DE PASOS EN X
ALT-Y	MODIFICA EL NÚMERO DE PASOS EN Y
ALT-O	MODIFICA LA SALIDA

La Tabla 3-4

La opción **Salir** se encuentra activa en todo momento ya que si ocurriera alguna eventualidad en el microcontrolador se podría suspender y regresar al menú de Maquinado de Piezas.

III.4 Diseño de programas.

El lenguaje de programación que se describe a continuación es relativamente sencillo, está compuesto por una gran variedad de instrucciones que realizan diferentes tareas. Las instrucciones se ejecutan en la secuencia en que se leen. Durante la ejecución de cada sentencia es posible que algunos datos puedan ser exhibidos en la pantalla o que algunas sentencias se repitan varias veces.

Este lenguaje cuenta con instrucciones o sentencias parecidas a las funciones utilizadas en otros lenguajes, aunque no se efectúan cálculos y mucho menos entregan resultados, sino que ejecutan una operación específica. Estas sentencias pueden o no necesitar algunos parámetros que establezcan o modifiquen las condiciones de operación del sistema. Se cuenta con una sentencia de control que se encarga de repetir un conjunto de instrucciones dentro del programa principal a la cual también se le puede denominar bucle. Por último, este lenguaje soporta la creación de bloques de código independientes del conjunto de sentencias del programa principal, a este tipo de bloques de código se le llaman subrutinas.

Aunque el término de lenguaje estructurado, no se le aplica estrictamente a este lenguaje, si es necesario que se respeten algunas condiciones en el desarrollo de los mismos. Ejemplo:

```
SUB(#);
-------
-------      DECLARACIÓN DE SUBRUTINAS
-------
-------
RTE();

BEG();
--------
--------     PROGRAMA PRINCIPAL
--------
END();
```

Como se muestra en el ejemplo anterior primero se debe escribir las subrutinas y después el programa principal, está estructura es similar a la utilizada en otros lenguajes de programación.

El programa principal está delimitado por las sentencias BEG() y END(), las cuales establecen el inicio y el fin del programa respectivamente, todas las instrucciones ejecutables o instrucciones que llevan a cabo el manejo de datos en curso pasan a ser parte de la secuencias de sentencias. Ejemplo:

```
BEG ( );  Inicio del programa
---------
---------  Secuencia de sentencias
---------
END ( )   Fin del programa
```

Las instrucciones deben finalizar con un punto y coma (;) ya que éste se utiliza como separador entre instrucciones debido a que se puede escribir más de una sentencia en una línea. Entre la secuencia de sentencias no deben existir líneas en blanco.

Las sentencias pueden ir precedidas con comentarios que describan el propósito del módulo y que ofrezcan mayor información que sea de interés para su mejor comprensión. Un comentario no puede ser escrito sin encontrarse antes una sentencia. En este caso los comentarios se encontrarán delimitados por corchetes [ ], que establecen el inicio y el fin del comentario. El comentario es ignorado por el compilador durante la traducción del programa, pero no durante la revisión de sintaxis, esto significa que el declarar mal un comentario incurriría en un error de sintaxis. Ejemplo:

```
BEG();
   -------
   -------
   -------
   EFT();                  [Enciende la fuente]
   -------
   -------
   -------
   ANG(12.34,78.90);       [Modifica la aceleración angular]
   GHM();                  [Desplaza los ejes al inicio]
   --------
    -------
   --------
   END();
```

III.4.1 Estructura de control.

La estructura de control es un bucle que nos permite repetir un conjunto de sentencias, hasta que se cumple cierta condición. El bucle JLP/FLP, realiza la misma función que el bucle FOR utilizado en otros lenguajes de programación. La sintaxis de está estructura es la siguiente:

```
JLP(#,N);     INICIO DEL BUCLE
---------
--------- SECUENCIA DE SENTENCIAS
---------
FLP(#);       FIN DEL BUCLE
```

DONDE: #: Especifica el número del bucle
N: Indica el número de repeticiones.

Cuando se declare algún bucle será necesario especificar el número del bucle y el número de veces que se ejecutarán dichas secuencias, no es posible que existan 2 bucles con el mismo número de especificación. En este lenguaje sólo se permiten declarar 100 bucles en todo el programa, y cada bucle sólo podrá realizar 1000 repeticiones. Es posible contar con la técnica de incluir uno o más bucles dentro de otro, a está técnica se le llama anidamiento, hay pautas definidas para crear bucles anidados y deberá tener cuidado cuando las utilice sin embargo no es difícil, en seguida se muestra un ejemplo de anidamiento de bucles.

```
JLP(1,10);
-----------
  JLP(2,7);
  -----------
    JLP(4,8);
    ---------
    ---------
    FLP(4);
  -------
  -------
  FLP(2);
 ----------
FLP(1);
```

Cuando se utilice está técnica debe cuidar que un bucle esté debidamente incluido dentro otro, esto evitara incurrir en un error.

III.4.2 Subrutinas.

En ciertas ocasiones, hay que realizar una tarea especifica varias veces, durante la ejecución del programa, esto puede resulta demasiado complicado para el bucle JLP/FLP. En este caso, sería mejor incluir una subrutina en el programa.

Una subrutina es un bloque de código independiente del programa principal el cual puede llamarse desde cualquier punto o parte del programa, prácticamente cuando se invoca o se ejecuta una subrutina, ésta realiza una tarea y después vuelve al programa principal una sentencia después de donde fue invocada.

Este lenguaje delimita una subrutina por medio de las sentencias SUB() y RTE(). La sentencia que hace la llamada a la subrutina es GSB(). A continuación se muestra un ejemplo de cómo se declara una subrutina. Ejemplo:

```
SUB(3);    INICIO DE LA SUBRUTINA 3
-----
------     SECUENCIA DE SENTENCIAS
------
-------
RTE();     FIN DE LA SUBRUTINA 3

BEG();
--------
--------
--------
GSB(3);    LLAMA A LA SUBRUTINA 3
--------
--------
--------
END();
```

La sentencia GSB(3) llama a la subrutina definida con el número 3 una vez que se ejecutaron las sentencias contenidas en la subrutina se regresa al programa principal por medio de la sentencia RTE(), una sentencia después donde fue llamada la subrutina.

Es posible que una subrutina llame a otra pero no se cuenta con la posibilidad de utilizar la técnica de recursividad, es decir que una subrutina se llame así misma. Dentro de las subrutinas no es posible dejar líneas en blanco.

III.4.3 Sentencias.

Las sentencias están definidas únicamente por tres caracteres y un par de paréntesis en los cuales se pueden escribir algunos parámetros aunque en algunos casos no son necesarios. Todas las sentencias se deben escribir con mayúscula.

INSTRUCCIÓN(<<Parámetros>>);

A continuación se presentan todas las sentencias con las cuales se puede programar la mesa X-Y.

SENTENCIA	DESCRIPCIÓN	PARÁMETROS	UNIDADES
ANG(X,Y)	ACELERACIÓN ANGULAR	0.01< X >999.99 0.01< Y >999.99	m/s²
ALN(X,Y)	ACELERACIÓN LINEAL	0.01< X >999.99 0.01< Y >999.99	m/s²
RNG(±X,±Y)	RECORRIDO ANGULAR	-20000< X >20000 -20000< Y >20000	pasos
RLN(±X,±Y)	RECORRIDO LINEAL	-435< X >435 -215< Y >215	mm
MVA(±X,±Y)	MOVIMIENTO ABSOLUTO	-20000< X >20000 -20000< Y >20000	pasos
MVR(±X,±Y)	MOVIMIENTO RELATIVO	-20000< X >20000 -20000< Y >20000	pasos
RMV()	REPITE MOVIMIENTO		
CDI()	CAMBIA DIRECCIÓN EN X y Y		
CDX()	CAMBIA DIRECCIÓN EN X		
CDY()	CAMBIA DIRECCIÓN EN Y		
AHF(N)	ACELERACIÓN A HOME	0.01< N >999.99	m/s²
VHF(N)	VELOCIDAD A HOME	0.001< N >20.00	m/s
GHM()	IR A HOME		
GEN()	IR A FIN		
VAN(X,Y)	VELOCIDAD ANGULAR	0.001< X >20.00 0.001< Y >20.00	m/s
VAN(X,Y)	VELOCIDAD LINEAL	0.001< X >20.00 0.001< Y >20.00	m/s
GAN()	EJECUTA MOVIMIENTO ANGULAR		
GLN()	EJECUTA MOVIMIENTO LINEAL		
MAN(±X,±Y)	EJECUTA MOVIMIENTO ANGULAR	-20000< X >20000 -20000< Y >20000	pasos
NOP()	NO OPERACIÓN		
RMD(#)	RESOLUCIÓN DEL MOTOR	50< # >20000	pasos
RIP()	INICIALIZAR PROCESO		
EFT()	ENCIENDE FUENTE		
AFT()	APAGA FUENTE		
GSB(#)	BRINCA A SUBRUTINA	0 < # > 100	
SUB(#)	SUBRUTINA	0 < # > 100	
RTE()	RETORNO DE SUBRUTINA		
JLP(#,N)	INICIO DE CICLO	0 < # > 100 0 < N > 1000	
FLP(#)	FIN DE CICLO	0 < # > 100	
LBF()	LIMPIA BUFFER		
ILX(#,±N)	MOV. LINEAL 'X' EN INCREMENTOS	0 < # > 100 -20000 < N > 20000	N=pasos
ILY(#,±N)	MOV. LINEAL 'Y' EN INCREMENTOS	0 < # > 100 -20000 < N > 20000	N=pasos
ITL()	INTERRUPCIÓN TOTAL		
IPS()	PAUSA		
CNP()	CONTINUA		
ATL()	ACTIVA TALADRO		
IPX(#,±N)	MOV. PASOS 'X' EN INCREMENTOS	0 < # > 100 -20000 < N > 20000	N=pasos
IPY(#,±N)	MOV. PASOS 'Y' EN INCREMENTOS	0 < # > 100 -20000 < N > 20000	N=pasos

SENTENCIA	DESCRIPCIÓN	PARAMETROS	UNIDADES
BEG()	INICIO DE PROGRAMA		
END()	FIN DE PROGRAMA		
PEA()	POSICIÓN ABS. CODIFICADOR X,Y		
PER()	POSICIÓN REL. CODIFICADOR X,Y		
INP(#)	ENTRADA BINARIA	# = BYTE	
OUT(#)	SALIDA BINARIA	# = BYTE	
LHX()	LIMITE HOME X		
LHY()	LIMITE HOME Y		
LFX()	LIMITE END X		
LFY()	LIMITE END Y		
EPP()	ENVÍA POSICIÓN PASOS		
WIN(#)	ESPERA ENTRADA BINARIA	# = BYTE	
MPT(#)	MUÉVETE AL PUNTO	0 < # > 1000	
MAP(#)	MUÉVETE A TRAVÉS DEL PUNTO	0 < # > 1000	
PNT(#)	PUNTO	0 < # > 1000	
DLY(#)	RETARDO	1 < # > 999.999	
ESC(#)	ESC(#)	0.001 < # > 1000	
ATC():	ACTIVA TALADRO CONTINUAMENTE		
PTL()	APAGUE TALADRO		
CMX()	CALCULA MOVIMIENTO 'X' EN BASE A VELOCIDAD, ACELERACIÓN Y POSICIÓN		
CMY()	CALCULA MOVIMIENTO 'Y' EN BASE A VELOCIDAD, ACELERACIÓN Y POSICIÓN		
MVL(±X,±Y)	EJECUTA MOVIMIENTO LINEAL	-20000< X >20000 -20000< Y >20000	pasos

III.4.4 Mensajes de error.

Este programa fue diseñado para detectar cualquier cosa que no sea sintácticamente correcta. Para esto el compilador revisa la sintaxis del programa si llegara a encontrar uno o más errores el compilador detiene su ejecución y abre una ventana, en la cual mostrará un máximo de 50 errores.

Está ventana se puede cerrar con sólo presionar un ESC, si se desea ver los demás errores se deben presionar las flechas de desplazamiento arriba o abajo para desplazarse a la siguiente ventana.

Está ventana mostrará en que línea aparecen el o los errores y el mensaje del posible error que se generó, a continuación se muestra la Tabla 3-5 con los mensajes de error que pueden aparecer y el significado de cada mensaje.

MENSAJES	DESCRIPCIÓN
?? BEG();/END();	Error en las sentencias de inicio y fin.
?? SUB();/RET();	Error en las sentencias de la subrutina
?? JLP();/FLP();	Error en las sentencias del bucle
NO HAY RECURSION	Error una subrutina no se puede llamar así misma.
? ESTRUCTURA LOOP'S	Error mal estructurado el bucle.
?? FLP();...JLP();	Se detecto un bucle que cierra y luego abre.
SE REPITE FLP()	Error se repite un fin del bucle.
SE REPITE JLP()	Error se repite un bucle
?? SE REPITE SUB	Error se repite una subrutina.
?? NO EXISTE GSB(?)	Error no existe la subrutina.
?? PROG. MAL ESTRUC	Error el programa se encuentra mal estructurado.
?? JLP(?);...FPL(?);	Error en el argumento del bucle.
?? BEG();...END()	Error de sintaxis al declarar el programa principal.
?? JLP();..FLP()	Error de sintaxis al declarar la estructura de control.
?? SUB();...RET();	Error de sintaxis al declarar la una subrutina.
CARAC. INDEF.	Error, existen caractere indefinidos.
ARG(?)	Error en el argumento.
SINTAXIS();	Error de sintaxis.
MENSAJE[]?	Error de sintaxis en el mensaje.
? INSTRUC	Error, no reconoce la sentencia.
RANGO INDEFINIDO	Error en el parámetro de la instrucción.

Tabla 3-5 de mensajes de error.

CAPÍTULO

IV

DESARROLLO DE RUTINAS EN ENSAMBLADOR 8051

Utiliza un lenguaje ordinario y di cosas extraordinarias

Arthur Schopenhauer

IV Plan de desarrollo

Las rutinas que se describen a continuación se desarrollaron sólo para comprobar el funcionamiento de los módulos que se encargan de establecer la comunicación entre la PC y el microcontrolador y que forman parte del software diseñado para el control del equipo de desarrollo X-Y, el cual se llevo acabo en el lenguaje de programación "C". Estas rutinas se escribieron en el lenguaje ensamblador 8051, y pueden ser la base para el programa que se encargue de realizar todas las tareas solicitadas en el equipo X-Y, el cual forma parte de la segunda etapa del proyecto, y que se llevarán acabo por medio del desarrollo de otra tesis de licenciatura.

Estas rutinas pueden parecer un tanto sencillas, debido a que sólo se simulan por medio de retardos las operaciones de desplazamiento o de perforación, así como también, sólo reflejan algunos datos enviados de la PC al microcontrolador, es decir, los datos transmitidos por la PC son retornados por el microcontrolador. Aunque el sistema contenido en la PC está diseñado para responder a alguna eventualidad, el programa de prueba no contiene las rutinas necesarias que tomen las alternativas o la decisiones de los caminos y las acciones que se deberán tomar en el momento que ocurra alguna situación inesperada.

Las rutinas están distribuidas en una serie de módulos llamadas subrutinas las cuales son invocadas por medio del mnemónico **ACALL** y cada una finalizará con el mnemónico **RET**, esto hace más fácil su identificación y por lo consiguiente su modificación o corrección. También al inicio se declaran una serie de constantes con los valores en hexadecimal de los caracteres empleados para la ejecución de alguna tarea, así como también para la transmisión y recepción de datos, esto con el objetivo de identificar el tipo de operación de cada una de las rutinas.

```
;************** DECLARACIÓN DE CONSTANTES **************

MAN   EQU  3AH  ; CARACTER PARA EL MODO MANUAL
AUT        EQU  3BH  ; CARACTER PARA EL MODO AUTOMATICO
INT        EQU  3CH  ; CARACTER PARA EL MODO INTERACTIVO
TAR        EQU  3DH  ; CARACTER PARA MODO PERFORADO DE TARJETAS
COMM EQU  3FH  ; CARACTER ENTABLA COMUNICACION
MOT   EQU  15H  ; CARACTER DEL MOTOR
OK0        EQU  4FH   ; TRANMITE EL SIGUIENTE DATO
SIG        EQU  53H   ; TRANMITE EL SIGUIENTE DATO
LIST  EQU  51H  ; TRANMITE EL SIGUIENTE DATO
```

```
OK1          EQU  4BH  ; TRANMITE EL SIGUIENTE DATO
ARR          EQU  40H  ; EJECUTANDO
PER          EQU  1FH  ; PERFORAR
EFT          EQU  17H  ; ENCIENDE FUENTE
AFT          EQU  18H  ; APAGA FUENTE
PX           EQU  4AH  ; CAMBIA EL PARAMETRO DEL EJE X
PY           EQU  4BH  ; CAMBIA EL PARAMETRO DEL EJE Y
DDX    EQU   42H  ; DESPLAZATE DERECHA EN EL EJE X
DIX          EQU  45H  ; DESPLAZATE IZQUIERDA EN EL EJE X
DAY    EQU   44H  ; DESPLAZATE ARRIBA EN EL EJE Y
DBY    EQU   43H  ; DESPLAZATE ABAJO EN EL EJE Y
SAL          EQU  4CH  ; SALIR DEL DEL MODO
GP           EQU  47H  ; GRABA UN PUNTO
DP           EQU  30H  ; DESPLAZATE AL PUNTO
OUT          EQU  28H  ; ENVIA DATO BINARIO/SALIDADE UN DATO
ALY          EQU  49H  ; ENVIA DATO ACELERACION LINEAL  Y
ALX          EQU  59H  ; ENVIA DATO ACELERACION LINEAL  X
VLX          EQU  5AH  ; ENVIA DATO VELOCIDAD LINEAL  X
VLY          EQU  48H  ; ENVIA DATO VELOCIDAD LINEAL  Y
TRANS EQU    3EH  ; TRANSMITIR UN DATOS O INFORMACION ">"
ESP          EQU  57H  ; ESPERA
DNX    EQU   6AH  ; COORDENADA EN X
DNY    EQU   6BH  ; COORDENANA EN Y
INI          EQU  55H  ; INICIALIZAR
STOP   EQU   1DH  ;PAUSA
CNT          EQU  1EH  ;CONTINUAR
```

El programa inicia en la dirección 2100H como se muestra en el mnemónico **ORG**, para después brincar a la rutina **PRECEP**, en donde se encuentra la estructura principal . La rutina **DELAY1** se emplea para efectuar un retardo para que no se rompa la comunicación entre la PC y el microcontrolador.

```
;*  INICIO DEL PROGRAMA  **

ORG  2100H

RD1   AJM PRECEP
      LJMP 2100H
      ORG  2140H
```

```
;********** RATARDO *****
DELAY1:
      PUSH  ACC
      PUSH  00
      MOV   PSW,#00H
      MOV   R2,#31H
DD3 MOV   R1,#63H
DD1 DEC   A
      CJNE  A,#00H,DD1
      DEC   R1
      CJNE  R1,#00H,DD2
      POP   00
      POP   ACC
      RET
```

IV.1 Estructura principal.

Antes de iniciar la ejecución de alguna modalidad es imprescindible que se verifique, que tanto la PC como el microcontrolador están listos para efectuar la operación de transmisión y recepción de los datos, esto se logra mediante la detección del caracter (?) o símbolo de interrogación que se debe de transmitir por ambas partes.

Debido a que el lenguaje ensamblador no cuenta con alguna estructura de control, el diseño de una estructura de decisión múltiple para el acceso a cualquier modalidad, se construyó empleando las instrucciones: **JMP** y **CJMP** . La primera, efectúa un salto a una etiqueta y la segunda, realiza una comparación, si la respuesta es verdadera continua con la siguiente instrucción, pero en el caso en que la respuesta sea falsa realiza un brinco a la etiqueta especificada.

Al inicio de está estructura se transmite el caracter (?) y después lee el puerto, si el dato que leyó es el mismo que transmitió no efectuara ningún proceso y se mantendrá en un ciclo hasta que dicho dato sea diferente. Cuando el dato leído sea diferente del signo de interrogación entonces se realizara una serie de comparaciones y brincos hasta que alguna comparación proporcione una respuesta verdadera, esto provocara la ejecución de una serie de rutinas que permitirán el acceso y ejecución de alguna modalidad, la cual finalizara con un salto al inicio de la estructura.

Cuando el dato leído no sea igual con ninguno de los datos comparados, tendrá que regresar al inicio debido a que el dato transmitido por la PC no corresponde a ninguno de los esperados. Cuando se de por terminada la sesión o ejecución de alguna modalidad, se regresará a la estructura principal para poder accesar a alguna otra modalidad que se desee, siempre y cuando no haya ocurrido alguna eventualidad que pudiese romper con la secuencia lógica de este programa de prueba. Las eventualidades se pueden definir como la interrupción de la secuencia de transmisión de las señales establecidas entre el microcontrolador y el equipo X-Y, así como también en la PC, para la realización de una tarea. Sería imposible hacer mención de todas las eventualidades que pudiesen surgir durante la ejecución del sistema, debido a que una eventualidad se puede originar desde una falla en alguno de los sensores hasta la suspención de energía en algún módulo.

Por lo consiguiente cuando surja alguna eventualidad el microcontrolador debe de informar del suceso a la PC, transmitiendo el caracter de error, que la PC lo relacionará con la tarea que solicito ejecutar desplegando en la pantalla un mensaje de error.

```
RECEP:
      SETB P3.5              ;
      MOV SP,#30H            ;
      MOV SBUF,#COMM         ; COLOCA EN EL PUERTO SERIE EL CAR. '?'
      MOV       PSW,#00H     ; SE HACE REFERENCIA AL BANCO CERO
      LCALL     DELAY2       ; RETARDO

INICIO:
      MOV SBUF,#COMM         ; COLOCA EN EL PUERTO SERIE EL CAR. '?'
      LCALL     DELAY1       ; RETARDO
      CLR       SCON.0       ; LIMPIA
      MOV A,SBUF     ; MUEVE EL CONT. DEL PUERTO AL ACUMULADOR
      LCALL     DELAY1       ; RETARDO
      CJNE A,#COMM,COMP1 ; SE MANTIENE EN COMUNICACION
      JMP       INICIO       ; REGRESA A INICIO

COMP1
      CJNE A,#MAN,COMP2      ; COMPARA Y BRINCA A COMP2 SI A= MAN
      LCALL     INIREG       ; INICIALIZA LOS REGISTROS
      LCALL     ACCESAR      ; MÓDULO PARA ACCESAR AL MODO MANUAL
      LCALL     INICMESA ; MÓDULO PARA INICIALIZAR EL EQUIPO
      LCALL     INIPANT  ; INICIALIZA EN LA PANTALLA DE LA PLATAFORMA
      LCALL     MMAN         ; LLAMA AL MODO MANUAL
      JMP       INICIO       ; REGRESA A INICIO

COMP2
      CJNE A,#AUT,COMP3      ; COMPARA Y BRINCA A COMP3 SI A= AUTO
      LCALL     INIREG       ; INICIALIZA LOS REGISTROS
      LCALL     ACCMAQ       ; ACCESA AL MÓDULO AUTOMATICO
      LCALL     INICMAQ      ; INICIALIZA EL EQUIPO
      LCALL     MUAT         ; LLAMA MODO AUTOMATICO
      JMP       INICIO       ; REGRESA A INICIO

COMP3
      CJNE A,#INT,COMP4      ; COMPARA Y BRINCA A COMP4 SI A= INT
      LCALL     INIREG       ; INICIALIZA LOS REGISTROS
      LCALL     ACCMAQ       ; ACCESA AL MÓDULO INTERACTIVO
      LCALL     INICMAQ      ; INICIALIZA EL EQUIPO
      LCALL     MUAT         ; LLAMA MODO INTERACTIVO
      JMP       INICIO       ; REGRESA A INICIO

COMP4
      CJNE A,#TAR,INICIO     ; COMPARA Y BRINCA A COMP3 SI A= AUTO
      LCALL     INIREG       ; INICIALIZA LOS REGISTROS
      LCALL     ACCTAR       ; ACES0 AL MÓDULO PERFORADO DE TARJETAS
      LCALL     INICMESA     ; INICIALIZA EL EQUIPO
      LCALL     MTAR         ; LLAMA A MODO PERFORADO DE TARJETAS
      JMP       INICIO       ; REGRESA A INICIO
```

IV.2 Rutinas de prueba para la ejecución Perforado de Tarjetas.

El acceso a la modalidad perforado de tarjeta se lleva acabo por medio de las rutinas **INIREG**, **ACCTAR**, **INICMESA** y **MTAR**. Las rutinas **INIREG** e **INICMESA** ya fueron descritas con anterioridad. La rutina **ACCTAR**, realiza la recepción de los datos de velocidad y aceleración lineal. Por el momento sólo se lee el dato correspondiente a X, pero cuando se desarrolle el programa origina, el valor de X también se le asignará a Y. Para finalizar se debe transmitir la instrucción que encienda la fuente para iniciar la sesión de perforado. La rutina **MTAR**, es una estructura de decisión múltiple que permite elegir cualquiera de las opciones disponibles en está modalidad.

```
ACCTAR:
    MOV  SBUF,#OK1
    LCALL DELAY1
    CLR  SCON.0
    MOV  A,SBUF
   CJNE A,#MOT,ACCTAR
    LCALL DELAY1
    MOV  SBUF,#OK0
    LCALL DELAY1
    MOV  SBUF,#SIG
    LCALL DELAY1
    MOV  SBUF,#COMM
    LCALL DELAY1
VELENT:
    MOV  SBUF,#COMM
    LCALL DELAY1
    CLR  SCON.0
    MOV  A,SBUF
    CJNE A,#VI,V,VELENT
      LCALL DELAY1
      MOV  SBUF,#OK0
      LCALL DELAY1
      MOV  SBUF,#SIG
      LCALL DELAY1
      MOV  SBUF,#OK0
      LCALL DELAY1
      MOV  SBUF,#SIG
      LCALL DELAY1
    MOV  SBUF,#COMM
ACEENT:
    MOV  SBUF,#COMM
      LCALL DELAY1
      CLR  SCON.0
      MOV  A,SBUF
    CJNE A,#ALY,ACEENT
      LCALL DELAY1
      MOV  SBUF,#OK0
      LCALL DELAY1
      MOV  SBUF,#SIG
      LCALL DELAY1
      MOV  SBUF,#OK0
      LCALL DELAY1
      MOV  SBUF,#SIG
    MOV  SBUF,#COMM
FUENTE:
    MOV  SBUF,#COMM
      LCALL DELAY1
      CLR  SCON.0
      MOV  A,SBUF
    CJNE A,#EFT,FUENTE
      LCALL DELAY1
      MOV  SBUF,#OK0
      LCALL DELAY1
    MOV  SBUF,#COMM
      RET
```

La rutina **MTAR**, sólo cuenta con tres opciones: inicializar la mesa, recepción de la coordenada y salida de la modalidad. Aunque existen otro tipo de opciones sólo quedaran disponibles cuando se inicie el proceso de desplazamiento o de transmisión de datos. La rutina **DT1** corresponde a la opción ejecutar y es la que se encarga de realizar la recepción de las coordenadas del punto a perforar, después simula el desplazamiento como se realizo en la anterior modalidad, al termino se realiza la perforación, la cual también es simulada de la misma manera que un desplazamiento, después regresa a la rutina **MTAR**, para recibir la siguiente instrucción que indica la operación que debe de ejecutar.

Durante el desplazamiento se puede ejecutar la opción PARAR, que dará la posibilidad de probar el funcionamiento de la opción CONTINUAR, debido a que dentro de la rutina **DT1**, se cuenta con una estructura de decisión múltiple en donde se encuentran integradas estas opciones.

```
DT1:
    MOV  SBUF,#OK0
    LCALL DELAY1
    MOV  SBUF,#SIG
    LCALL DELAY1
    MOV  SBUF,#OK0
    LCALL DELAY1
DT2:
    MOV  SBUF,#COMM
    CLR  SCON.0
    MOV  A,SBUF
    CJNE A,#DNY,DT2
    MOV  SBUF,#OK0
    LCALL DELAY1
    MOV  SBUF,#SIG
    LCALL DELAY1
    MOV  SBUF,#OK0
    LCALL DELAY1
    MOV  SBUF,#ARR
    MOV  R3,#00H
DESPTAR:
    MOV  SBUF,#ARR
    LCALL DELAY1
    CLR  SCON.0
    MOV  A,SBUF
    LCALL DELAY1
    CJNE A,#STOP,RUT7
    LCALL DELAY1
PAUSA:
    MOV  SBUF,#ESP
    LCALL DELAY1
    CLR  SCON.0
MOV  A,SBUF
    LCALL DELAY1
    CJNE A,#CNT,RUT8
    LCALL DELAY1
    JMP  DESPTAR
RUT8 CJNE A,#INI,RUT04
    LCALL DELAY1
    RET
RUT04  CJNE A,#SAL,PAUSA
    LCALL DELAY1
    RET
RUT7: CJNE A,#SAL,RUT9
      RET
RUT9:
      MOV  SBUF,#ARR
      INC  R3
      CJNE R3,#30H,DESPTAR
      LCALL DELAY1
PERTAR:
      MOV  SBUF,#COMM
      LCALL DELAY1
      CLR  SCON.0
      MOV  A,SBUF
      LCALL DELAY1
      CJNE A,#PER,PERTAR
      MOV  A,#00H
BUCLE1
      MOV  SBUF,#ARR
      INC  A
      CJNE A,#20H,BUCLE1
      LCALL DELAY1
      MOV  SBUF,#COMM

MTAR:
      MOV  SBUF,#COMM
      LCALL DELAY1
      CLR  SCON.0
      MOV  A,SBUF
      CJNE A,#COMM,RUT1
      LJMP MTAR
RUT1  CJNE A,#DNX,RUT2
      LCALL DELAY1
      LCALL DT1
      LCALL DELAY1
      CLR  SCON.0
      MOV  A,SBUF
      CJNE A,#COMM,RUT2
      JMP  MTAR
RUT2  CJNE A,#INI,RUT3
      LCALL DELAY1
      LCALL INICMESA
      LJMP MTAR
RUT3  CJNE A,#SAL,MTAR
      RET
```

IV.3 Rutinas de prueba para la ejecución Automático e Interactivo.

Las rutinas para la modalidad Automático e Interactivo, son similares y por lo consiguiente sólo se emplea una estructura de decisión múltiple denominada **MUAT**.

Para accesar a cualquiera de estas dos modalidades, se emplean las rutinas **INIREG**, **ACCMAQ** e **INIMAQ**. La primera rutina ya ha sido descrita con anterioridad. La rutina **ACCMAQ**, tiene la tarea de recepción de los datos con los que funcionara inicialmente, estos datos corresponden a la velocidad y aceleración lineal en X y en Y. Después se ejecuta la rutina **INIMAQ**, la cual debe desplazar la plataforma de la mesa al inicio, para después transmitir los datos necesarios que despliegue en la pantalla los datos recibidos en la rutina **ACCMAQ**, además debe informar en estado en el cual se encuentra la plataforma, después se accesa a la rutina **MUAT**.

```
ACCMAQ:
    MOV   SBUF,#OK1
    CLR   SCON.0
    MOV   A,SBUF
    CJNE  A,#MOT,ACCMAQ
    MOV   SBUF,#OK0
    MOV   SBUF,#SIG
    LCALL DELAY1
    MOV   SBUF,#COMM
VELINY:
    MOV   SBUF,#COMM
    CLR   SCON.0
    MOV   A,SBUF
    CJNE  A,#VLY,VELINY
    LCALL DELAY1
    MOV   SBUF,#OK0
    LCALL DELAY1
    MOV   SBUF,#SIG
    LCALL DELAY1
    MOV   SBUF,#OK0
MOV  SBUF,#SIG
    LCALL DELAY1
    MOV   SBUF,#COMM
VELINX:
    MOV   SBUF,#COMM
    LCALL DELAY1
    CLR   SCON.0
    MOV   A,SBUF
    CJNE  A,#VLX,VELINX
LCALL DELAY1
    MOV   SBUF,#OK0
    LCALL DELAY1
MOV  SBUF,#SIG
    MOV   SBUF,#OK0
    LCALL DELAY1
    MOV   SBUF,#SIG
    LCALL DELAY1
    MOV   SBUF,#COMM
ACELINY:
    MOV   SBUF,#COMM
    CLR   SCON.0
    MOV   A,SBUF
    CJNE  A,#ALY,ACELINY
    MOV   SBUF,#OK0
    LCALL DELAY1
    MOV   SBUF,#SIG
    LCALL DELAY1
    MOV   SBUF,#OK0
    MOV   SBUF,#SIG
    LCALL DELAY1
    MOV   SBUF,#COMM
ACELINX:
    MOV   SBUF,#COMM
    LCALL DELAY1
    CLR   SCON.0
    MOV   A,SBUF
    CJNE  A,#ALX,ACELINX
    MOV   SBUF,#OK0
    MOV   SBUF,#SIG
    LCALL DELAY1
    MOV   SBUF,#OK0
    LCALL DELAY1
    MOV   SBUF,#SIG
    MOV   SBUF,#COMM
    RET
INICMAQ:
    MOV   SBUF,#TRANS
    LCALL DELAY1
    CLR   SCON.0
    MOV   A,SBUF
    LCALL DELAY1
    CJNE  A,#53H,INICMAQ
LOOPM8:
    MOV   SBUF,#61H
    CLR   SCON.0
    MOV   A,SBUF
    LCALL DELAY1
    CJNE  A,#51H,LOOPM8
    LCALL DELAY1
LOOPM9:
    MOV   SBUF,#31H
    LCALL DELAY1
    CLR   SCON.0
    MOV   A,SBUF
    LCALL DELAY1
    CJNE  A,#OK1,LOOPM9
INICMAQ2:
    LCALL DELAY1
    MOV   SBUF,#TRANS
    LCALL DELAY1
    CLR   SCON.0
    LCALL DELAY1
    MOV   A,SBUF
    LCALL DELAY1
    CJNE  A,#53H,INICMAQ2
```

```
LOOPM82:
    MOV  SBUF,#62H
    CLR  SCON.0
    MOV  A,SBUF
    LCALL DELAY1
    CJNE A,#51H,LOOPM82
    LCALL DELAY1
LOOPM92:
    MOV  SBUF,#31H
CLR  SCON.0
    MOV A,SBUF
    LCALL DELAY1
    CJNE A,#OK1,LOOPM92
INICMAQ3:
    LCALL DELAY1
    MOV  SBUF,#TRANS
    LCALL DELAY1
    CLR  SCON.0
    LCALL DELAY1
    MOV  A,SBUF
    LCALL DELAY1
    CJNE A,#53H,INICMAQ3
LOOPM83:
    MOV  SBUF,#63H
    CLR  SCON.0
    MOV  A,SBUF
    LCALL DELAY1
    CJNE A,#51H,LOOPM83
    LCALL DELAY1
LOOPM93:
    MOV  SBUF,#30H
    LCALL DELAY1
    CLR  SCON.0
    MOV  A,SBUF
    CJNE A,#OK1,LOOPM93
INICMAQ4:
    LCALL DELAY1
    MOV  SBUF,#TRANS
    LCALL DELAY1
    CLR  SCON.0
    LCALL DELAY1
    MOV  A,SBUF
    CJNE A,#53H,INICMAQ4
LOOPM84:
    MOV  SBUF,#64H
    CLR  SCON.0
    MOV  A,SBUF
    LCALL DELAY1
    CJNE A,#51H,LOOPM84
LOOPM94:
    MOV  SBUF,#30H
    CLR  SCON.0
    MOV  A,SBUF
    CJNE A,#OK1,LOOPM94
INICMAQ5:
    MOV  SBUF,#TRANS
    LCALL DELAY1
    CLR  SCON.0
    LCALL DELAY1
    MOV  A,SBUF
    CJNE A,#53H,INICMAQ5
LOOPM85:
    MOV  SBUF,#6BH
    LCALL DELAY1
    CLR  SCON.0
    MOV  A,SBUF
    LCALL DELAY1
    CJNE A,#51H,LOOPM85
LOOPM95:
    MOV  SBUF,#00H
    LCALL DELAY1
    CLR  SCON.0
    MOV  A,SBUF
    LCALL DELAY1
    CJNE A,#52H,LOOPM95
LOOPM995:
    MOV  SBUF,#00H
    CLR  SCON.0
    MOV  A,SBUF
    LCALL DELAY1
    CJNE A,#54H,LOOPM995
LPM95:
    MOV  SBUF,#0C8H
    LCALL DELAY1
    CLR  SCON.0
    MOV  A,SBUF
    LCALL DELAY1
    CJNE A,#OK1,LPM95
INICMAQ6:
    LCALL DELAY1
    MOV  SBUF,#TRANS
    LCALL DELAY1
    CLR  SCON.0
    LCALL DELAY1
    MOV  A,SBUF
    CJNE A,#53H,INICMAQ6
LOOPM86:
    MOV  SBUF,#70H
    LCALL DELAY1
    CLR  SCON.0
    MOV  A,SBUF
    CJNE A,#51H,LOOPM86
LOOPM96:
    MOV  SBUF,#00H
    LCALL DELAY1
    CLR  SCON.0
    MOV  A,SBUF
    CJNE A,#52H,LOOPM96
LOOPM996:
    MOV  SBUF,#0FH
    LCALL DELAY1
    CLR  SCON.0
    MOV  A,SBUF
    LCALL DELAY1
    CJNE A,#54H,LOOPM996
LPM996:
    MOV  SBUF,#00H
    LCALL DELAY1
    CLR  SCON.0
    MOV  A,SBUF
    LCALL DELAY1
    CJNE A,#52H,LPM996
LPM96:
    MOV  SBUF,#0AH
    LCALL DELAY1
    CLR  SCON.0
    MOV  A,SBUF
    LCALL DELAY1
    CJNE A,#51H,LPM96
    LCALL DELAY1
LPM963:
    MOV  SBUF,#00H
    LCALL DELAY1
    CLR  SCON.0
    MOV  A,SBUF
    LCALL DELAY1
    CJNE A,#52H,LPM963
LPM936:
    MOV  SBUF,#0FH
    LCALL DELAY1
    CLR  SCON.0
    MOV  A,SBUF
    LCALL DELAY1
    CJNE A,#54H,LPM936
LOPM926:
    MOV  SBUF,#00H
    CLR  SCON.0
    MOV  A,SBUF
    LCALL DELAY1
    CJNE A,#52H,LOPM926
LOPM962:
    MOV  SBUF,#05H
    CLR  SCON.0
    MOV  A,SBUF
    LCALL DELAY1
    CJNE A,#OK1,LOPM962
NICMAQ7:
    MOV  SBUF,#TRANS
    LCALL DELAY1
    CLR  SCON.0
MOV  A,SBUF
CJNE A,#53H,INICMAQ7
```

```
LOOPM87:
    MOV  SBUF,#75H
    LCALL DELAY1
    CLR  SCON.0
    MOV  A,SBUF
    LCALL DELAY1
    CJNE A,#51H,LOOPM87
    LCALL DELAY1
LOOPM97:
    MOV  SBUF,#03H
    LCALL DELAY1
    CLR  SCON.0
    MOV  A,SBUF
    LCALL DELAY1
    CJNE A,#52H,LOOPM97
LOOPM997:
    MOV  SBUF,#0B0H
    LCALL DELAY1
    CLR  SCON.0
    MOV  A.SBUF
    LCALL DELAY1
    CJNE A.#54H,LOOPM997
```

```
LPM997:
    MOV  SBUF,#01H
    LCALL DELAY1
    CLR  SCON.0
    MOV  A,SBUF
    LCALL DELAY1
    CJNE A,#52H,LPM997
LPM97:
    MOV  SBUF,#00H
    LCALL DELAY1
    CLR  SCON.0
    MOV  A.SBUF
    LCALL DELAY1
    CJNE A,#51H,LPM97
    LCALL DELAY1
LPM973:
    MOV  SBUF,#02H
    LCALL DELAY1
    CLR  SCON.0
    MOV  A.SBUF
    LCALL DELAY1
    CJNE A.#52H,LPM973
```

```
LPM937:
    MOV  SBUF,#0A1H
    LCALL DELAY1
    CLR  SCON.0
    MOV  A,SBUF
    LCALL DELAY1
    CJNE A,#54H,LPM937
LOPM927:
    MOV  SBUF,#00H
    LCALL DELAY1
    CLR  SCON.0
    MOV  A.SBUF
    LCALL DELAY1
    CJNE A,#52H,LOPM927
LOPM972:
    MOV  SBUF,#00H
    LCALL DELAY1
    CLR  SCON.0
    MOV  A.SBUF
    LCALL DELAY1
    CJNE A.#0K1,LOPM972
    RET
```

Cuando en la rutina **MUAT**, reciba un dato que corresponda a la ejecución de una tarea se accesa a la rutina correspondiente la cual debe estar diseñada para saber si dicho dato viene acompañado con un parámetro y si este está compuesto por uno o dos datos.

```
MUAT:
    MOV  SBUF,#COMM
    CLR  SCON.0
    MOV  A.SBUF
    LCALL DELAY1
    CJNE A,#COMM,PA1
    JMP  MUAT
PA1 CJNE A,#22H,PA2
    LCALL DELAY1
PAA1
    MOV  SBUF,#SIG
    CLR  SCON.0
    MOV  A,SBUF
CJNE A,#54H,PAA1
    LJMP MUAT
PA2
    CJNE A,#17H,PA3
    LCALL EFM
    LJMP MUAT
PA3
    CJNE A,#18H,PA4
    LCALL VALA
    LJMP MUAT
```

```
PA4
    CJNE A,#1FH,PA5
    LCALL VALA
    LCALL DELAY1
    LJMP MUAT
PA5
    CJNE A,#01H,PA6
    LCALL VALA
    LJMP MUAT
PA6
    CJNE A,#02H,PA7
LCALL VALA
    LJMP MUAT
PA7
    CJNE A,#10H,PA8
    LCALL DELAY1
    LCALL VALA
    LJMP MUAT
PA8
    CJNE A,#0FH,PA9
    LCALL DELAY1
    LCALL VALA
    LJMP MUAT
```

```
PA9
    CJNE A,#28H,PA10
    LCALL DATO1
    LJMP MUAT
PA10
    CJNE A,#27H,PA11
    LCALL DATO1
    LJMP MUAT
PA11
    CJNE A,#SAL,PA12
    RET
PA12
    CJNE A,#23H,MUAT2
    LCALL DELAY1
PAA12
    MOV  SBUF,#SIG
    LCALL DELAY1
    CLR  SCON.0
    MOV  A,SBUF
    CJNE A,#54H,PAA12
    RET

MUAT2 LJMP  MUAT
```

Para las dos modalidades sólo se crearon unas cuantas rutinas que comprueban que los datos transmitidos por la PC se reciben correctamente, entre los cuales se encuentran: encender y apagar la fuente, activar el taladro, transmisión y recepción de un dato binario, transmisión de la velocidad y aceleración tanto angular como lineal.

```
DATOI:
    MOV  SBUF,#SIG
    CLR  SCON.0
    MOV  A,SBUF
    CJNE A,#54H,DATOUT
    MOV  SBUF,#COMM
    MOV  PSW,#08H
    CLR  SCON.0
    MOV  A,SBUF
    MOV  R0,A
    LCALL DELAY1
    MOV  SBUF,#SIG
    CLR  SCON.0
    MOV  A,SBUF
    MOV  R1,A
I10:
    MOV  SBUF,#TRANS
    LCALL DELAY1
    CLR  SCON.0
    LCALL DELAY1
    MOV  A,SBUF
    LCALL DELAY1
    CJNE A,#53H,I10
C90:
    MOV  SBUF,#67H
    CLR  SCON.0
    MOV  A,SBUF
    LCALL DELAY1
    CJNE A,#51H,C90
C91:
    MOV  A,R0
    MOV  SBUF,A
    LCALL DELAY1
    CLR  SCON.0
    MOV  A,SBUF
    LCALL DELAY1
    CJNE A,#52H,C91
C92:
    MOV  A,R1
    MOV  SBUF,A
    LCALL DELAY1
    CLR  SCON.0
    MOV  A,SBUF
    LCALL DELAY1
    CJNE A,#4BH,C92
    RET
EAFM:
    MOV  SBUF,#SIG
    CLR  SCON.0
    MOV  A,SBUF
    LCALL DELAY1
    CJNE A,#54H,AFM
AFM1:
    MOV  SBUF,#TRANS
    LCALL DELAY1
    CLR  SCON.0
    MOV  A,SBUF
    CJNE A,#53H,AFM1
AFM2:
    MOV  SBUF,#69H
    LCALL DELAY1
    CLR  SCON.0
    MOV  A,SBUF
    LCALL DELAY1
    CJNE A,#51H,AFM2
AFM3:
    MOV  SBUF,#30H
    LCALL DELAY1
    CLR  SCON.0
    MOV  A,SBUF
    LCALL DELAY1
    CJNE A,#4BH,AFM3
    RET
ATM:
    MOV  SBUF,#SIG
    LCALL DELAY1
    CLR  SCON.0
    MOV  A,SBUF
    CJNE A,#54H,AFM
ATM1:
    MOV  SBUF,#TRANS
    LCALL DELAY1
    CLR  SCON.0
    MOV  A,SBUF
    LCALL DELAY1
    CJNE A,#53H,ATM1
ATM2:
    MOV  SBUF,#6AH
    LCALL DELAY1
    CLR  SCON.0
    MOV  A,SBUF
    CJNE A,#51H,ATM2
ATM3:
    MOV  SBUF,#31H
    LCALL DELAY1
    CLR  SCON.0
    MOV  A,SBUF
    LCALL DELAY1
    CJNE A,#4BH,ATM3
    LCALL DELAY1
    MOV  A,#00H
ATM4
    MOV  SBUF,#ARR
    LCALL DELAY1
    INC  A
    CJNE A,#20H,ATM4

ATM5:
    MOV  SBUF,#TRANS
    LCALL DELAY1
    CLR  SCON.0
    MOV  A,SBUF
    LCALL DELAY1
    CJNE A,#53H,ATM5
ATM6:
    MOV  SBUF,#6AH
    LCALL DELAY1
    CLR  SCON.0
    MOV  A,SBUF
    LCALL DELAY1
    CJNE A,#51H,ATM6
ATM7:
    MOV  SBUF,#30H
    LCALL DELAY1
    CLR  SCON.0
    MOV  A,SBUF
    LCALL DELAY1
    CJNE A,#4BH,ATM7
    RET
```

```
VALA:
    MOV  SBUF,#SIG
    LCALL DELAY1
    CLR  SCON.0
    MOV  A,SBUF
    LCALL DELAY1
    CJNE A,#54H,ANM
    LCALL DELAY1
    MOV  SBUF,#COMM
    MOV  PSW,#00H
    LCALL DELAY1
    CLR  SCON.0
    MOV  A,SBUF
    MOV  R0,A
    LCALL DELAY1
    MOV  SBUF,#SIG
    CLR  SCON.0
    MOV  A,SBUF
    MOV  R1,A
    LCALL DELAY1
    MOV  SBUF,#COMM
    CLR  SCON.0
    MOV  A,SBUF
    MOV  R2,A
    LCALL DELAY1
    MOV  SBUF,#SIG
    CLR  SCON.0
    MOV  A,SBUF
    MOV  R3,A
    LCALL DELAY1
    MOV  SBUF,#COMM
    CLR  SCON.0
    MOV  A,SBUF
    MOV  R4,A
    LCALL DELAY1
    MOV  SBUF,#SIG
    CLR  SCON.0
    MOV  A,SBUF
    MOV  R5,A
    LCALL DELAY1
    MOV  SBUF,#COMM
    CLR  SCON.0
    MOV  A,SBUF
    MOV  R6,A
    LCALL DELAY1
    MOV  SBUF,#SIG
    CLR  SCON.0
    MOV  A,SBUF
    MOV  R7,A
    LCALL DELAY1
INICMAQ8:
    LCALL DELAY1
    MOV  SBUF,#TRANS
    LCALL DELAY1
    CLR  SCON.0
    LCALL DELAY1
    MOV  A,SBUF
    LCALL DELAY1
    CJNE A,#53H,INICMAQ8
LOOPM88:
    MOV  SBUF,#75H
    LCALL DELAY1
    CLR  SCON.0
    MOV  A,SBUF
    LCALL DELAY1
    CJNE A,#51H,LOOPM88
LOOPM98:
    MOV  A,R2
    MOV  SBUF,A
    LCALL DELAY1
    CLR  SCON.0
    MOV  A,SBUF
    LCALL DELAY1
    CJNE A,#52H,LOOPM98
LOOPM998:
    MOV  A,R0
    MOV  SBUF,A
    LCALL DELAY1
    CLR  SCON.0
    MOV  A,SBUF
    LCALL DELAY1
    CJNE A,#54H,LOOPM998
LPM998:
    MOV  A,R1
    MOV  SBUF,A
    LCALL DELAY1
    CLR  SCON.0
    MOV  A,SBUF
    LCALL DELAY1
    CJNE A,#52H,LPM998
LPM98:
    MOV  A,R3
    MOV  SBUF,A
    LCALL DELAY1
    CLR  SCON.0
    MOV  A,SBUF
    LCALL DELAY1
    CJNE A,#51H,LPM98
LPM983:
    MOV  A,R6
    MOV  SBUF,A
    LCALL DELAY1
    CLR  SCON.0
    MOV  A,SBUF
    LCALL DELAY1
    CJNE A,#52H,LPM983
LPM938:
    MOV  A,R5
    MOV  SBUF,A
    LCALL DELAY1
    CLR  SCON.0
    MOV  A,SBUF
    LCALL DELAY1
    CJNE A,#54H,LPM938
LOPM928:
    MOV  A,R4
    MOV  SBUF,A
    LCALL DELAY1
    CLR  SCON.0
    MOV  A,SBUF
    LCALL DELAY1
    CJNE A,#52H,LOPM928
LOPM982:
    MOV  A,R7
    MOV  SBUF,A
    LCALL DELAY1
    CLR  SCON.0
    MOV  A,SBUF
    LCALL DELAY1
    CJNE A,#OK1,LOPM982
    RET
```

Las rutinas **VALA**, **DATOI**, **EAFM** y **ATM**, comprueban el buen funcionamiento de recepción y transmisión de datos de sólo algunas sentencias las cuales se describen a continuación. La rutina **VALA**, se emplea para las sentencias **ANMO**, **ALNO**, **VLMO**, **VAMO**, las cuales cuentan con un argumento compuesto por dos datos que corresponden al eje X y al eje Y, dichos datos estan formados por una perte entera y una decimal.

La rutina **DATIO**, se emplea para las sentencias **OUTO** e **INPO**, las cuales cuentan con un argumento compuesto por un dato entero.

La rutina **EAFM**, se emplea para las sentencias **EFTO** y **AFTO**, las cuales no cuentan con un argumento. La rutina **ATM**, se emplea para la sentencia **ATMO**, la cual también no cuenta con un argumento pero está rutina simula el tiempo necesario que se tarda el taladro en realizar una perforación.

IV.4 Rutinas de prueba para la ejecución Manual.

Cuando en la estructura principal, el dato que se lea del puerto serie sea igual al dato en hexadecimal 3AH. se ejecutaran las rutinas **INIREG**, **ACCESAR**, **INICMESA**, **INIPANT** y **MMAN**, las cuales corresponden al acceso a la modalidad MANUAL.

La rutina **INIREG**. inicializa los registros del banco cero, está rutina es empleada en las otras tres modalidades. La rutina **ACCESAR**, se emplea para realizar el intercambio de caracteres que permitirán el acceso a la modalidad MANUAL, en está modalidad sólo se necesita recibir el dato correspondiente a la resolución del motor. La rutina **INICMESA**, es la que simula el tiempo que se necesita para llevar la plataforma de la mesa X-Y al inicio. La rutina **INIPANT**, modifica el estado en la pantalla de la plataforma. Por último la rutina MMAN, es una estructura decisión múltiple que se diseño a partir del mismo criterio con la que se construyó la estructura principal.

```
INICMESA:
      MOV       A,#00H
LOP2
      MOV       SBUF,#ESP
      INC       A
      CJNE      A,#0FFH,LOP2
      MOV       SBUF,#COMM
      RET
```

```
INIREG:
      MOV PSW,#00H
      MOV R0,#00H
      MOV R1,#31H
      MOV R2,#31H
      MOV R3,#30H
      MOV R4,#30H
      MOV R5,#00H
      MOV R6,#00H
      MOV R7,#00H
      RET
```

```
INIPANT:
      MOV       SBUF,#COMM
      MOV PSW,#00H
      MOV R1,#61H
      MOV R2,#31H
      ACALL DATOHFXY
      MOV PSW,#00H
      MOV R1,#62H
      MOV R2,#31H
      ACALL DATOHFXY
      MOV PSW,#00H
      MOV R1,#63H
      MOV R2,#30H
      ACALL DATOHFXY
      MOV PSW,#00H
      MOV R1,#64H
      MOV R2,#30H
      RET
```

```
ACCESAR:
      MOV       SBUF,#OK1
      LCALL     DELAY1
      CLR       SCON.0
      MOV       A,SBUF
      CJNE      A,#MOT,ACCESAR
      LCALL     DELAY1
      MOV       SBUF,#OK0
      LCALL     DELAY1
      MOV       SBUF,#SIG
      LCALL     DELAY1
      MOV       SBUF,#COMM
      LCALL     DELAY1
      RET
```

La estructura de decisión múltiple **MMAN**, permite simular la ejecución de algunas tareas de está modalidad. Aunque no se desarrollo una rutina para cada opción debido a que la secuencia de transmisión de caracteres para varias opciones es similar, entonces con una rutina se puede probar el buen funcionamiento de varias opciones. Como se muestra a continuación.

```
MMAN:
      MOV       SBUF,#COMM
      LCALL     DELAY2
      CLR       SCON.0
      MOV       A,SBUF
      CJNE      A,#COMM,COMP5
      LJMP      MMAN
COMP5
      CJNE      A,#PER,COMP6
      LCALL     DELAY1
      ACALL     PERDES          ; RUTINA QUE REALIZA UNA PERFORACION
      LJMP      MMAN
COMP6
      CJNE      A,#MAN,COMP7    ; INICIALIZA EL SISTEMA PARA MODIFICAR LA RESOLU-
      LCALL     DELAY1          ; CION DEL MOTOR
      RET
COMP7
      CJNE      A,#EFT,COMP8
      LCALL     DELAY1
      ACALL     EAFTE           ; RUTINA QUE ENCIENDE LA FUENTE
      LJMP      MMAN
COMP8
      CJNE      A,#AFT,COMP9
      LCALL     DELAY1
      ACALL     EAFTE           ; RUTINA QUE APAGA LA FUENTE
```

```
      LJMP        MMAN
COMP9
      CJNE        A,#PX,COMP10
      LCALL       DELAY1
      ACALL       CPXY          ; RUTINA QUE CAMBIA LOS PARAM. EN X O EN Y
      MOV         SBUF,#COMM
      LCALL       DELAY1
      MOV         R0,#66H
      ACALL       DATOEXY   ; RUTINIA QUE ENVIA LOS DATOS DEL CODIFICADOR EN X
      LJMP        MMAN
COMP10
      CJNE        A,#PY,COMP11
      LCALL       DELAY1
      ACALL       CPXY          ; RUTINA QUE CAMBIA LOS PARAM. EN X O EN Y
      MOV         SBUF,#COMM
      LCALL       DELAY1
      MOV         R0,#65H
      ACALL       DATOEXY       ; RUTINIA QUE ENVIA LOS DATOS DEL CODIFICADOR EN
X
      LJMP        MMAN
COMP11
      CJNE        A,#DAY,COMP12
      LCALL       DELAY1
      ACALL       PERDES        ; RUTINA QUE DESPLAZA AL EJE Y

PMMAN
      LJMP        MMAN
COMP12
      CJNE        A,#DBY,COMP13
      LCALL       DELAY1
      ACALL       PERDES        ; RUTINA QUE DESPLAZA AL EJE Y
      LJMP        MMAN


COMP13
      CJNE        A,#DDX,COMP14
      LCALL       DELAY1
      ACALL       PERDES        ; RUTINA QUE DESPLAZA AL EJE X
      LJMP        MMAN
COMP14
      CJNE        A,#DIX,COMP15
      LCALL       DELAY1
      ACALL       PERDES        ; RUTINA QUE DESPLAZA AL EJE X
PMMAN1
      LJMP        MMAN
COMP15    CJNE        A,#GP,COMP16
      LCALL       DELAY1
      ACALL       GRAP_DEZP     ; RUTINA QUE GRABA UN PUNTO
      LJMP        MMAN
COMP16
      CJNE        A,#DP,COMP17
      LCALL       DELAY1
      ACALL       GRAP_DEZP     ; RUTINA QUE RECIBE EL PUNTO
```

```
      LCALL     DELAY1
      ACALL     PERDES          ; RUTINA QUE DESPLAZA LOS EJES AL PUNTO
      LJMP      MMAN
COMP17
      CJNE      A,#OUT,COMP18
      LCALL     DELAY1
      ACALL     SALDAT          ; LLAMA A LA RUTINA QUE ENVIA UN DATO BINARIO
      MOV       SBUF,#COMM
      LCALL     DELAY1
      ACALL     SALIDA
PMMAN2:
      LJMP      PMMAN1
COMP18    CJNE      A,#VLX,COMP19
      LCALL     DELAY1
      ACALL     DVALA           ; RUTINA QUE ENVIA LA VEL,ACE,LIN,ANG
      LJMP      MMAN
COMP19    CJNE      A,#VLY,COMP20
      LCALL     DELAY1
      ACALL     DVALA           ; RUTINA QUE ENVIA LA VEL,ACE,LIN,ANG
      LJMP      MMAN
COMP20
      CJNE      A,#ALX,COMP21
      LCALL     DELAY1
      ACALL     DVALA           ; RUTINA QUE ENVIA LA VEL,ACE,LIN,ANG
      LJMP      MMAN
COMP21
      CJNE      A,#ALY,COMP22
      LCALL     DELAY1
      ACALL     DVALA           ; RUTINA QUE ENVIA LA VEL,ACE,LIN,ANG
      LJMP      PMMAN2
COMP22    CJNE  A,#SAL,PMMAN2
      RET
```

La rutina **PERDES**, simula una perforación o un desplazamiento en el eje X o en el eje Y, se diseño apartir de un ciclo en el cual el acumulador A se va incrementando y terminará cuando dicho valor sea igual a 0FFH, durante la ejecución de está rutina se trasmite el caracter que indica que se está ejecutando una tarea. La rutina **EAFTE**, se encarga de encender y apagar la fuente.

La función **CPXY** realiza la recepción de los incrementos tanto del eje X como del eje Y, en este caso los datos que son transmitidos por la PC son reflejados por el microcontrolador para la opción del codificador en X y en Y, empleando la rutina **DATOEXY**. La función **GRAP_DEZP**, se emplea para las opciones de grabar un punto y desplazarse al punto.

La función **DVALA**, se emplea para la recepción de los datos de velocidad y aceleración , lineal y angular. La función **SALDAT**, se emplea para la recepción del dato de salida. La función **SALIDA** es un dato externo que recibe el microcontrolador y que es proporcionado a la PC.

```
PERDES:
      LCALL          DELAY1
      MOV            A,#00H
LOOP1     LCALL           DELAY1
      MOV            SBUF,#ARR
      LCALL          DELAY1
      INC       A
      CJNE           A,#0FFH,LOOP1
      LCALL          DELAY1
      RET

EAFTE:
      LCALL DELAY1
      MOV  SBUF,#4FH
      LCALL DELAY1
      RET

GRAP_DEZP:
      LCALL DELAY1
      MOV  SBUF,#OK0
      LCALL DELAY1
      MOV  SBUF,#SIG
      LCALL DELAY1
      RET

SALDAT:
     LCALL DELAY1
     MOV  SBUF,#OK0
     LCALL DELAY1
     CLR  SCON.0
     MOV  PSW,#00H
     LCALL DELAY1
     MOV  R0,SBUF
     LCALL DELAY1
     MOV  SBUF,#SIG
     LCALL DELAY1
     RET

CPXY:
      LCALL DELAY1
      MOV  SBUF,#OK0   ; O
      LCALL DELAY1     ; RETARDO
      CLR  SCON.0
      LCALL DELAY1
      MOV  PSW,#00H
      MOV  R5,SBUF
      LCALL DELAY1
      MOV  SBUF,#SIG   ; S
      LCALL DELAY1     ; RETARDO
      CLR  SCON.0
      LCALL DELAY1
      MOV  PSW,#00H
      MOV  R6,SBUF
      LCALL DELAY1
      MOV  SBUF,#OK0   ; O
      LCALL DELAY1
      CLR  SCON.0
      LCALL DELAY1
      MOV  PSW,#00H
      MOV  R7,SBUF
      LCALL DELAY1
      RET

DVALA:
     LCALL DELAY1
     MOV  SBUF,#OK0
     LCALL DELAY1
     MOV  SBUF,#SIG
     LCALL DELAY1
     MOV  SBUF,#OK0
     LCALL DELAY1
     MOV  SBUF,#SIG
     LCALL DELAY1
     RET
```

```
DATOHFXY:
MOV   SBUF,#TRANS
      LCALL DELAY1
      CLR   SCON.0
      MOV   A,SBUF
CJNE  A,#53H,DATOHX
LOOP8:
      MOV   SBUF,R1
      CLR   SCON.0
      MOV   A,SBUF
      CJNE  A,#51H,LOOP8
      LCALL DELAY1
LOOP9:
      MOV   PSW,#08H
      MOV   SBUF,R1
      CLR   SCON.0
      MOV   A,SBUF
      CJNE  A,#4EH,LOOP9
      MOV   SBUF,#COMM
      RET

LCALL DELAY1
      MOV   SBUF,#TRANS
      CLR   SCON.0
      LCALL DELAY1
      MOV   A,SBUF
      LCALL DELAY1
      CJNE  A,#53H,DATOEY


SALIDA:
      MOV SBUF,#TRANS   ;>
      CLR   SCON.0
      MOV   A,SBUF
      LCALL DELAY1
      CJNE  A,#53H,SALIDA
LOOP6:
      MOV   SBUF,#67H
      CLR   SCON.0
      MOV   A,SBUF
      CJNE  A,#51H,LOOP6
      LCALL DELAY1
```

```
DATOEXY:
LOOP19:
      MOV   SBUF,R0
      LCALL DELAY1
      CLR   SCON.0
      MOV   A,SBUF
      LCALL DELAY1
      CJNE  A,#51H,LOOP19
      LCALL DELAY1
LOOP20:
      MOV   PSW,#00H
      MOV   SBUF,R6
      LCALL DELAY1
      CLR   SCON.0
      MOV   A,SBUF
      LCALL DELAY1
      CJNE  A,#52H,LOOP20
LOOP21:
      MOV   PSW,#00H
      MOV   SBUF,R5
      LCALL DELAY1
      CLR   SCON.0
      MOV   A,SBUF
      LCALL DELAY1
      CJNE  A,#4EH,LOOP21
      MOV   SBUF,#COMM
      RET


LOOP7:
      MOV   SBUF,R0
      LCALL DELAY1
      CLR   SCON.0
      MOV   A,SBUF
      LCALL DELAY1
      CJNE  A,#4EH,LOOP7
      MOV   SBUF,#COMM
      RET
```

Cabe resaltar que estas rutinas sólo son de prueba y que para diseñar el programa en ensamblador que pondrá en función todo el sistema, se deben tomar una serie de consideraciones que en este caso no se realizaron, como la velocidad y aceleración de HOME (INICIO) y END (FIN) para desplazar la plataforma y que son datos que deben estar integrados en el sistema al momento de su ejecución.

CAPÍTULO

V

PRUEBA Y RESULTADOS

¡Y mire como se agita, se pone en marcha, se desplaza y parece sentir una emoción vital. !

Henry Wadsworth Longfellow

V Plan de prueba del sistema.

La finalidad de este capítulo tiene por objeto probar el funcionamiento correcto del software diseñado para el control del equipo de desarrollo X-Y. Aunque no es posible realizar pruebas para la corrección de errores con todo el equipo en función, se empleo un sistema mínimo de control el cual está compuesto de :

1	Microcontrolador 8051	
1	Memoria RAM 6264	Para el almacenamiento de datos.
1	Memoria EPROM 2764	Para el almacenamiento de código de programas.
1	Circuito integrado 74LS02	Para la selección de localidades superiores a los de 8Kbytes
1	Circuito integrado 75176	Para la comunicación en la norma RS-485
1	Circuito integrado 75LS373	Para demultiplexar las direcciones del puerto 0

Además se empleo el programa MSD51, el cual almacena y ejecuta en el sistema mínimo las rutinas de prueba que se describieron en el capítulo anterior. Para no tener que estar grabando y borrando la memoria EPROM con las rutinas de prueba compiladas en el ensamblador 8051, el programa MSD51 carga dichas rutinas dentro de la memoria externa (RAM), denominada área de datos, y por medio del circuito que habilita a está memoria cuando se seleccionan localidades superiores a los 8Kbytes, se pueden ejecutar las rutinas de prueba creadas en ensamblador 8051 sin tener que accesar al área de memoria de programas, es por esto que todos los programas inician en la dirección 2100H. También se emplea una interfaz y una fuente de alimentación de 9volts.

Tomando en cuenta que es muy raro que un programa corra correctamente a la primera vez que se prueba, la depuración se realizo conforme se estaban codificando y compilando cada una de los módulos del programa. Esto se llevo acabo probando con una gran variedad de datos y los resultados que se arrojaron se compararon contra otros resultados con el objetivo de verificar que se han obtenido los resultados correctos. Como no se pudieron probar todas las condiciones que requieren el empleo de la comunicación entre la PC y el microcontrolador, no fue posible encontrar todos los errores existentes. Aunque dichas rutinas se probaron y se corrigieron durante su diseño, simulando la operación de comunicación empleando una terminal tonta. Es por esto que cuando se cuente con el sistema completo se tomaran las decisiones pertinentes para corregir los errores que se presenten.

V.1 Modalidad Perforado de Tarjetas

Para probar las opciones contenidas en está modalidad se emplearon los archivos: SI-1A.TXT, DEMO.TXT y FUENTE.TXT, los cuales se crearon en los paquetes Smartwork, Tango y Orcad, respectivamente. Para efecto de prueba se empleo el archivo SI-1A.TXT, como se muestra en la Figura 5-1.

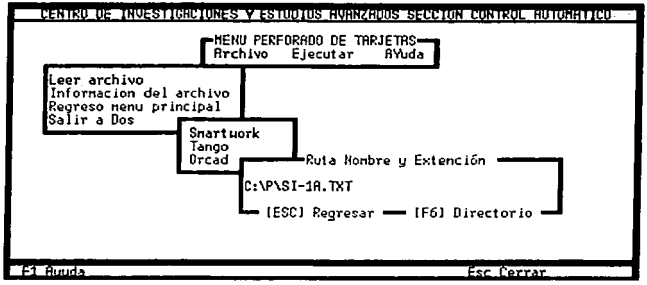


Figura 5-1. Ventana que carga un archivo de texto.

Después de ser cargado el archivo, se accesa a la opción **Información del archivo** que se encuentra dentro del menú **Archivo**, esto con el objeto de visualizar que en realidad los datos se filtraron y almacenaron correctamente, esto en base al número de perforaciones desplegado en la pantalla y al circuito que se muestra.

Si se desea que se pueden modificar las condiciones en que ha de operar el sistema durante el proceso de perforación con respecto a la velocidad y aceleración de los ejes.

También es posible modificar la resolución del motor, cabe resaltar que el sistema no contempla el cambio de resolución para cada eje, por lo cual los motores de la mesa deben tener la misma resolución. Como lo muestra la Figura 5-2.

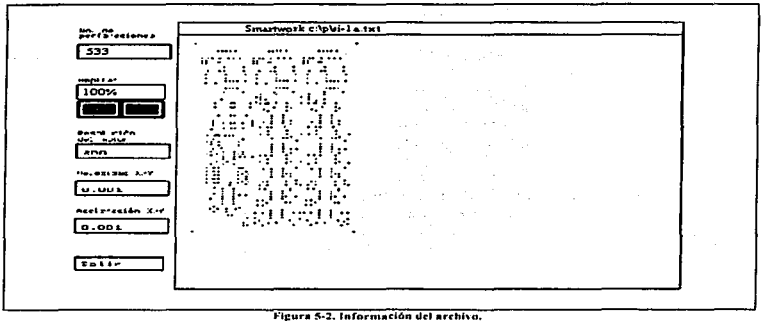


Figura 5-2. Información del archivo.

El objetivo principal es probar la comunicación que se establece al momento de la ejecución de está modalidad, ya que es aquí donde el microcontrolador interactua con la PC. Al momento de elegir la opción **Ejecutar**, se realiza la transmisión de los datos con los que debe funcionar durante la operación del perforado, al termino se accesa a la pantalla que se muestra en la Figura 5-3.

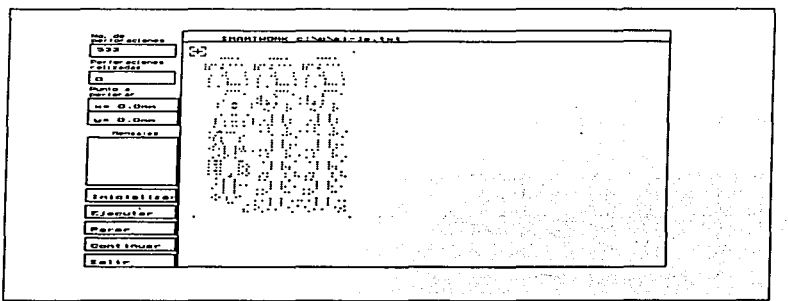


Figura 5-3. Opción de ejecución.

Al elegir la opción **Ejecutar** se inicia el proceso de perforación y en el área de mensajes, se despliega lo siguiente: **Transmitiendo la información, Desplazando la mesa y Perforando.**

El primer mensaje, se refiere al envío de las coordenadas del punto a perforar. El segundo mensaje se refiere, al momento en que se está desplazando la mesa al punto indicado y por último el tercer mensaje indicarán que los datos se están transmitiendo correctamente.

Recordamos que el desplazamiento y la perforación se simulan en base a una rutina de retardo que se encuentra en el microcontrolador, y es por esto que no importa la distancia que tenga que recorrer de un punto a otro el tiempo siempre es el mismo. Aquí también se pueden probar las opciones de **Parar, Continuar e Inicializar**, así como también la opción de **Salir**.

V.2 Modalidad Maquinado de Piezas.

En está modalidad se cuenta con un Editor de texto y un Compilador en los que se escriben programas para después ejecutar los en forma Automática o Interactiva.

Tomando en consideración que la rutina que se encuentra en el microcontrolador es muy limitada, se pondrán crear y compilar programas con todas la sentencias disponibles, pero para probar la ejecución Automática o Interactiva sólo se emplearan las sentencias: **ANG()**, **ALN()**, **VAN()**, **VLN()**, **BEG()**, **END()**, **OUT()**, **INP()**, **EFT()**, **AFT()** y **ATL()**.

Los programas escritos se denominan programas fuentes y al momento de ser almacenados se les asignara la extensión [.DAT], esto con el objetivo de poder ser modificado para otro tipo de aplicaciones .

Cuando la codificación ha concluido y después de compilar, ya no existen errores, las sentencias se codifican en hexadecimal y dichos datos son almacenados en un archivo con el mismo nombre pero con la extensión [.HEX], al cual se le denominara programa objeto. Como se muestra en la Figura 5-4.

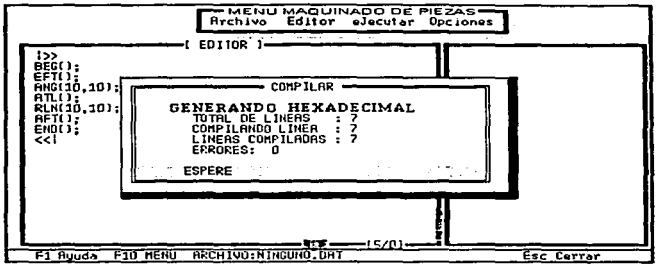


Figura 5-4. Ventana que genera el archivo en hexadecimal.

A continuación se presentan una serie de programas que ejemplifican el empleo de las sentencias, así como del uso de la estructura de control y las subrutinas, estos programas pueden ser compilados sin ningún problema.

Para contar con una mayor información del desarrollo de programas , así como del uso de todas las opciones disponibles en está modalidad, puede recurrir al capítulo III.

Programa perfora una placa de Nylamit de 15x15, las perforaciones son de 1/8 de pulgada. En este programa sólo se emplean sentencias.

```
BEG();
RMD(2000);
GHM();
ALN(1.6,0.7);
VAN(1.6,11);
EFT(); RLN(100,100);
GLN(); ATL();
RLN(50,0);
GLN();
ATL();
RLN(100,50);
GLN();
ATL(); RLN(-50,0);
GLN();
ATL();
RLN(25,-25);
GLN(); ATL();
GHM();
AFT();
END();
```

Programa perfora una placa de Nylamit de 15x15, las perforaciones son de 1/8 de pulgada. En este programa se muestran el empleo de sentencias junto con la estructura de control anidada.

```
BEG();
RMD(2000);
GHM()
ALN(12,10);
VAN(10,10.8);
EFT();
JLP(1,50);
JLP(2.10);
 RLN(10,10);
 GLN();
 ATL();
FLP(2);
JLP(3,10);
 RLN(10,10);
 GLN();
 ATL();
FLP(3);
FLP(1);
GHM();
AFT(1);
END();
```

Programa que perfora una placa de Nylamit de 15x15, las perforaciones son de 1/8 de pulgada. En este programa se muestra el empleo de sentencias junto con la subrutina y la estructura de control.

```
SUB(1);
JLP(1.50);
 RLN(10,0);
 GLN();
 ATL();
FLP(1);
RTE();

BEG();
RMD(2000);
GHM()
ALN(1.8,1.0);
VAN(11.5,10.7);
EFT();
RLN(50,50);
GLN();
GSB(1);
RLN(50,100);
GLN();
GSB(1);
RLN(50,100);
GLN();GSB(1);
GHM();
AFT();
END()
```

V.2.1 Ejecución en forma Automática e Interactiva.

Cuando se haya escrito el programa y después de compilar lo se encuentre libre de errores y se haya generado el archivo objeto, el siguiente paso consiste en ejecutar el programa. La elección entre ejecutar un programa en forma Automática o Interactiva, reside únicamente en las necesidades del usuario, ya que la diferencia entre ambas, consiste en la secuencia de ejecución de las instrucciones y el tiempo que puede durar la sesión de ejecución de cualquier de estas dos opciones.

En ocasiones la opción de ejecución en forma Interactiva, se puede emplear para probar el funcionamiento de algunas sentencias del programa, antes de elegir la opción Automática.

Como únicamente se cuentan con tansolo unas cuantas rutinas en el microcontrolador que reconocen unas cuantas sentencias, para probar cualquiera de estas dos formas de ejecución, se introduce el siguiente programa:

Programa que emplea una subrutina en donde se activa el taladro ocho veces. También modifica la aceleración y velocidad tanto angular como lineal.

```
SUB(5);
JLP(7,8);
ATL();
FLP(7);
RTE();

BEG();
EFT();
ANG(12.8,13.4);
VAN(4.7,5.8);
GSB(5);
ALN(1.2,0.9);
VLN(5.3,5.8);
GSB();
AFT();
END();
```

Después de acabar la edición, se ingresa el menú principal y se elige la opción **Compilar** que se encuentra en el menú principal en la opción **Editor** para revisar la sintaxis y generar el archivo objeto. Por último se elige la forma en que se ha de ejecutar el programa, ingresando al menú **Ejecutar**. Los resultados se observaran en la pantalla de mensajes, como se muestra en la Figura 5-5.

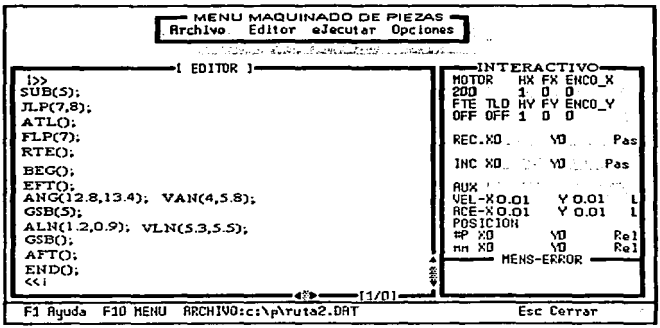


Figura 5-5. Pantalla de ejecución en forma interactiva.

Cabe recordar que para iniciar la ejecución en forma Interactiva no es necesario ejecutar la sentencia **BEG()**, pero para dar por terminada dicha sesión se tiene que ejecutar la sentencia **END()**. Además para ejecutar una sentencia se debe seleccionar primero mediante el desplazamiento del cursor a la sentencia y presionar Enter. No importa el número de sentencias que existan en una linea, todas se ejecutan conforme al orden en que se encuentren.

V.2.2 Ejecución en forma Manual.

Las opciones disponibles en la ejecución en forma Manual, se identifican mediante la letra mayúscula resaltada con rojo y para elegir alguna de estas opciones sólo basta con presionar conjuntamente la tecla **ALT** y la letra de la opción resultada. También esto se puede realizar por medio del **Mouse**.

Para probar las opciones que modifican los incrementos en paso para los ejes X-Y, sólo basta con elegir el eje que se va ha desplazar y escribir la nueva cantidad de pasos, si los datos son recibidos correctamente por el microcontrolador, los reflejara en el área correspondiente a los codificadores ópticos.

El mismo procedimiento se emplea para probar las opciones de salida y entrada en un dato binario, ya que el dato que se escriba en el área de salida lo retornara el microcontrolador en el área de entrada. Para probar las opciones de desplazamiento de la plataforma de la mesa sólo se presiona cualquiera de las flechas de desplazamiento en ese instante transcurrirá un tiempo antes de los ejes en la pantalla se desplacen y se actualicen los datos. Durante ese tiempo se activaran y aparecerán en el área de mensajes las opciones de **Parar**, **Continuar** y **Cancelar**, estas opciones no se pueden simular puesto que el microcontrolador no cuenta con esas rutinas.

De igual forma la opción **Desp. al Pto** presenta está característica, si se elige alguna de estas opciones puede ocasionar que se rompa la secuencia de ejecución del programa en el microcontrolador, por lo consiguiente se romperá la comunicación con la PC. Las opciones de **Perforar**, **Reg. al pto**, **Cam. al pto** y **Graba-pto** se encuentra disponibles para probar su funcionamiento. Todas estas opciones se pueden visualizar en la Figura 5-6.

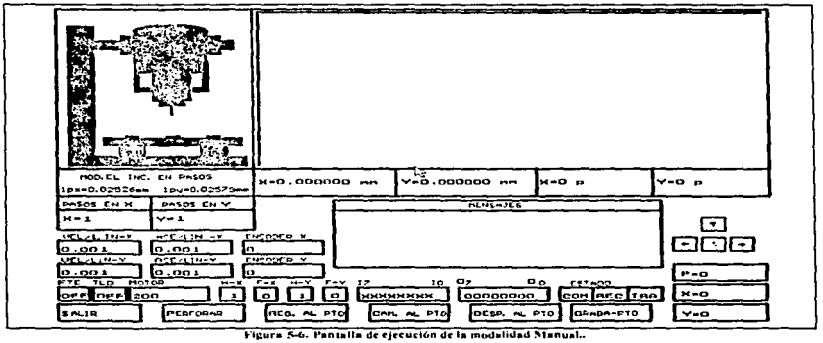


Figura 5-6. Pantalla de ejecución de la modalidad Manual..

Los resultados son satisfactorios, siempre y cuando se someta el sistema a una serie de pruebas funcionales, es decir, que la secuencia de selección de las opciones disponibles sean congruentes, debido a que la rutina contenida en el microcontrolador es limitada y ciertas operaciones no las soporta.

Conclusiones

Este tipo de software se desarrolla con el fin de satisfacer las necesidades particulares de los usuarios cuando se realizan aplicaciones determinadas. Aunque para algunos este tipo de software resulta ser más caro, debido a que se realiza una gran inversión de tiempo de programación y prefiere invertir en la compra de un paquete de software que realiza la misma operación. Pero este resulta ser en muchos casos contraproducente, ya que este tipo de software requiere ser ejecutado junto con todas sus herramientas para el cual fue diseñado. Y además este tipo de software nunca crece conforme a las necesidades del o los usuarios. Los beneficios se pueden observar realizando la comparación del siguiente equipo de perforado que se encuentra en el mercado contra el desarrollo del proyecto en el CINVESTAV.

COMPRA DEL EQUIPO

Marca: Kepro.
Modelo: CNC128RCC6
Tamaño: 94.44*cm* de largo
96.52*cm* de ancho
45.72*cm* alto
Peso: 83.9*kg*
2 Motores a pasos: 1 paso=0.00254*cm*
1 Taladro neumático: 28000rpm

Costo aproximado: $35,925

Además requiere:

Software: Symphony y Side-Kich

DESARROLLO DEL PROYECTO

2 motores pasos	$ 800.00
1 Taladro Neumático	$ 2,500.00
Mesa X-Y	$ 3,000.00
Tiempo de programación	$ 8,000.00
Sistema electrónico	$ 2,000.00

Costo aproximado: $16,300.00

En comparación nuestro proyecto es más factible con respecto al costo, dado que el equipo KEPRO, le faltaría al costo de importación debido a que este tipo de equipo no existen en nuestro mercado.

Otro tipo de beneficios, que se pueden originar con el desarrollo de este tipo de software, es que además de cumplir con el objetivo de satisfacer las necesidades de los usuarios del laboratorio de Robótica del CINVESTAV, con los elementos que originalmente se cuentan para el proceso de perforación, es posible contar con el código fuente y la documentación necesaria para futuras modificaciones debido a que este tipo de sistemas no es algo que quede invariable para siempre ya que después de un cierto tiempo se puede crear la necesidad de alterar el sistema para satisfacer nuevas condiciones con respecto al entorno operativo o alguna otra necesidad.

Glosario

Análisis: Etapa de desarrollo de sistemas en la cual los analistas y varios usuarios establecen las especificaciones del sistema que desea implantar.

Apuntador: Véase: puntero.

Archivo de texto: Un archivo contiene solamente letras, números y símbolos. este archivo no puede contener instrucciones de formato con operación de avances de línea y retornos de carro. Un archivo de texto es un archivo ASCII.

Archivo: Conjunto de información identificado con un nombre y almacenado en un disco. está información puede ser un documento o una aplicación.

Argumento de una función: Es el valor que pasa a la función en el momento que se llama.

Array: Es una colección de variables del mismo tipo que se referencian utilizando un nombre común. Un Array consta de posiciones de memoria contiguas. La dirección más baja corresponde al primer elemento y la más alta al último. Un Array puede tener una o varias dimensiones. Para acceder a un elemento especifico de un Array se usa un índice.

ASCII: Abreviatura de American Standard Code for Information Interchange.

Bifurcación: Instrucción de un programa que permite saltar cierto número de instrucciones.

Binario: Sistema provisto de sólo dos valores 0 y 1.

Bit: Abreviación estándar que se desprende de **BI**nary digi**T**

Bucle: es un conjunto de instrucciones que se ejecutan hasta que se cumple una condición.

Byte u octeto: Un conjunto de bits que en su totalidad suman 8 y que representan unidades básicas de información.

Cadena: Una secuencia de símbolos de datos que han de tratarse juntos como un dato lineal.

Caracter: Cifra, letra o signo representado por un conjunto de bits. Celda Un lugar en la memoria principal donde la información puede ser almacenada; posición de memoria identificada por una dirección.

Codificación de datos: Es la conversión de la información de una forma inteligible para los seres humanos a otra forma inteligible para la máquina

Codificación: es la actividad de la programación implicada en escribir las instrucciones en el lenguaje de programación escogido.

Código ASCII: Un estándar creado para representar información internamente en la computadora y que después puede ser visualizado por la computadora

Comando: Una palabra u oración que generalmente realiza una acción.
Comparador: Un dispositivo para hacer una comparación y que, frecuentemente en computadoras , actúa sobre los resultados de la comparación.
Comparar: Examina dos valores con el propósito de descubrir identidad o magnitud relativa que las diferencia.
Compilador: Programa que sirve para traducir un lenguaje de programa simbólico más alto a un lenguaje máquina que sea comprensible para la máquina.
Computadora: Dispositivo capaz de aceptar información, procesarla y entregar los resultados de este proceso en forma operante.
Constante: Un valor de un dato que no cambia con cada ejecución de un procedimiento.
Contador: Dispositivo que permite alteraciones sobre números por una cantidad arbitraria. Ordinariamente se usa para determinar cuando un proceso repetitivo ha terminado.
Datos: Hechos que describen entidades.
Dirección: Identificación, en forma numérica, de la ubicación de una información en la memoria o en otro dispositivo de almacenamiento.
Editor: El programa que utilizan los programadores para componer y modificar otros programas; programa de procesamiento de texto utilizando para la escritura de las instrucciones de los programas.
Estructura de control: La estructura de un programa definida por referencias con las cueles se representan las transferencias de control. Construcciones mediante las cuales se escriben los programas.
Estructura: La representación concreta del estado de una entidad.
Función: Es una colección independiente de declaraciones y sentencias, generalmente enfocadas a realizar una tarea especifica.
Hardware: Los circuitos electrónicos y dispositivos electromecánicos que constituyen el sistema de computación.
Hexadecimal: Número de un sistema de base 16 en donde un dígito *n* asume valores de **0** a **9** y además otros seis valores que por lo general son representados por las letras del abecedario de la **A** a la **F**.
Información: Colección de datos significativos y pertinentes que describen sucesos y entidades.
Instrucción: Conjunto de caracteres que representan una orden dada a la máquina y está puede, naturalmente, cumplir.
Interrupción: Señal utilizada por el sistema operativo MS-DOS para comunicarse con el procesador de la computadora.
Lenguaje de programación: Aquel que utilizan los programadores para escribir un programa en forma más o menos cómoda y que por lo general requiere de una traducción para ser transformado a lenguaje máquina.

Lenguaje máquina: es el lenguaje binario de la computadora.

Lenguaje: Conjunto de caracteres, símbolos, palabras, frases, instrucciones y reglas que permiten escribir y describir programas para una aplicación dada.

Menú: Área rectangular de la pantalla en la cual aparece una aplicación o documento.

Mnemónico (Nemónico): Termino utilizado para definir la acción que ha de tomar una instrucción en el lenguaje ensamblador.

Modelo matemático: Representación matemática de un proceso, dispositivo o concepto que permite la manipulación matemática de variables a fin de estudiar el comportamiento del proceso, dispositivo o concepto en diferentes condiciones.

Módulo: Es una parte individual de un programa para computadoras que está limitado en su alcance y que proporciona alguna función requerida para el objetivo del programa entero.

Parámetro: Lista de variables que envía durante una llamada un programa a una rutina.

PC: Computadora Personal.

Píxel: La unidad gráfica más pequeña que se puede representar en la pantalla. Píxel es la abreviatura en inglés de "picture element" (elemento de imagen).

Programa: Un determinado conjunto de instrucciones escrito para realizar una tarea determinada en un sistema.

Programación estructurada: Un conjunto de principios y técnicas para la escritura de programas como un conjunto anidado de entradas y salidas únicas de bloques de código: usando un restringido número de estructuras de control.

Protocolo: Juego de reglas y convenciones que regulan el intercambio de datos en una red. Dado que existen diferentes tipos de protocolos y cada una utiliza un juego de reglas y convenciones diferentes. Los miembros del grupo de trabajo que deseen intercambiar información deberán utilizar el mismo protocolo en sus computadoras.

Puerto: Conexión o enchufe utilizado para conectar un dispositivo como una impresora, un modem, ala computadora. Los dos puertos más comunes son los puertos serie (por ejemplo, COM1, COM2) utilizados por los dispositivos que aceptan información bit por bit y los puertos paralelos (por ejemplo, LPT1 y LPT2), utilizados por los dispositivos que aceptan ocho bits por vez y generalmente son más rápidos que los puestos serie.

Puntero: Es una variable que contiene una dirección de memoria.

Recursividad: Capacidad de una función de llamarse así misma.

Ruta de acceso: Especifica la ubicación de un archivo dentro de un árbol de directorios o la posición de un recurso compartido en una configuración de grupo de trabajo

Rutina: Segmento de un programa que se caracteriza por un principio y un fin en la codificación total del proceso.

Simulación: Representación de un fenómeno o una acción.

Sistema: Cualquier conjunto de componentes cuya función conjunta es alcanzar un objetivo.

Software: Las instrucciones que se utilicen para dirigir los componentes de hardware de la computadora para la realización de tareas determinadas.

Subrutina: Es el módulo de las instrucciones del programa, que puede ser ejecutado mediante referencia a una sentencia del programa; se utiliza para reducir los conjuntos redundantes de instrucciones del programa.

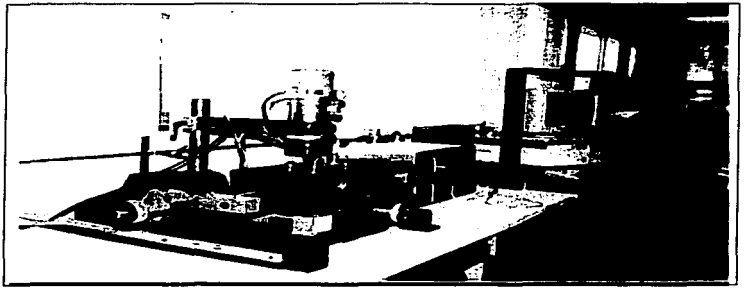
Terminal: Es cualquier dispositivo de entrada y salida. El término está asociado más corrientemente con aquellos ingenios a través de los cuales se comunican los usuarios con el sistema: El teclado, la pantalla y o impresora.

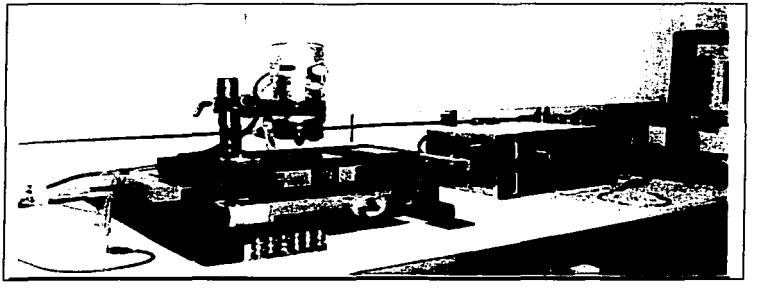
Variables: Nombres de elementos de datos para los cuales pueden variar los valores asociados del elemento de un dato.

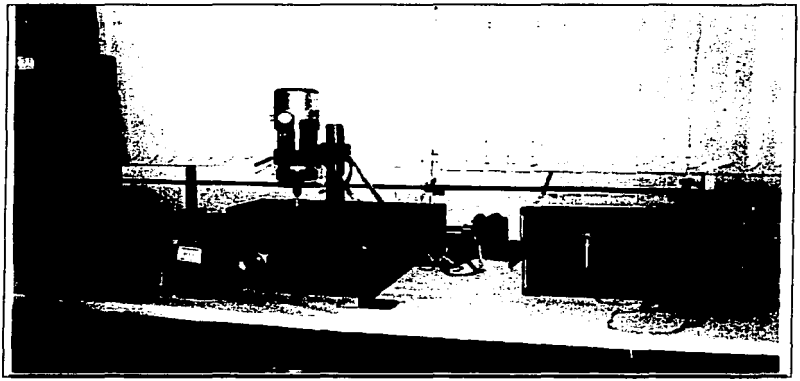
Ventana: Área rectangular de la pantalla en la cual aparece una aplicación o documento.

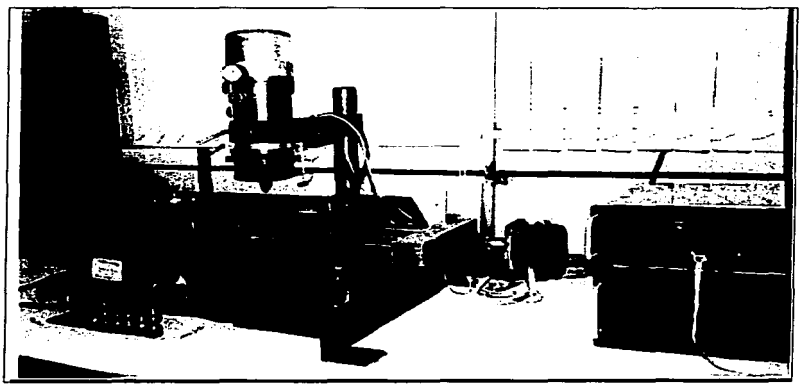
APÉNDICE A

Equipo X-Y



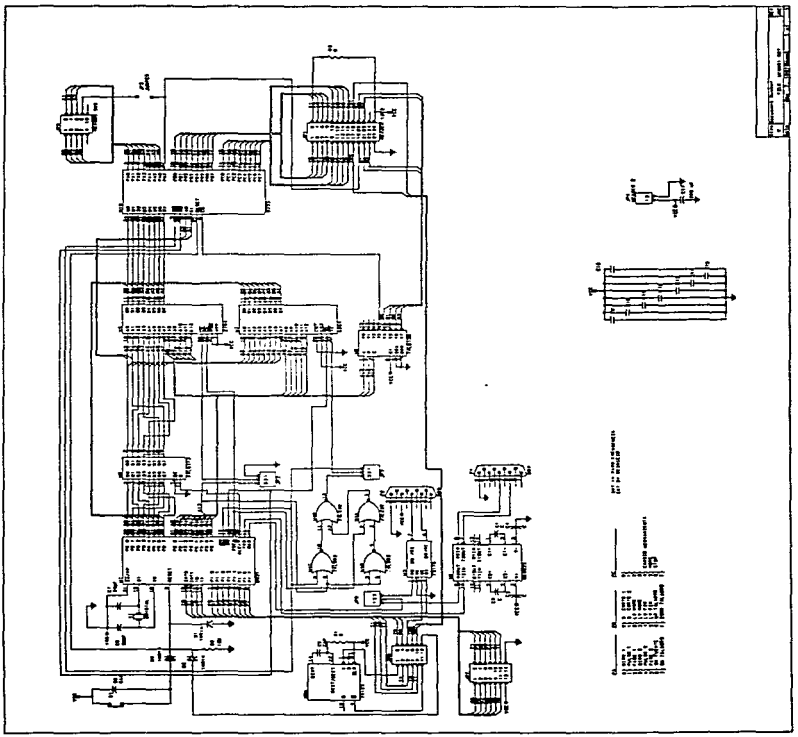






APÉNDICE B

Diagrama del sistema de desarrollo



Referencias bibliográficas

1) **Acell Technologis Ing.** *Tango PCB.*

2) **Intel** 1989. *8 Bit Embebbed Controller Handbook.*

3) **Ceballos Sierra, Fco. Javier.** 1991.*Enciclopedia del lenguaje C.* Addison Wesley

4) **Cristtie Linda y John** 1986. *Enciclopedia de términos de microcomputación* Prentice Hall.

5) **García Guillen, Pedro** 1991. *Orcad SDT/III.* Paraninfo Madrid.

6) **Guzman Mendoza, Carlos** 1993. *Manual de Usuario del Paquete Smartwork.* Textos del centro de computo ENEP Aragón.

7) **Kernighan B y Ritchi Denis** 1985. *Lenguaje de programación C.* Mc Graw Hill.

8) **Koffman, Elliot B** 1985. *PASCAL, Introducción al lenguaje y resolución de problemas con programación estructurada* Fondo Educativo Interamericano

9) **Mora, José Luis** 1988. *Introducción a la informática.* Trillas.

10) **Peñaloza Romero, Ernesto** 1996. *Fundamentos de programación.* UNAM.

11) **Schild, Helrbet** 1990. *Lenguaje C Programación Avanzada.* Mc Graw Hill.

12) **Schild, Helrbet** 1990. *Programación en Turbo C.* Mc Graw Hill.

13) **Schild, Helrbet** 1994. *Manual de referencia Turbo C/C++ 3.1.* Mc Graw Hill.

14) **Schwartz, Arleen G** 1987. *E libro del Basic.* Interamericana

15) **Stallings, Willian** 1995. *Sistemas operativos.* Limusa.

16) **Tenenbaum, Aaron M** 1985. *Estructura de datos en Pascal.* Prentice Hall.

17) **Tenenbaum, Andrew S** 1988. *Sistemas operativos Diseño e implementación.* Prentice Hall.

18) **Tinoco Alvarado, Alejandro** 1989. *Sistema Distribuido con microcontroladores de línea 8051.* Depto. de Ingeniería Eléctrica. CINVESTAV.

19) **Vega Salinas, Alejandro** 1992. *Manual y Aplicaciones del microcontrolador 8051.* Depto. de Ingeniería Eléctrica. CINVESTAV.

20) **Versellezo, Robert** 1986. *Procesamiento de datos.* Mc Graw Hill.