ESCUELA SUPERIOR POLITECNICA DEL LITORAL
FACULTAD DE INGENIERIA ELECTRICA Y COMPUTACION

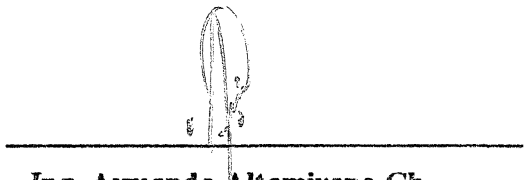
DISEÑO Y CONSTRUCCION DE UN SISTEMA DE DESARROLLO (SDI - 86) PARA EL MICROPROCESADOR INTEL 8086 PARA SER USADO EN EL LABORATORIO DE MICROPROCESADORES DE LA FACULTAD DE INGENIERIA ELECTRICA Y COMPUTACION DE LA ESPOL

TESIS DE GRADO
Previa a la obtención del Titulo de:
INGENIERO EN ELECTRICIDAD
ESPECIALIZACION: ELECTRONICA

Presentada por:

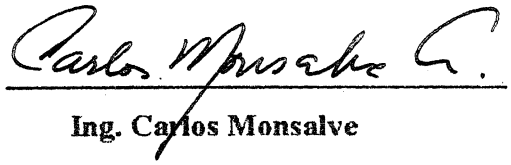
ALEX ARELLANO SILVA
ERICK ARELLANO SILVA
HOLGER NARANJO PEREZ

GUAYAQUIL - ECUADOR
1998

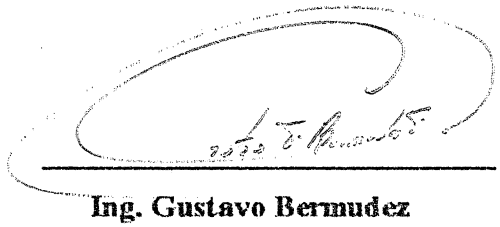


Ing. Armando Altamirano Ch.
SUBDECANO DE LA F.I.E.C.

Ing. Juan Del Pozo
DIRECTOR DE TESIS



Ing. Carlos Monsalve
MIEMBRO PRINCIPAL
DEL TRIBUNAL

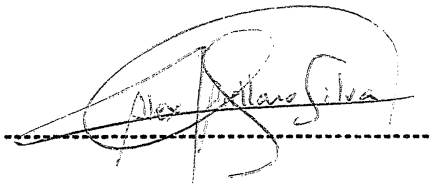


Ing. Gustavo Bermudez
MIEMBRO PRINCIPAL
DEL TRIBUNAL

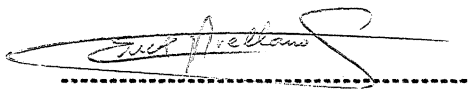
DECLARACION EXPRESA

"La responsabilidad por los hechos, ideas y doctrinas expuestos en esta tesis, nos corresponden exclusivamente; y, el patrimonio intelectual de la misma, a la ESCUELA SUPERIOR POLITECNICA DEL LITORAL".

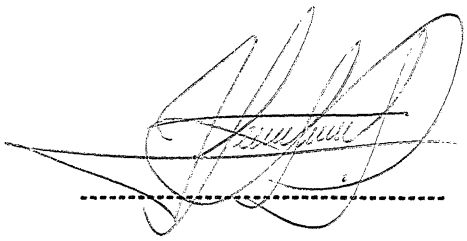
(Reglamento de Exámenes y Títulos profesionales de la ESPOL).



Alex Arellano Silva



Erick Arellano Silva



Holger Naranjo Perez

Agradecemos a nuestra querida y abnegada Madre Noemí,
Ya que sin su ayuda no hubiéramos podido tener
Los conocimientos para desarrollar esta Tesis.
Ella se ha esforzado mucho para darnos la educación
Y ha luchado para que no nos falte nada.

Erick y Alex.

Agradezco a mis Padres que con entero sacrificio
Y abnegación, supieron entregar todo de si,
Para hacer de mí un ser útil a la Patria y a la Sociedad.

Holger.

Agradecemos a Magda y César por habernos ayudado
Cuando más lo necesitabamos y sin pedir nada a cambio.

Erick, Alex y Holger.

A nuestra Madre y Hermanos
Por su constante cariño y comprensión.

Erick y Alex.

A mis queridos Padres y apreciados Hermanos.

Holger

RESUMEN

El SDI - 86 es un *Sistema de Desarrollo Integrado* para aplicaciones electrónicas con microprocesadores *8086* de Intel, consistente en tres módulos fundamentales: SDI - 86P, SDI - 86S y SDI - 86G.

El SDI- 86P es un software bajo Windows™ para editar un programa en lenguaje ensamblador del 8086, depurarlo, compilarlo y enlazarlo; con el fin de ejecutarlo en el equipo denominado SDI - 86S que permite la ejecución de las aplicaciones o simplemente usarlo como un entorno de programación. Para esto el equipo SDI - 86S tiene su propio Sistema Operativo.

Entre las principales ventajas del software SDI - 86P está la visualización de la memoria, introducir programas en ella y rastrear su ejecución, (esto es, reemplaza al DEBUG del DOS™). Además realiza una simulación por software del SDI - 86S.

El SDI - 86G es un equipo diseñado para grabar memorias tipo EPROM 2764, dos de las cuales se utilizan en el SDI - 86S, y es necesario conectarlo a un Computador Personal, a través de su puerto paralelo, para ser controlado por el programa SDI - 86P.

INDICE GENERAL

INDICE DE FIGURAS

INDICE DE TABLAS

INDICE DE ANEXOS

INTRODUCCION

Aprender a programar en ensamblador no es tarea fácil al inicio, hay que comprender la arquitectura del chip que se va a programar, para nuestro caso un microprocesador INTEL 8086. El set de instrucciones que éste chip nos ofrece y los cambios que ocurren en su estructura interna al ejecutarse cada una de ellas. El objetivo principal de nuestra tesis no es sólo proveer de un sistema que les permita desarrollar aplicaciones electrónicas para éste procesador, sino también que ayude al usuario aprender de una forma visual y amigable la programación en ensamblador, en este caso del 8086. Cabe destacar, que aprendiendo bien la programación de un chip en ensamblador se podrá fácilmente aprender a programar en ensambladores de otros chips.

En la asignatura SISTEMAS DE MICROPROCESADORES en la ESPOL se estudia la familia de microprocesadores Intel™, siendo el 8086 la base de todos ellos. La ESPOL cuenta con un Laboratorio para ésta materia en la que se encuentran algunas máquinas basadas en el 8086, las mismas que utilizan una tarjeta conectada a los buses del sistema, para obtener sus señales (dirección, datos, control, etc.) en un protoboard para que los estudiantes puedan realizar aplicaciones electrónicas con el 8086. Pero éste sistema presenta algunos inconvenientes tales como el deterioro de los equipos, dificultades de uso, insuficiente disponibilidad del laboratorio, entre otras. Por tales motivos surgió la idea de proporcionar al estudiante un sistema que permita realizar aplicaciones de una manera más sencilla y didáctica, permitiéndole incluso poder trabajar en su propia casa y con un computador mucho más avanzado, con herramientas de simulación para que el usuario no necesite realizar ningún tipo de cableado a menos que desee implementarlo físicamente.

Se planteó crear un Sistema de Desarrollo que inicialmente sería para el 8086 y que luego podría ser extendido a los demás microprocesadores de esta familia, el cual permitiría reemplazar las funciones del DEBUG, realizar la edición de un programa en ensamblador de una manera más sencilla y poder crear una aplicación basada en el 8086, la cual se ejecutaría independientemente del PC en el que inicialmente se trabaja, creándose para ello un sistema de aplicaciones que no es otra cosa que una

microcomputadora con sus propias características de hardware y sistema operativo, que permita controlar los dispositivos utilizados en su construcción. Se escogió al 8086 porque realizar una microcomputadora para éste es más sencillo que si se utilizará un chip más avanzado de ésta familia, además para realizar sistemas electrónicos basados en microprocesadores no se requiere que éstos sean muy avanzados, por cuanto para ello se requiere armar una microcomputadora y mientras más avanzado es el microprocesador, más complejo es el diseño y más costoso resulta. Hay que tomar en cuenta que si los microprocesadores han tenido la necesidad de mejorar es debido a que las PCs han sido diseñadas para realizar diversas actividades, requiriendo el manejo de dispositivos cada vez más sofisticados como scanners, cámaras digitales, monitores de mayor resolución y con más colores, etc. Mientras que para realizar una aplicación específica en electrónica no se demanda de tanto poder en un microprocesador y más bien esto depende de la aplicación. Una prueba de ésto es que se han creado *microcontroladores* para realizar el diseño de sistemas electrónicos específicos, siendo un microcontrolador una microcomputadora mucho menos avanzada que la que hemos construído en nuestra tesis, pero en un solo chip y que permite realizar sistemas electrónicos de control muy interesantes, compactos y económicos.

CAPITULO I

INTRODUCCION AL SDI - 86

1.1. COMPONENTES DEL SDI - 86

El **SDI - 86** es un *Sistema de Desarrollo Integrado,* que permite crear aplicaciones electrónicas basadas en el microprocesador 8086. Está constituído por las siguientes partes:

- **SDI - 86P** (Programa creado en Visual Basic 3.0 de Microsoft)
- **SDI - 86G** (Grabador de EPROMs 2764)
- **SDI - 86S** (Sistema de Aplicaciones)

1.1.1. SDI - 86P

El **SDI - 86P** es un software que contiene un programa principal creado en Visual Basic 3.0 (versión profesional) de Microsoft (Ver figura 1). Este permite al usuario editar un programa en el ensamblador del 8086, depurarlo, compilarlo y enlazarlo. Estas dos últimas acciones realizadas con ayuda del Turbo Ensamblador de Borland.

Para la depuración de un programa escrito en ensamblador, el **SDI - 86P** utiliza 2 opciones: Ejecutar paso a paso o con objetivo. La primera permite ver el cambio en registros y banderas que ocurren cada vez que se ejecuta una instrucción, mientras que la segunda permite colocar objetivos donde se

desean ver dichos cambios. Esta última opción fue incluida, porque a veces depurar un programa paso a paso puede resultar muy molestoso y sólo se necesitan analizar cambios antes y después de un determinado bloque de instrucciones. La depuración paso a paso puede servir para ayudar a comprender el funcionamiento del microprocesador cuando ejecuta determinadas instrucciones. Por esto, es recomendable que alguien que recién comienza a estudiar el 8086, realice programas pequeños y los ejecute paso a paso, para que pueda entender el funcionamiento de las instrucciones de una manera más fácil que recurriendo a la lectura de un texto académico. Después de todo, el ser humano aprende más fácilmente de una manera visual que leyendo.

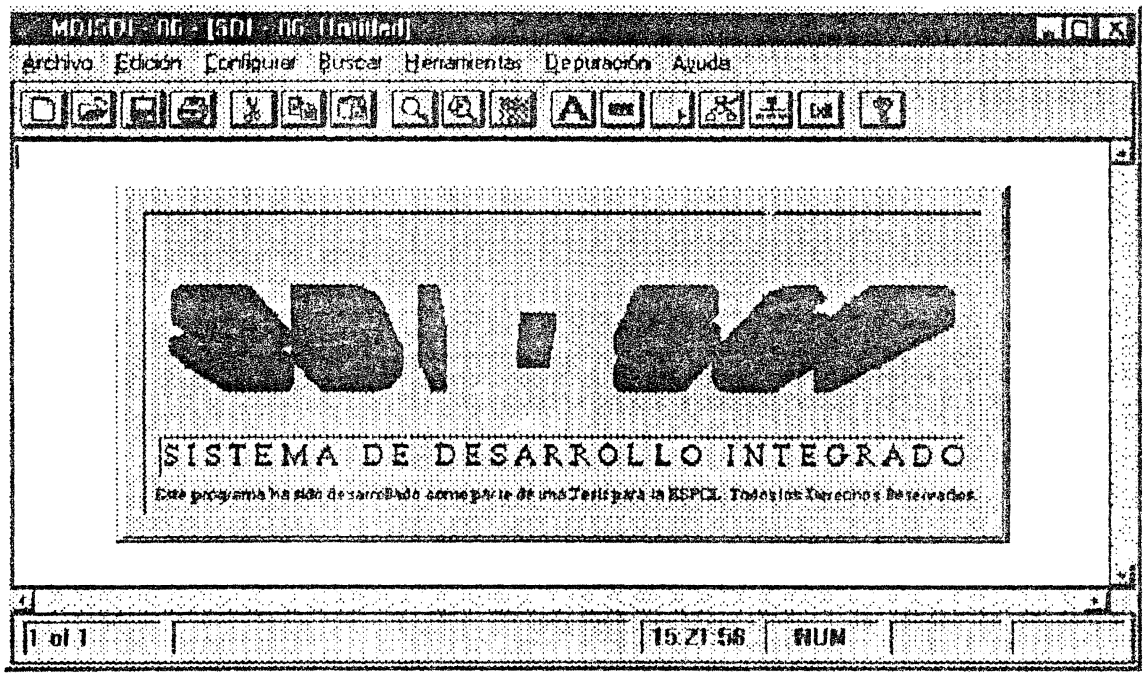


Figura 1: Interface principal del SDI-86P

1.1.2. SDI - 86G

El SDI - 86G (Ver Figuras 2 y 3), es un hardware que se conecta a la computadora mediante el puerto paralelo y que permite grabar EPROMs 2764 con el programa que contiene la aplicación del usuario, además del sistema operativo y la tabla de datos del display HD-44780 de Sharp del SDI - 86S. El programa escrito por el usuario del SDI - 86, contiene un segmento de código que se grabará a partir

de la localidad 1000h en dos memorias EPROMs, una para las pares (banco bajo) y una para las impares (banco alto), siendo el inicio real del segmento de código en la localidad 2000h (1000h x 2); además de un segmento de datos, que se grabarán a partir de la localidad 800h, siendo el inicio real la localidad 1000h para el segmento de datos.

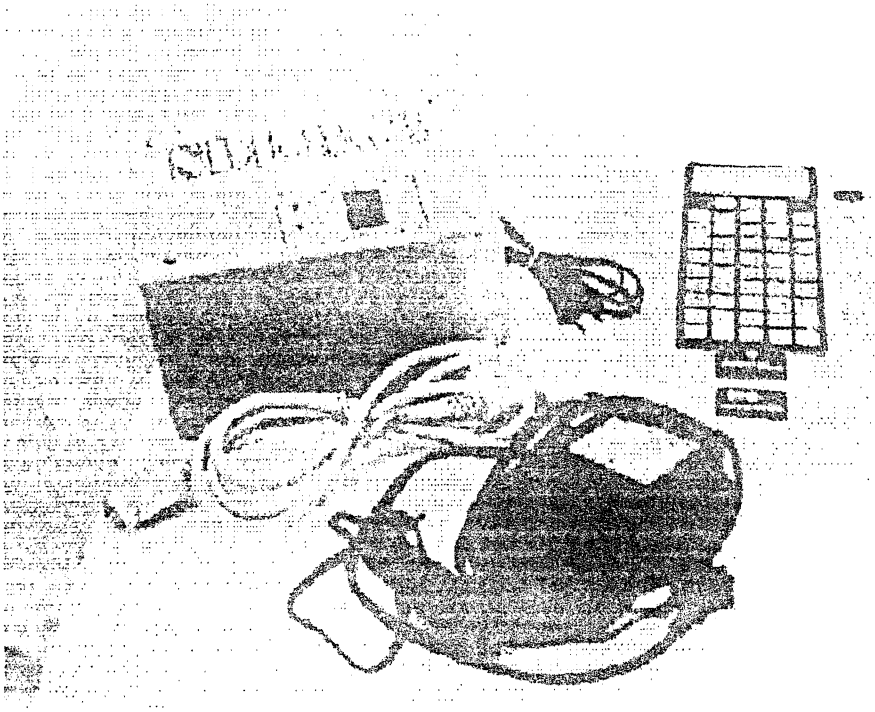


Figura 2: Presentación de los módulos

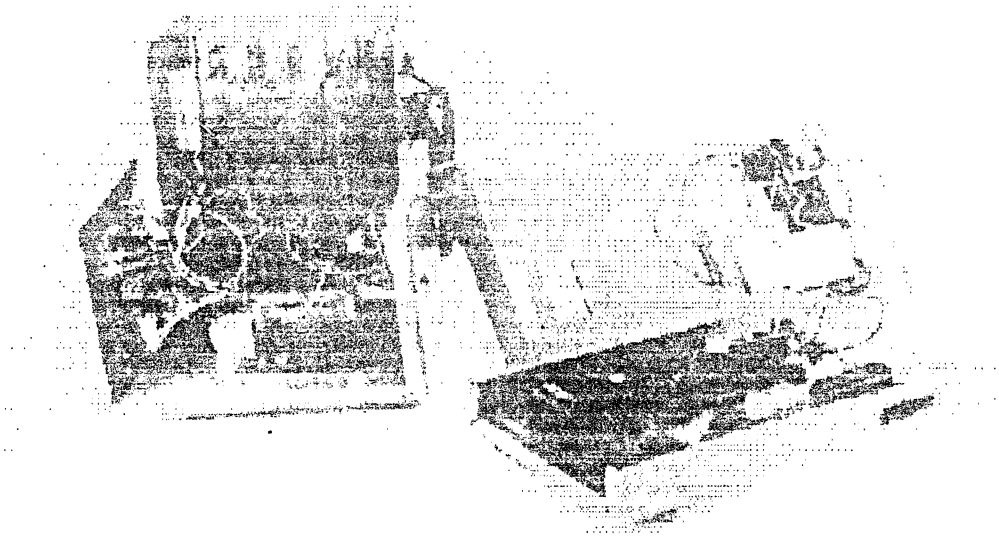


Figura 3: Presentación interna

1.1.3. SDI - 86S

El SDI – 86S (Ver Figuras 2 y 3), es un hardware que constituye el sistema de aplicaciones del SDI - 86, que no es otra cosa que una microcomputadora que permite realizar programas de hasta 8K de segmento de código, 3K de datos y 1K de pila. Físicamente el SDI - 86S tiene 16K de ROM y 4K de RAM, pero 4K se reservan para el sistema operativo, 4K para una copia de ROM en RAM, ésto permite que el usuario no tenga que ingresar sus datos manualmente en RAM, quedando 12K para el usuario, separados de la manera indicada al inicio del párrafo. El SDI - 86S contiene un sistema operativo de 778 bytes. Este se encarga de inicializar el 8279 utilizado para el manejo del teclado y el 8255 que conecta el display HD-44780 con el microprocesador 8086 mediante los puertos A y C alto, ambos configurados para salida, utilizando el A para datos o instrucciones y el C alto para señales de control. Dejando un puerto de entrada de 4 bits (Puerto C bajo) y un puerto de salida de 8

bits (Puerto B), disponibles para el usuario. Además, contiene las funciones necesarias para manejar el teclado y el display, así como bloques de instrucciones dedicados para realizar la copia de ROM a RAM del segmento de datos, hacer una relocalización del segmento de datos antes y después de ejecutar un programa, definir los inicios del segmento de código, de pila y extra. El segmento de pila inicia en la localidad 4C00h y el extra en la localidad 0000h, éste es utilizado para poder manejar la tabla de datos del display que se encuentra en ROM, necesarios para realizar la función de borrar pantalla.

1.2. FUNCIONAMIENTO DEL SDI - 86

Primero debe utilizar el SDI – 86P para escribir un programa en el ensamblador del 8086, luego de depurarlo debe ejecutarlo con simulación, ésta opción permite simular el sistema de aplicaciones (SDI - 86S). Una vez que el programa trabaja en la manera deseada, se puede crear un archivo compatible, el cual al grabarse en 2 EPROMs, hará que el SDI - 86S trabaje tal como lo vió en la simulación. Esto se logra porque se tienen funciones que realizan la simulación del SDI - 86S programadas en ensamblador por el SDI - 86P automáticamente, cuyo código es reemplazado luego por el código necesario para que el SDI - 86S trabaje de la misma manera como se lo ha simulado.

La creación del archivo compatible, es necesaria porque el SDI - 86P permite que el usuario tenga una estación de trabajo en su casa, siempre y cuando tenga una computadora cargada con Windows de Microsoft, necesario para correr el SDI - 86P. Por lo tanto, el sistema operativo bajo el que corre el SDI - 86P no es compatible con el que utiliza el SDI - 86S, que es un sistema operativo creado por nosotros y que maneja periféricos diferentes.

1.3. VENTAJAS QUE OFRECE

Para desarrollar un sistema electrónico basado en un microprocesador, es necesario construir una microcomputadora con él, junto con la circuitería adicional que va a ser controlada por éste. Además, se requiere de un programa de control que debe ser almacenado en una memoria tipo ROM, el cual también debe encargarse de inicializar y manejar los periféricos conectados a la microcomputadora, este debe ser editado, depurado, compilado y enlazado en un entorno de trabajo para crear programas en lenguaje ensamblador del microprocesador utilizado. A continuación, se detallan las ventajas que ofrece el SDI – 86 con relación a lo anteriormente dicho:

- Con el SDI – 86 el usuario no necesita construir una microcomputadora para realizar sus aplicaciones, puesto que cuenta con el SDI – 86S.

- El programa de control del usuario será más sencillo, debido a que no requiere incluir código para inicializar y manejar los periféricos conectados al SDI – 86S, labor que es realizada por el sistema operativo del SDI – 86S.

- No requiere un grabador de memorias EPROMs, ya que cuenta con el SDI – 86G para esto.

- Posee el SDI – 86P como un módulo para editar, depurar, compilar, enlazar, simular y utilizar el SDI – 86G para grabar el programa de control del usuario.

Otras ventajas:

- Pese a que el SDI – 86 fue diseñado para desarrollar aplicaciones electrónicas con el 8086, el SDI – 86P puede ser utilizado como un entorno de trabajo para programar en lenguaje ensamblador, sin importar si se quiere realizar una aplicación electrónica o no.

- El SDI – 86P puede ejecutarse en computadoras con Windows 3.1 o superior.

- El SDI – 86P contiene una ayuda completa para Windows, la cual puede ser ampliada si se desea.

- Sus tres módulos son de fácil transportación.

1.4. COMPATIBILIDAD ENTRE EL SDI – 86P Y EL SDI – 86S

Debido a que el SDI – 86P y el SDI – 86S trabajan con sistemas operativos diferentes, se necesita lograr que las aplicaciones creadas en el SDI – 86P funcionen igual ejecutándose en el SDI – 86S, consiguiéndolo simulando este en el SDI – 86P y utilizando 2 EPROMs 2764 para grabar el código de la aplicación del usuario con el sistema operativo del SDI – 86S, grabadas en el SDI – 86G conectado al PC mediante el puerto paralelo. Antes de grabar las EPROMs el SDI – 86P reemplaza las subrutinas de simulación del SDI – 86S por subrutinas que harán al SDI – 86S funcionar igual a la simulación cuando se ejecute una aplicación, logrando una compatibilidad e independencia entre estas dos partes.

CAPITULO II

DISEÑO DEL SDI-86

En el capítulo anterior se hizo una introducción al **SDI - 86** y se explicó brevemente cada una de sus partes o módulos. En este capítulo se describirá el funcionamiento y desarrollo de cada uno ellos, primero se abarcará al **SDI – 86P**, luego al **SDI – 86G** y finalmente al **SDI – 86S**.

2.1. DESCRIPCION DEL SDI – 86P

El SDI – 86P posee algunas características para permitir al usuario crear los programas de control de su aplicación electrónica, pese a ello puede ser utilizado como un entorno de trabajo para programar en lenguaje ensamblador, con ciertas características que permiten reemplazar al DEBUG del DOS.

2.1.1. REEMPLAZANDO AL DEBUG DE MICROSOFT

El programa **DEBUG** es un programa creado por Microsoft, que lo incluyó en su sistema operativo **DOS**, el cual permite visualizar la memoria, introducir programas en ella y rastrear su ejecución. El usuario puede utilizar el **SDI - 86P** para reemplazar al **DEBUG**, realizando estas funciones de una manera más sencilla.

2.1.1.1. VISUALIZACION DE LA MEMORIA CON EL SDI - 86P

Para ver la memoria se puede elegir la opción "Ver Contenido de Memoria" del menú "Herramientas", Se debe elegir el segmento y desplazamiento de las localidades que se desean ver, los cuales deben ser ingresados en decimal. Aparecerá un mapa de memoria con las localidades seleccionadas y su contenido organizado en pantallas. Se puede avanzar hasta 20 pantallas más, elegir otro segmento y/o desplazamiento o salir.

Por ejemplo, si queremos ver el segmento 40h y desplazamiento 00h, debemos elegir la opción "Ver Contenido de Memoria" del menú "Herramientas" y aparecerá lo siguiente:

Ingrese el segmento (en Decimal de 0 a 65535):?

Debemos escribir 64 y presionar [ENTER], luego aparecerá:

Ingrese el desplazamiento (en Decimal de 0 a 65535):?

Debemos escribir 0 y presionar [ENTER], luego aparecerá lo que se muestra en la figura 4.

Si presionamos la tecla "A" avanzaremos otra pantalla de memoria, sólo hasta 20 pantallas. Si presionamos "R" retrocederemos pantallas, sólo hasta la pantalla inicial. Si presionamos "E" podemos elegir otro segmento y/o desplazamiento y si presionamos "Q" salimos de la opción.

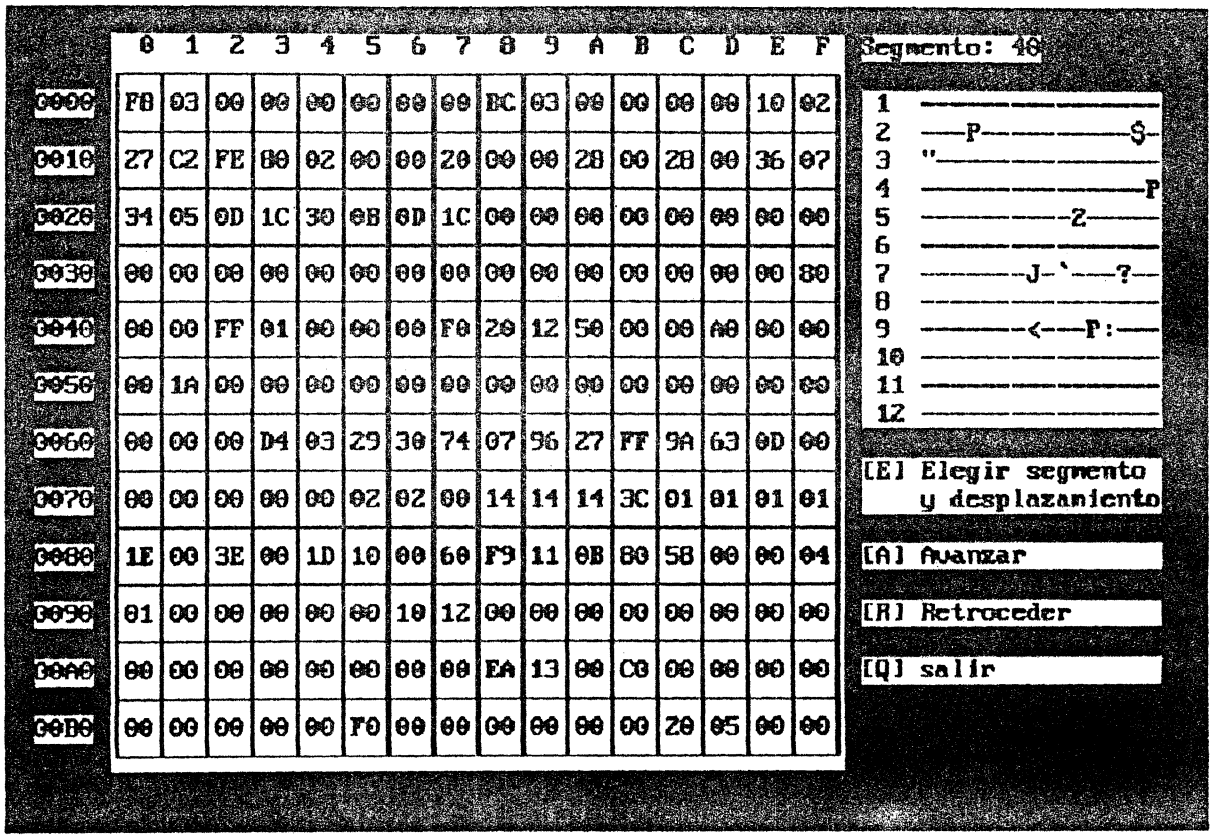


Figura 4: Interface del programa de visualización de la memoria

2.1.1.1.1. FORMA COMO LOGRA VER LA MEMORIA

El **SDI - 86P** llama a un programa realizado en Qbasic de Microsoft, al cual lo hemos llamado "memory.exe" cuyo código fuente se encuentra en el archivo "memory.bas". Este programa contiene 7 subrutinas, las cuales se detallan a continuación:

La subrutina *Celda* se encarga de crear una celda y ubicarla en la posición especificada en la pantalla, pudiendo incluir texto en su interior, esto lo hace mediante las variables Row1, Col1, Row2, Col2, text$. Row1 para la fila inicial, Row2 para la fila final, Col1 para la columna inicial, Col2 para la fila final y text$ para el texto en su interior.

La subrutina GRID se encarga de llamar varias veces a Celda para formar un GRID, con los datos del contenido de la memoria real, extraídos por la subrutina VerMem y almacenados en el arreglo dat$(), que es el arreglo de entrada para la subrutina GRID.

La subrutina *Etiq$* se encarga de colocar las etiquetas respectivas para las columnas y las filas, Desp$() son los desplazamientos calculados a partir del desplazamiento inicial seleccionado por el usuario.

La subrutina *Actualizar* permite al usuario ingresar el segmento y desplazamiento en la memoria a la que desea tener acceso.

La subrutina *VerMem* se encarga de extraer el contenido de la memoria seleccionada por el usuario, en un arreglo llamado Desp$(). Dspl es el desplazamiento seleccionado por el usuario, pero transformado a valor hexadecimal.

La subrutina *Direcc* se encarga de que los valores de dirección de desplazamiento siempre contengan 4 dígitos hexadecimales, completando con ceros a la izquierda, si éste valor contiene una longitud inferior de dígitos.

La subrutina *PrintCar*, imprime un guión (-) en el caso de que un caracter no sea un ASCII imprimible.

El programa principal llama a las subrutinas antes mencionadas para crear la presentación del entorno y mostrar los contenidos de las localidades de memoria seleccionadas por el usuario, así como controlar el funcionamiento de los caracteres de control utilizados en el programa.

2.1.1.2. RASTREANDO LA EJECUCION DE UN PROGRAMA CON EL SDI-86P

Primero debemos escoger la opción "Definir Segmentos" del menú "Herramientas", hacer click en la opción "Incluír salida al D.O.S.", aparecerá una "x" en ella, después presionar el botón "Aceptar". Luego escoger si el texto queremos que se presente en mayúsculas y minúsculas, sólo mayúsculas o sólo minúsculas. Puede escoger sólo minúsculas y presionar el botón "Aceptar", después de esto puede ingresar el código del programa luego de la línea "mov ds,ax", sin preocuparse por la definición de segmentos, la cual ha sido realizada automáticamente por el programa.

Ahora ya puede escribir un programa en el ensamblador del 8086, como el siguiente:

"MOV AX,0123h
ADD AX,0025h
MOV BX,AX
ADD BX,AX
MOV CX,BX
SUB CX,AX
SUB AX,AX
NOP"

Nota: No incluya las comillas.

Fuente: Este ejemplo fue tomado del libro **"LENGUAJE ENSAMBLADOR Y PROGRAMACION PARA IBM PC Y COMPATIBLES", Tercera Edición, Peter Abel, PRENTICE-HALL HISPANOAMERICANA S. A., 1996, Pag. 33.**

Luego de ingresado el programa elija la opción "Ejecutar Paso a Paso" del menú "Depuración" y siga las instrucciones que le da el programa.

Si desea grabarlo, elija la opción "Guardar" del menú "Archivo" y póngale el nombre que desee como "Ejemplo.asm".

2.1.1.2.1. FORMA COMO LOGRA RASTREAR UN PROGRAMA

El **SDI - 86P** genera automáticamente un programa depurador en ensamblador llamado 8086dep.asm (Ver anexo 1), el mismo que lo compila y enlaza automáticamente, recurriendo para esto al Turbo Ensamblador de Borland. Este contiene todo el código necesario para depuración, así como el código del programa del usuario con el formato adecuado para su depuración. Luego de que el usuario ve los cambios que ocurren en los registros y banderas con las instrucciones, el **SDI - 86P** destruye los archivos: 8086dep.asm, 8086dep.map, 8086dep.lst, 8086dep.obj y 8086dep.exe. Es necesario destacar, que el programa 8086dep.exe no altera de ningún modo el funcionamiento de las instrucciones especificadas por el usuario, por cuanto los valores de banderas y registros son los mismos antes y después de la ejecución de la depuración de cada instrucción.

Las subrutinas utilizadas para la depuración se detallan a continuación:

La mnuRunStep_Click ocurre cuando se hace un click en la opción Ejecutar paso a paso, utiliza la subrutina ActCont para actualizar el contenido del programa cada que sea necesario, anexando los códigos de las macros y subrutinas necesarias para realizar la depuración, éstas son: Las subrutinas Espacio, Tecla, Nuevo y Desp; las macros ClrScr, Cursor, Vereg y Depurar. Estas son anexadas siempre y cuando el programa a depurar no las utilice, ya que de lo contrario se generaría un error. La mnuRunTarget_Click ocurre cuando se hace click en la opción Ejecutar con objetivo, la diferencia de ésta con la primera es que mientras la primera llama a la macro Depurar por cada línea de código escrita por el usuario, la última busca por la palabra Objetivo y solamente en éstas líneas llama a la macro Depurar. Ambas utilizan subrutinas como ElimCom que es para eliminar comentarios y la subrutina LeftET que elimina tabuladores a la izquierda. Además, definen los datos necesarios para la

depuración, junto con los definidos por el usuario en su programa. Una vez que el archivo con el código necesario para depuración ha sido anexado, éste es grabado en un archivo 8086dep.asm, el cual luego es compilado, enlazado y ejecutado por el **Turbo Ensamblador** al ser invocado por el **SDI - 86P**, para esto se debe realizar una comunicación entre el **SDI - 86P** y el **Turbo Ensamblador de Borland**.

2.1.2. COMUNICACIÓN DEL SDI - 86P CON EL TURBO ENSAMBLADOR

La comunicación entre el **SDI - 86P** y el Turbo Ensamblador de Borland es realizada mediante el archivo tesis.bat (ver anexo 1) creado para ésto, el cual luego de realizar su labor es destruido por el **SDI - 86P**. Este invoca al Turbo Ensamblador para compilar, enlazar y ejecutar cualquier archivo, ya sea uno creado por el usuario o uno creado por el **SDI - 86P**. Los archivos creados por el **SDI - 86P** son los archivos 8086dep.asm utilizado para la depuración y el Sim8086.asm (ver anexo 1) usado en la simulación del **SDI - 86S**. La subrutina mnuRunNoSim_Click crea el archivo tesis.bat para la ejecución del archivo previamente compilado y enlazado por el Turbo Ensamblador invocado por el **SDI - 86P**, tarea que es realizada por la subrutina mnuMakeEXE_Click dentro del **SDI - 86P**, la cual es ejecutada cada vez que se hace click en la opción Crear archivo ejecutable.

2.1.3. UTILIZACION DEL SDI - 86P PARA ENTENDER LOS NEMONICOS

El **SDI - 86P** también puede ser usado para comprender el funcionamiento de cada uno de los nemónicos del 8086 o los principales, sin la necesidad de recurrir a un libro para esto. Para cumplir este objetivo, es necesario realizar pequeños programas con la utilización de cada uno de los nemónicos que se desee.

Por ejemplo, ejecute el **SDI - 86P** y si se quisiera ver cómo trabaja la instrucción "mov bx,05h", elija la opción "Definir Segmentos" del menú "Herramientas", escriba la instrucción luego de "mov ds,ax"

y elija la opción Ejecutar Paso a Paso. Igual si se deseara ver el funcionamiento de otras instrucciones como "add ax,cx", "sub ax,bp", "xor bx,cx", etc., se necesitarían programas de 3, 4 o hasta 5 líneas nada más. Esto hará que de una manera visual y rápida se pueda comprender el funcionamiento de los nemónicos, que es la parte principal para aprender a programar eficientemente en ensamblador.

2.1.4. HERRAMIENTAS ADICIONALES INCORPORADAS EN EL SDI - 86P

A parte de presentar la ventaja de no preocupar al usuario en la definición de segmentos, por cuanto el programa lo hace automáticamente, acción que es realizada por la *forma Segments* la cual permite la inclusión de algunas subrutinas importantes en caso de necesitarlo, datos en el segmento destinado para ellos, definición de la pila y salida al DOS (Disk Operating System) de Microsoft, esto último para permitir que el programa no termine de una manera inadecuada. Ofrece otras ventajas, permite agregar subrutinas y macros realizadas con anterioridad, lo que hace que el usuario pueda tener a su disposición una librería de ellas en el momento que las necesite. Además, permite llamar dentro del programa a la calculadora de Windows, por cuanto al momento de programar suele ser necesario realizar cálculos y conversiones, la cual es llamada por la subrutina mnuTCalc_Click, lo que ocurre cada que se hace click en la opción "Calculadora" del menú "Herramientas". La subrutina mnuSegments_Click se encarga de llamar a la forma segments, cuando se hace click en la opción "Definir Segmentos". También se puede ver las características del sistema al hacer click en la opción "Ver Información del Sistema", dándose lugar a la ejecución de la subrutina mnuTSysInf_Click .

Se ha creado una forma que se encargará de permitir al usuario tener acceso a una librería de subrutinas y macros para el ensamblador, así también como editar, añadir y eliminar las mismas. La mencionada librería está almacenada en una base de datos en formato de Microsoft Access llamada LIB8086.MDB. El proceso de interacción entre la base de datos y la forma se realiza a través del control personalizado llamado DATA CONTROL el cual tiene el motor de Access incorporado y da soporte a los más populares formatos de bases de datos. Esto se realiza con el propósito de facilitar

la programación en ensamblador.

2.1.5. TIPOS DE AYUDA INCORPORADAS EN EL SDI - 86P

El SDI - 86P posee una ayuda completa para Windows, la cual ha sido creada en la versión shareware de **helpMATIC V2.0**. Esta es invocada al elegir la opción Contenido del menú Ayuda, ejecutándose la subrutina mnuHelpItem_Click. También se posee de una línea de ayuda en el programa que sirve para informar al usuario las acciones que se están realizando, así como las que debe realizar en determinados casos. A más de esto, el SDI - 86 enviará mensajes de ayuda a través de pequeñas ventanas cada vez que sea necesario, haciendo que su uso sea más fácil.

2.1.6. RESUMEN DE OPCIONES DEL SDI – 86P

Este software permite al usuario desde una interfaz gráfica de usuario ejecutar todas las opciones disponibles en el mismo, tales como abrir archivos, guardarlos, compilar, crear ficheros ejecutables, ejecutar programas paso a paso, con objetivo, editar y/o ingresar subrutinas y/o macros, ver la memoria, obtener ayuda del sistema, etc.

El programa tiene disponible las siguientes opciones organizadas de la siguiente forma:

Archivo:

- **Nuevo**
- **Abrir**
- **Guardar**
- **Guardar Como**
- **Cerrar**

- Imprimir
- Configurar Impresora
- Salir

Edición:

- Deshacer
- Cortar
- Copiar
- Pegar

Borrar

- Seleccionar Todo
- Preferencias

Configurar:

- Barra de Herramientas
- Barra de Estado Estándar
- Colores:
  - Color de Fondo
  - Color de Primer Plano
  - Por Defecto
- Fuentes

Buscar:

- Buscar
- Búsqueda Siguiente
- Buscar Archivo

Heramientas:

- Definir Segmentos
- Utilizar Macros
- Utilizar Subrutinas

Calculadora

Información del Sistema

Colocar Objetivos

Limpiar Objetivos

Eliminar Archivo

Renombrar Archivo

Crear Archivo Compatible

Crear Archivo Ejecutable

Ejecutar Programa

Ver Código Máquina

Grabar Código Máquina

Leer EPROM

Ver Contenido de Memoria

Depuración:

Salida en Pantalla

Chequer Programa

Ejecutar Paso a Paso

Ejecutar con Objetivos

Ayuda:

Contenido

Buscar Ayuda acerca de

¿Cómo Usar Ayuda?

Acerca de

Archivo:

El menú Archivo permite realizar todas las funciones que involucran archivos (ficheros) tales como abrir, guardar, cerrar, imprimir, etc.

Nuevo.- Esta opción nos permite crear un nuevo archivo, teniendo en cuenta que se cerrará el programa actualmente en memoria en caso de haberlo ya que este programa solo permite abrir un programa a la vez.

Abrir.- Esta opción nos permite abrir un archivo ya existente, teniendo en cuenta que se cerrará el programa actualmente en memoria en caso de haberlo ya que este programa solo permite abrir un programa a la vez.

Guardar.- Esta opción nos permite salvar (grabar) el archivo actualmente en memoria bajo el nombre que el usuario desee. Es recomendable grabar el programa actual ya que en caso de generarse un error el trabajo no se perderá.

Guardar Como.- Esta opción es exactamente igual a la anterior con la diferencia que nos permitirá especificar un nuevo nombre para el archivo.

Cerrar.- Esta opción nos permite cerrar el archivo actualmente en memoria en caso de que así se desee.

Imprimir.- Esta opción nos permite imprimir ya sea el programa fuente, objeto (lenguaje máquina) del archivo que está actualmente en memoria.

Configurar Impresora.- Esta opción nos permite configurar la impresora por defecto que Windows tiene asignada.

Salir.- Esta opción nos permitirá salir del programa. En caso de existir modificaciones posteriores a la última vez que se grabó el archivo será preguntado si desea salvar las modificaciones. Cabe señalar que el programa esta diseñado para tener una sola copia del mismo a la vez.

Edición:

El menú Edición permite realizar todas las funciones que involucran edición de archivos (ficheros) tales como deshacer, cortar, pegar, etc...

Deshacer.- Esta opción nos permitirá deshacer el último cambio hecho al programa en caso de así desearlo.

Cortar.- Esta opción permite cortar un bloque seleccionado para enviarlo al Clipboard del Windows.

Copiar.- Esta opción nos permitirá copiar el bloque marcado al Clipboard de Windows.

Pegar.- Esta opción nos permitirá copiar el contenido del Clipboard de Windows al lugar donde se encuentra actualmente el punto de inserción (cursor). Hay que tener en cuenta que el Editor de este programa solo acepta líneas de texto y por lo mismo copiar cualquier otra cosa generará un error.

Borrar.- Esta opción permite borrar el caracter inmediatamente a la derecha del punto de inserción del cursor. En caso de haber un bloque seleccionado este se borrará completamente.

Seleccionar Todo.- Esta opción permite seleccionar todo el contenido del archivo que se encuentra actualmente cargado en memoria.

Configurar:

El menú Configurar permite configurar el color tanto de primer como de segundo plano, elegir el tipo de fuente, asignar valores iniciales a los registros internos del Microprocesador o ingresar datos en la memoria.

Barra de Herramientas.- Esta opción permite mostrar o esconder la barra de herramientas estándar del programa.

Barra de Estado Estándar.- Esta opción permite mostrar o esconder la barra de estado del Programa.

Colores.- Esta opción permite cambiar el color tanto de primer plano (Foreground) como de segundo plano (Background) de acuerdo a la paleta de Windows. Es recomendable dejar los colores asignados por defecto ya que otra asignación de colores podría causar excesivo cansancio a la vista

Color de Fondo.- Cambia el color de Fondo.

Color de Primer Plano.- Cambia el color de Primer Plano.

Por Defecto.- Regresa a la combinación de colores inciales.

Fuentes.- Esta opción permite cambiar la fuente asignada por defecto al editor del programa de acuerdo a las fuentes que tenga disponible Windows en ese instante.

Buscar:

El menú Buscar contiene opciones que permiten realizar la búsqueda de palabras dentro del texto de acuerdo a un formato determinado o de archivos en las diferentes unidades del sistema.

Buscar.- Esta opción permite realizar una búsqueda de una palabra o conjunto de palabras dentro del archivo que se encuentra actualmente cargado en memoria. Hay que tener presente que las búsquedas siguientes de la misma selección se realizará simplemente presionando la tecla de función F3.

Búsqueda Siguiente.- Esta opción permite realizar búsquedas consecutivas del texto seleccionado. La búsqueda se puede realizar en orden ascendente o descendente a través del contenido del archivo. La opción "Match Case" permite realizar la búsqueda respetando la sintaxis del texto seleccionado.

Buscar Archivo.- Esta opción permite realizar la búsqueda de un archivo por su nombre. Hay que tener presente que la búsqueda solo se realizará desde el directorio actual y todos sus subdirectorios, por lo que es conveniente ubicarse en el directorio raíz de la unidad en el caso de no tener ningún indicio del lugar en el que se encuentra el archivo.

Heramientas:

El menú Herramientas permite utilizar opciones que realizan diferentes acciones, tales como llamar a la calculadora de Windows, obtener información del Sistema y borrar archivos que ya no sean necesarios.

Definir Segmentos.- Por medio de esto podremos definir los segmentos del programa, ya sean estos de datos, pila o código. Esto también puede hacerse por medio de programación y dependerá del usuario elegir cual de las dos formas es la más conveniente para sus necesidades.

Utilizar Macros.- Por medio de ésta opción podremos invocar a las macros presentes en la librería del programa. También podremos editar y eliminar funciones anteriormente creadas o crear unas nuevas.

Utilizar Subrutinas.- Por medio de ésta opción podremos invocar a las subrutinas presentes en la librería del programa. También podremos editar y eliminar funciones anteriormente creadas o crear unas nuevas.

Calculadora.- Esta opción permite llamar a la calculadora de Windows directamente desde el Programa.

Información del Sistema.- Esta opción permite obtener información referente al Sistema en un instante determinado.

Colocar Objetivos.- Por medio de esto podremos usar delimitadores los cuales nos permitirán colocar objetivos en la ejecución del programa para de ésta forma analizar el resultado del programa en determinados puntos del mismo.

Limpiar Objetivos.- Por medio de esto podremos eliminar los delimitadores, los cuales nos permitían ejecutar el programa en forma parcial o también llamado por objetivos. Luego si el usuario desea puede volver a colocar otros delimitadores para de ésta forma analizar el programa en otros puntos del mismo.

Eliminar Archivo.- Esta opción permite eliminar archivos que ya no se consideren necesarios. Hay que tener presente de no borrar archivos que presenten una gran importancia para la ejecución del sistema.

Renombrar Archivo.- Esta opción nos permite cambiar el nombre de un archivo.

Crear Archivo Compatible.- Esta opción nos permite crear un archivo compatible con el microcomputador diseñado (SDI - 86S), lo que nos permitirá que los programas que hemos realizado se puedan verificar en forma física.

Crear Archivo Ejecutable.- Esta opción nos permite crear un archivo ejecutable (COMPILACION + ENLACE) del archivo que está activo actualmente. En caso de haber errores durante el proceso, se notificará de los mismos.

Ejecutar Programa.- Esta opción nos permite ejecutar el programa que actualmente esta en memoria en caso de que la compilación hubiera sido un éxito o que anteriormente se haya creado un archivo ejecutable.

Ver Código Máquina.- Por medio de esta opción podremos visualizar el código máquina del programa que está actualmente cargado en memoria. Al mismo tiempo se presentará el código fuente para de ésta forma tener una mejor comprensión del mismo. Luego en caso de desearlo podremos grabar el código máquina en una memoria.

Grabar Código Máquina.- Este módulo nos permitirá grabar dos memorias EPROM con el código máquina correspondiente al programa (código fuente) que actualmente esta en la memoria.

Leer EPROM.- Por medio de esto podremos leer una memoria EPROM, para de ésta forma verificar que lo que se envió a grabar estaba correcto.

Ver Contenido de Memoria.- Esta opción nos permite ver el contenido de la memoria en cualquier instante.

Depuración:

El menú Depuración permite chequear el programa antes de ejecutarlo (requisito indispensable).

Salida en Pantalla.- Esta opción nos permite recuperar en pantalla la salida generada por un programa al ejecutarse el mismo.

Chequear Programa.- Esto nos permite realizar un chequeo del programa que esta actualmente cargado en memoria.

Ejecutar Paso a Paso.- Esto nos permitirá ejecutar el programa línea a línea de código para de ésta manera poder ver cómo se altera el contenido de los registros internos del microprocesador y analizar los errores (depurar) en caso de haber.

Ejecutar con Objetivos.- Esto nos permitirá ejecutar el programa en forma íntegra o parcial dependiendo del gusto del usuario. Para poder realizar esto el programador deberá indicarle al programa desde dónde hasta dónde quiere observar la ejecución del mismo por medio del uso de delimitadores (colocación de objetivos).

Ayuda:

El menú Ayuda nos permite obtener información acerca de este programa y por lo mismo invocará este sistema de ayuda.

Contenido.- Esta opción permite visualizar el contenido de este sistema de ayuda.

Buscar Ayuda acerca de.- Esta opción permite buscar ayuda sobre un tópico específico en este sistema de ayuda.

¿Cómo Usar Ayuda?.- Esta opción permite obtener pautas para usar el sistema de ayuda.

Acerca de.- Esta opción informa acerca de la realización de este programa.

2.1.7. DEFINICION DE SEGMENTOS CON EL SDI - 86P

Al empezar a trabajar con el SDI - 86P, el usuario tiene dos opciones: Llamar a un archivo creado con anterioridad o crear uno nuevo, para llamar a un archivo el usuario tiene la opción "Abrir" en el menú "Archivo" y para crear un nuevo archivo tiene primero que elegir la opción "Definir

Segmentos" del menú "Herramientas". Al elegir esta opción aparecerá la ventana mostrada en la figura 5.

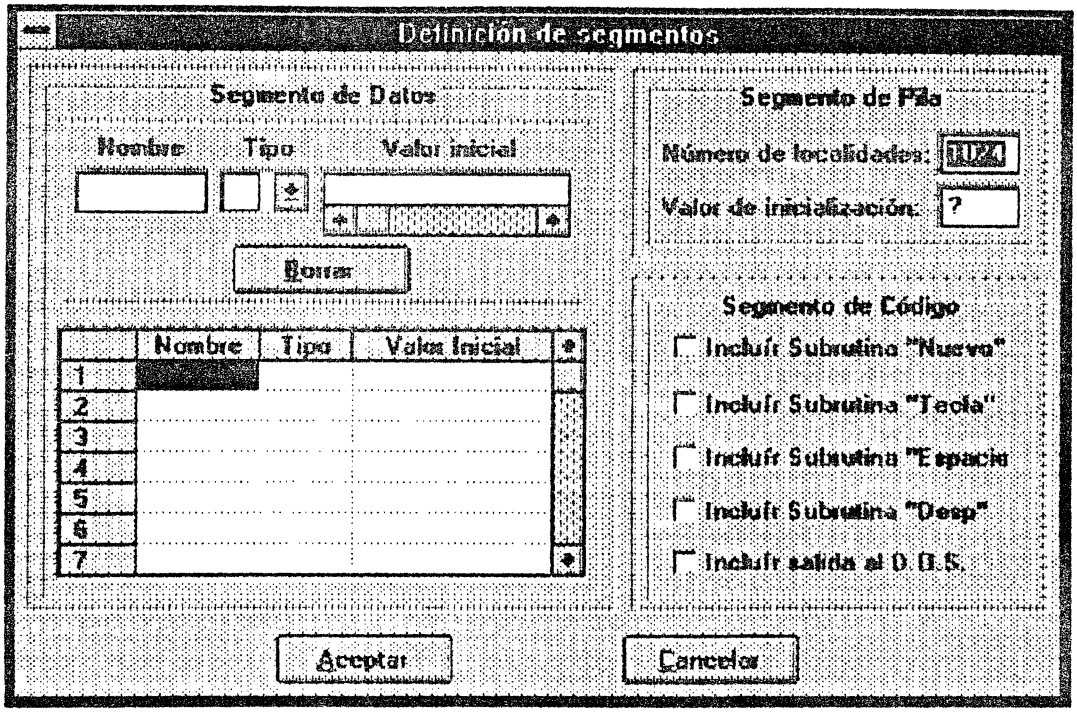


Figura 5: Interface de la definición de segmentos

En el campo del segmento de datos el usuario puede definir las variables que formarán parte de éste, escribiendo su nombre, eligiendo su tipo (DB, DW, DD, DF, DQ o DT) y el valor inicial que le desea dar. Luego de esto debe hacer click en el lugar del GRID donde desea ubicarla. En caso de querer borrar un dato mal escrito, debe presionar el botón "Borrar" y hacer click en el lugar donde se encuentra el dato a borrar.

En el campo del segmento de pila, debe definir el número de localidades que desea y el valor de inicialización. En el segmento de código, el usuario podrá incluir las subrutinas Nuevo, Tecla, Espacio y/o Desp, en caso de necesitarlas, así como una salida al D.O.S. para que el programa termine correctamente.

Luego de que los segmentos de datos, código y pila están definidos como lo quiere el

usuario, se debe presionar el botón "Aceptar" y el programa se encargará de escribir automáticamente el código necesario para que esto ocurra. De esta manera, el usuario puede comenzar a escribir el código de su programa luego de la línea "mov ds,ax", sin preocuparse por la definición de segmentos.

2.1.7.1. FORMA DE DEFINIR LOS SEGMENTOS

Los datos definidos por el usuario en el segmento de datos son mostrados en el *GridSD*, definido para esto. El segmento de pila utiliza las cajas de texto *TxtLocSP* para el número de localidades y *VinSP* para su valor de inicialización. El segmento de código contiene controles de opciones que se activan o desactivan, lo que le indica al **SDI - 86P**, las opciones que el usuario desea incluir el segmento de código. Al presionar el botón aceptar, se ejecuta el código de la subrutina *PrefOK_Click* la que se encarga de ubicar los datos respectivos en el segmento de datos, pila y código.

2.1.8. TRATAMIENTO DE ARCHIVOS CON EL SDI - 86P

Para el tratamiento de archivos el **SDI - 86P** tiene 5 opciones: "Nuevo", que permite al usuario crear un archivo nuevo tomándose en cuenta que por cada archivo creado se debe volver a definir el segmento de datos, lo que se realiza una vez por cada archivo creado. "Abrir", que permite abrir un archivo creado anteriormente, incluso cualquier archivo *.asm, teniendo en cuenta que si no es uno creado en **SDI - 86P**, se pueden producir errores al usar algunas opciones, como ejecutar paso a paso, ejecutar con objetivo, colocar objetivos, limpiar objetivos, etc. "Guardar y Guardar como", permiten guardar un archivo *.asm con el código fuente creado por el usuario, la diferencia entre la primera y la segunda, es que la primera graba siempre en el archivo actualmente abierto, mientras que la segunda permite al usuario especificar donde. "Cerrar", permite al usuario cerrar el archivo actualmente abierto.

2.1.9. IMPRESIÓN DE ARCHIVOS CON EL SDI - 86P

El **SDI - 86P** permite al usuario imprimir su código fuente, para esto tiene dos opciones: "Imprimir", que se encarga de enviar los datos a la impresora para su impresión y la opción "Configurar Impresora", la cual permite al usuario definir el tipo de impresora que tiene, tamaño del papel y su orientación.

2.1.9.1. FORMA DE REALIZAR EL TRATAMIENTO E IMPRESION DE ARCHIVOS

El programa **SDI – 86P** permite realizar funciones que involucran el manejo de archivos, impresión, tales como: Nuevo, Abrir, Guardar, Guardar Como,Cerrar, Imprimir, Configurar Impresora y Salir. Estos diferentes procedimientos se lo realiza a través de la **Subrutina FileItem(Index)** la cual es general a todos los procesos del menú Archivo.

Analizando la Subrutina observamos primeramente una sentencia **On Error Goto errFileItem** la que nos permitirá el tratamiento de errores dentro de la subrutina. Luego tenemos un lazo **Select Case Index**, donde la variable **Index** puede tomar valores desde 0 hasta 9 dependiendo de la acción a realizar. En casi todas las opciones tenemos la invocación a la función **TextoModificado** la cual se encarga de determinar si el archivo de texto (código fuente en ensamblador (*.asm)) ha cambiado desde la última vez que se grabó, y en caso de haber sufrido alteración envía un mensaje correspondiente. Esta función hace una llamada a la **API** (Aplication Program Interface) de Windows a través de la función **SendMessage(frmEditor!txtEdit.hWnd, EM_GETMODIFY, 0, 0&) * -1**. Es necesario mencionar que la presentación de la cajas de diálogo entre el usuario y el programa se la realiza a través del control personalizado **CMDIALOG.VBX**. Dependiendo de la acción realizada

tendremos que el sistema invoca dos subrutinas adicionales llamadas **OpenFile (Filename)** y **CloseFile(Filename)**, donde la variable global **Filename** tiene el nombre del archivo. Es necesario señalar que en el entorno de programación Visual Basic tenemos disponibles tres maneras de abrir y/o guardar archivos las cuales son Binaria, Aleatoria y Secuencial, nosotros hemos escogido ésta última aunque la que sea el modo Binario el que da más eficiencia. En el proceso **OpenFile** leemos el archivo en forma secuencial y se lo asignamos a la propiedad Text del objeto **frmEditor!txtEdit**. El proceso **CloseFile** hace el procedimiento contrario ya que permite grabar el contenido de la propiedad Text del objeto **frmEditor!txtEdit** a un archivo. El proceso de impresión y de cofiguración de impresión se lo hace a través del control personalizado **CMDIALOG.VBX**, el cual tiene unas propiedades para el manejo de impresión de archivos debido a que el mismo establece un enlace con la **DLL** (Dinamic Link Library) **COMMDLG.DLL**, el cual nos permite especificar el número de copias, páginas y tipo de impresora a utilizar. (Para más información revisar el Manual de Programación de Visual Basic).

2.1.10. EDICION DE TEXTO CON EL SDI - 86P

El menú "Edición" permite realizar todas las funciones que involucran edición de archivos (ficheros) tales como "deshacer", la cual nos permitirá deshacer el último cambio hecho al programa en caso de así desearlo, "cortar" la que nos permite cortar un bloque seleccionado para enviarlo al Clipboard del Windows, "copiar" la que permitirá copiar el bloque marcado al Clipboard de Windows, "pegar" la que nos permitirá copiar el contenido del Clipboard de Windows al lugar donde se encuentra actualmente el punto de inserción (cursor). Hay que tener en cuenta que el Editor de este programa sólo acepta líneas de texto y por lo mismo copiar cualquier otra cosa generará un error, "borrar" la encargada de borrar el caracter inmediatamente a la derecha del punto de inserción del cursor y "seleccionar todo" la que permite seleccionar todo el contenido del archivo que se encuentra actualmente cargado en memoria.

2.1.10.1 FORMA DE REALIZAR LA EDICION DE TEXTO

Analizando la Subrutina **EditItem(Index)** observamos primeramente una sentencia **On Error Goto errEditItem** la que nos permitirá el tratamiento de errores dentro de la subrutina. Luego tenemos un lazo **Select Case Index**, donde la variable **Index** puede tomar valores desde 0 hasta 7 dependiendo de la acción a realizar.

En el caso de la opción "Deshacer" esto se hace a través de la **API** de Windows por medio de la función **SendMessage(frmEditor!txtEdit.hWnd, EM_UNDO, 0, 0&)**, donde **frmEditor!txtEdit.hWnd** retorna el Handle de la ventana que contiene el archivo en código ensamblador y de esta manera Windows sabrá cual ventana invoca el servicio. La alternativa "Cortar" utiliza el objeto **Clipboard** y sus propiedades **Clear** y **SetText** las cuales vacían el contenido del **Clipboard** y la otra copia el texto seleccionado al **Clipboard** respectivamente. El caso de "Copiar" es el mismo que "Cortar" con la diferencia de que no se borra el texto seleccionado. Para la acción "Pegar" tenemos que utlizamos la sentencia **frmEditor!txtEdit.SelText = Clipboard.GetText()** la que copiara el contenido del **Clipboard** a la propiedad **Text** del objeto **frmEditor!txtEdit**. Para la opción "Borrar" tenemos que utilizamos **Screen.ActiveControl.SelStart** y **Screen.ActiveControl.SelLength** las cuales indicarán el inicio y el final del bloque a borrar y por medio de **Screen.ActiveControl.SelText = ""** finalmente se ejecuta la acción de borrar. Cabe señalar que **Screen.ActiveControl** apuntará al objeto **frmEditor!txtEdit**. Finalmente tenemos la opción "Seleccionar Todo" la que se realiza por medio de **frmEditor.txtEdit.SelStart = 0** y **frmEditor.txtEdit.SelLength = Len(frmEditor.txtEdit.Text)** las que apuntan al inicio y final del contenido del archivo de texto (código ensamblador) actualmente activo.

2.1.11. CONFIGURACION DEL SDI - 86P

Este menú presenta opciones como la de presentar y esconder una Barra de Herramientas la que presenta íconos de uso común como los de abrir archivos, guardar archivos, imprimir, cortar, copiar,

pegar, etc...; la Barra de Estado Estándar en la cual obtendremos información sobre el estado del número total de líneas, hora, etc...; Colores con la cual podremos cambiar el color de las letras y el fondo y finalmente Fuentes que nos permite cambiar la fuente asignada por defecto al editor del programa de acuerdo a las fuentes que tenga disponible Windows en ese instante.

2.1.11.1. FORMA DE REALIZAR LA CONFIGURACION

Primeramente hablaremos sobre cómo mostrar y esconder tanto las Barras de Herramientas y de Estado. Estas están ubicadas sobre los objetos llamados "cajas de imagen" y por medio de su propiedad Visible la cual toma los valores de True o False presentaremos o esconderemos las mismas. En lo que respecta a la selección del tipo de color y fuente del programa lo realizamos a través del control personalizado CMDIALOG.VBX que como se menciono antes establece un enlace con la librería COMMDLG.DLL y por medio de las propiedades del mencionado control le indicaremos nuestra selección. Así tenemos que por ejemplo que en frmEditor!CMDialog1.Color retorna el color elegido y usando *frmEditor!txtEdit.BackColor = frmEditor!CMDialog1.Color* ó *frmEditor!txtEdit.ForeColor = frmEditor!CMDialog1.Color* cambiaremos el color del Fondo ó del Primer Plano respectivamente. Los valores *frmEditor!txtEdit.ForeColor = RGB(0, 0, 0)* y *frmEditor!txtEdit.BackColor = RGB(255, 255, 255)* serán los valores por defecto considerados por nosotros en el momento que diseñamos el programa. Cuando se utiliza el control CMDIALOG.VBX se hace necesario el manejo de errores ya que cuando se presenta la caja de dialogo ya sea para abrir, guardar o imprimir archivos el elegir la alternativa de "Cancelar" generará un error y el programa se "caería".

2.1.12. BUSCANDO TEXTO Y ARCHIVOS CON EL SDI - 86P

La opción de "buscar" una palabra determinada dentro de un texto se hace necesaria en nuestro sistema, por lo que se consideró la implementación de las mismas. Adicionalmente se dispone de una alternativa para "buscar archivos" a través de las diferentes unidades del sistema, aunque

generalmente no es necesario ya que el software está diseñado para almacenar los archivos en el directorio en donde se encuentra ejecutándose el mismo ya que se hizo uso de **ChDir App.Path** y **ChDrive App.Path,** los que invocan al objeto **App.Path** que almacena la "trayectoria" (PATH) del programa **SDI – 86P** y con las funciones **ChDir** y **ChDrive** cambiamos al sitio en que se ejecutó el software el cual pasa en ese momento a ser el directorio activo.

Cuando es invocada la función "Buscar" tendremos que darnos cuenta si anteriormente estaba marcado alguna palabra o grupo de palabras, ya que de ser afirmativo tendremos que realizar la búsqueda de la misma. Por ello **frmEditor!txtEdit.SelText** nos da el contenido del bloque seleccionado. Si éste es una cadena nula ("") no se presentará nada en la ventana de búsqueda, caso contrario la presentaremos en ese sitio. Sea cual fuere el resultado la subrutina **FindIt** realizará el proceso de búsqueda. Para más detalles sobre este proceso remítase al apéndice que contiene el código fuente del programa escrito en Microsoft Visual Basic. En lo referente a la búsqueda de un archivo es necesario señalar que se ha realizado una "forma" llamada **WinSeek** y también será necesario remitirse al código fuente.

2.1.13. DESCRIPCION DEL SISTEMA DE AYUDA DEL SDI-86P

Para la elaboración del sistema de ayuda para el SDI - 86P se utilizó el Software helpMATIC Versión 2.00 el cual permite una construcción relativamente fácil del archivo en formato RTF (Rich Text Format) el que es absolutamente necesario para hacer uso del compilador de ayuda de la Microsoft HC31.EXE. El archivo de ayuda del sistema contiente además hypergráficos, los cuales análogamente a los hypervínculos permite hacer enlaces en forma no lineal dentro del arhivo de ayuda, sólo que en esta vez en forma gráfica. Los mencionados hypervínculos se realizaron con el módulo HotSpot Editor del entorno de desarollo Microsoft Visual Basic, el cual generará archivos con formato SHG el que tiene enlaces codificados directamente al archivo RTF. Este último se incopora con el archivo RTF antes del proceso de compilación y en caso de no haber errores habremos creado con éxito un archivo en formato HLP.

CAPITULO III

CONSTRUCCION DEL SDI - 86

3.1. DESCRIPCION DEL SDI-86G

Como se indicó anteriormente, una vez que se haya editado, depurado y compilado un programa en ensamblador, se puede escoger la opción de crear un programa compatible para poder ejecutarlo físicamente en el SDI-86S, lo cual resulta muy interesante porque se puede comprobar de una manera directa cómo funciona nuestro programa en un sistema pequeño, sin tantos componentes de hardware como los que posee un PC, lo cual permite asimilar más fácilmente la interacción entre el programa y el microprocesador.

Para poder correr un programa de ésta manera, se necesita de un medio de enlace entre el **SDI - 86P** y el **SDI - 86S**, ésto se logra mediante la grabación de un par de memorias *EPROMs 2764* (Figura 6) con el **SDI-86G**. Por tanto el hardware de éste sistema comprende el **SDI-86G** que se detalla a continuación y el **SDI-86S** que se describirá más adelante.

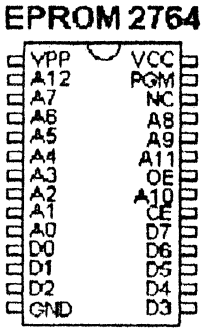


Figura 6: Distribución de pines de una EPROM 2764

3.1.1. CONTROL DEL SDI - 86G POR MEDIO DEL SDI - 86P

La grabación de las EPROMs es controlada mediante una subrutina, denominada "Grabar Código Máquina" del menú "Herramientas", la misma que desplegará un submenú para escoger el voltaje de grabación de las memorias tanto Par como Impar, dependiendo de la opción escogida se llamará a otra subrutina que se encarga de separar sea par o impar la información a grabar, luego se desplegará una ventana en la que se visualizará un GRID con el contenido de la información para ser grabada en la memoria seleccionada, ésta información corresponde al *Sistema Operativo* que se halla en el arreglo *OS*, el *programa del usuario* que se halla en el arreglo *ROM* y al *Segmento de datos* del arreglo *RAM*, ubicados en sus respectivas localidades, luego se procede a grabar mediante el botón correspondiente; en éste momento la subrutina envía el dato 8 por las líneas de control del puerto paralelo, que hace que se active la alimentación de la EPROM con Vcc, luego envía el dato 9 que activa el voltaje de grabación y finalmente se envían los datos 11, 12 o 13 que corresponde a 12.5, 21 o 25 Voltios (Vpp) de acuerdo a la opción escogida. Después de ésta configuración la subrutina empieza a enviar los datos 8 y 0 alternadamente, conjuntamente con la información de la dirección y el dato por las líneas de datos del puerto paralelo, es decir se efectúa una multiplexación de la información. Una vez terminada la grabación, el programa procede a desactivar la alimentación de Vpp y de Vcc (5 V), momento en el cual se retirará la EPROM.

También se puede realizar la lectura de una EPROM, para ésto se escoge la opción "Leer Eprom" del

menú "Herramientas", el cual presentará una forma con un GRID vacío que será llenado con el contenido de la EPROM leída. La subrutina que controla ésta opción funciona de manera similar a la anterior, con la diferencia de que se envía una señal para que el pin Vpp reciba solamente 5 voltios de alimentación igual a Vcc, luego se envían alternadamente los datos 8 y 0 conjuntamente con los datos de dirección de las localidades de la EPROM por el puerto paralelo, pero ahora se lee por las líneas de estado el contenido de la memoria, todo esto también de forma multiplexada; una vez leída la información el programa procede con la desactivación de la memoria, este proceso es más veloz que la grabación.

3.1.1.1. FUNCIONAMIENTO DEL SDI-86G.

El SDI-86G, es un Programador de memorias EPROM 2764 que está diseñado para funcionar para grabación y lectura, las memorias EPROM actuales funcionan con un voltaje de grabación de 12.5 V, pero hay varias que también funcionan a voltajes más altos como 25 V, existe un grupo específico de memorias del fabricante Intel cuya grabación corresponde a 21 V, cabe recalcar que si se graba una memoria a un valor superior al nominal, ésta se destruye. Tomando en cuenta este antecedente, el SDI-86G puede grabar a estos tres voltajes, los cuales serán conmutados de acuerdo a la elección del usuario, por tanto se debe asegurar cual es el voltaje de grabación de una determinada EPROM para evitar que ésta se destruya. Todas las acciones del Programador son controladas por medio de una rutina específica dentro del SDI-86P indicada anteriormente, la misma que se encarga de enviar las señales respectivas para la alimentación y conmutar las funciones de lectura o grabación con sus respectivos voltajes.

Este programador está dividido en tres partes específicas:

1. Sección de procesamiento de las señales del puerto paralelo
2. Sección de pulso de grabación
3. Sección de alimentación.

3.1.1.1.1. SECCIÓN DE PROCESAMIENTO DE LAS SEÑALES DEL PUERTO PARALELO

En el diseño del SDI-86G, se hizo uso del puerto paralelo como interface con el PC, el cual consta de 8 bits de datos (salidas), 4 bits de control (salidas) y 5 bits de estado (entradas), Ver Figura 7 y Tabla I.

Como se ha indicado se utilizan memorias EPROM 2764 cuya capacidad es de 8KB, por tanto posee una barra de 13 líneas de dirección y una barra de 8 líneas de datos, Ver figura 6.

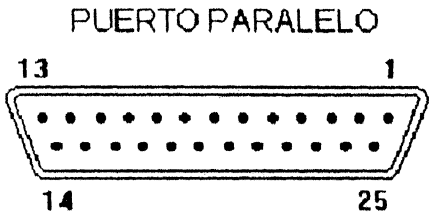


Figura 7: Numeración de los pines del puerto paralelo

PIN	GRUPO	BIT	TIPO
2	Dato	0	Salida
3	"	1	"
4	"	2	"
5	"	3	"
6	"	4	"
7	"	5	"
8	"	6	"
9	"	7	"
15	Estado	0	Entrada
13	"	1	"
12	"	2	"
10	"	3	"
11	"	4 *	"
1	Control	0 *	Salida
14	"	1 *	"
16	"	2	"
17	"	3 *	"
18-25	Tierras		

* Invertida.

Tabla I: Descripcion de los pines del puerto paralelo

Para una mejor comprensión de la estructura interna del SDI-86G podemos remitirnos al Diagrama esquemático (Ver anexo 2). El cual pasamos a explicar.

Como podemos fijarnos es necesario multiplexar la barra de datos del puerto de salida para poder comunicarnos con las dos barras de la EPROM, para esto se ha procedido a emplear tres 74LS374 (U3, U4, U5) los cuales son registros de 8 bits cada uno, los mismos que tienen sus entradas conectadas en paralelo provenientes de un buffer, el 74LS245 (U1) el cual recepta las señales de datos del puerto paralelo, U3 recibe los bits de direcciones más altos, U4 los bits de direcciones bajo y U5 recepta los datos, cabe recalcar que esto se aplica tanto para la grabación como para la lectura.

Para realizar la lectura de una EPROM sólo tenemos disponibles 5 bits de entrada en el puerto paralelo y no 8 como se desearía, por tanto es necesario multiplexar la barra de datos de la EPROM, es decir se realiza la lectura en grupos de 4 bits, para esto se emplea un 74LS244 (U7) que es un bufer cuyas entradas provienen de la barra de datos de la EPROM, y las salidas se conectan con un 74LS245 un bufer bidireccional (U6), el cual envía los datos al puerto paralelo, con esto ya no es necesario utilizar los 5 bits de estado sino solamente los 4.

Hasta aquí se ha explicado la forma de conectar la barra de direción y de datos de la EPROM con el puerto paralelo con la técnica de la multiplexación, pero para que esto funcione se necesita de la sincronización adecuada entre estos componentes, y la forma de conseguir esto es utilizando el grupo de líneas de control del puerto paralelo que tenemos todavía disponibles.

Los bits de control se utilizan no solamente para enviar las señales de sincronización para los registros que almacenan las direcciones y datos de la EPROM, sino que también se emplean para enviar señales de control para su alimentación, así como de seleccionar el voltaje de grabación de las mismas y también para seleccionar una lectura o grabación.

Para obtener las señales de control del puerto, hacemos uso del bufer U6 que tenemos disponible. A través de U6 enviamos las señales de alimentación y de selección de voltaje de grabación por intermedio de los tres primeros bits, quedando el restante, es decir el más significativo para la sincronización de la recepción de señales de las barras de dirección y datos, a través de U3, U4, U5 y U7 .

Las tres líneas correspondientes a los bits 0, 1 y 2 de control ingresan a dos decodificadores 74LS138 (U10 y U12) que se encargan de separar las señales de voltaje y de alimentación de las EPROM. Las salidas de U10 se conectan con inversores 74LS04 (U11) debido a que sus salidas se activan en bajo, luego estas ingresan a unas compuertas OR 74LS32 (U14) y a un contador binario 74LS161 (U16) el cual está configurado en modo carga, las compuertas OR se utilizan en combinación con la señal del bit más significativo de control por intermedio de una compuerta AND 74LS08 (U15), para controlar la carga de las señales provenientes de U10 hacia U16, este bit de control se emplea para controlar el flujo de señales por las barras, esto se explica más adelante. Las salidas de U16 envían señales para lectura o para voltajes de grabación, y lógicamente sólo puede estar activa una de ellas a la vez, la salida QA sirve para realizar una lectura de una EPROM, envía una señal de deshabilitación de U5, de ésta forma quedan activas las barras de dirrección y la barra de datos provenientes de la memoria, es decir que solamente están activos U3, U4 y U7. La salida QB activa el voltaje de grabación Vpp a 12.5 V, QC a 21 V y QD a 25 V.

Las salidas de U12 son utilizadas de forma complementaria con respecto a las salidas de U10, es decir U10 utiliza las salidas Y1, Y2, Y3 y Y4, mientras que U12 utiliza las salidas Y0, Y5, Y6 y Y7, las salidas Y0 y Y6 de U12 pasan por inversores (U13) y estas señales sirven para el clock de los dos Flip Flops tipo D del integrado 74LS74 (U17), las otras dos señales de U12 se necesita que se activen en bajo para el Clear de U17. Del primer Flip Flop se toma la salida complementaria (~Q) que envía una señal para activar la alimentación de la memoria (Vcc), la salida del segundo Flip Flop se utiliza para activar el Voltaje de grabación (Vpp).

El cuarto bit de control también se utiliza como entrada de clock para el contador binario 74LS161 (U8), del cual sólo se utilizan las dos salidas más significativas, es decir QA y QB, con esto se tiene una cuenta de 00, 01, 10 y 11, estas señales ingresan a un demultiplexor 74LS155 (U9), el cual distribuye una señal de habilitación a U3, U4, U5, U7 y a un inversor de U13, ésta señal proviene de *un detector de señal de datos del puerto paralelo*, que está conformado por una compuerta NAND de 8 entradas 74LS30 (U2), que están conectadas con la salida del bufer U1, la salida de la compuerta NAND se distribuye a un inversor y a una compuerta OR (U14), con esto se consigue que cuando haya una señal por cualquier línea del bufer sea detectada y enviada al demultiplexor.

3.1.1.1.2. SECCIÓN DE PULSO DE GRABACIÓN.

Para realizar la grabación de una EPROM, se necesita enviar la dirección, el dato a grabar y el voltaje necesario Vpp que puede ser de 12.5, 21 o 25 voltios, pero además se requiere de una señal muy importante que es el *Pulso de Grabación*, pues así esté alimentada la memoria adecuadamente con Vpp si no se envía este pulso no quedará grabado el dato enviado. Esta señal consiste de un pulso de voltaje de 50 milisegundos activo en alto, la obtención de esta señal consiste la función de esta sección. Ver anexo 2.

Para obtener el pulso de grabación, se utiliza un timer 555 en configuración monoestable, es decir su salida permanece activa en alto por unos instantes, cuando se envía una señal de disparo, el tiempo que permanece activa depende de una red RC externa; en nuestro caso esta salida estará activa por unos 50 milisegundos, para esto se configura al 555 de la siguiente manera:

Entre los pines 6 y 7 se ubican en serie un capacitor de 1 uF y un potenciómetro de 50K, el cual debe ser ajustado hasta obtener el pulso adecuado. La señal de disparo proviene de U9 que es el demultiplexor, a través de un inversor de U13 de la primera sección, hacia un interruptor electrónico formado por un resistor R1 de 1K y un transistor Q1 NPN de uso común, del colector de este

transistor se conecta a un filtro pasa-altos formado por un capacitor y un resistor, el objetivo de esto es proveer de una señal activa en bajo para el pin 2 del 555 que corresponde al *Trigger* o gatillo que hará posible que la salida genere el pulso de grabación.

Cabe indicar que ésta sección sólo entra en acción cuando existe la función de grabación, por ello se ha dispuesto de una red que detecta el voltaje de grabación Vpp, bajo estas circunstancias cuando se seleccione grabar existirá un voltaje Vpp mínimo de 12.5 V, entonces esta red compuesta por R7 y R8 forman un divisor de voltaje que al ser mayor a 10 Voltios aproximadamente, saturará al transistor Q3 y pondrá en corte al transistor Q4, obteniéndose en el colector de este último un voltaje bajo haciendo descargar al capacitor C4, esta señal se lleva al pin 4 del 555 que sirve para habilitarlo, por tanto cuando haya grabación el timer será habilitado y generará los pulsos, caso contrario no entrará en funcionamiento.

El pulso de grabación se obtiene del pin 3, esta señal se distribuye hacia un LED que indicará al usuario de que se está procediendo con la grabación, también se envía esta señal a la primera sección hacia los pines de la EPROM que corresponden a Output Enable (OE-) y a Program (PGM), éste último vía un inversor de U13.

3.1.1.1.3. SECCIÓN DE ALIMENTACIÓN.

En ésta sección se va a detallar la distribución de la alimentación del Programador y de la conmutación de los voltajes de grabación hacia la EPROM, la cual es comandada por la primera sección. Ver anexo 2

Inicialmente disponemos de un transformador con primario a 120 Vca y secundario de 30 Vca con capacidad de 2 A. El secundario se conecta con un puente rectificador integrado y de aquí se filtra con un capacitor de gran capacidad (2200 uF) C1 y luego se distribuye a dos fuentes independientes,

una para obtener la alimentación de 5 Vcc para los circuitos del Programador y la otra para alimentar a la memoria EPROM y para grabarla.

Para la primera fuente, la salida filtrada es regulada por medio de un Regulador de Voltaje integrado LM317 (REG1), en el pin de ajuste 1 tenemos un potenciómetro multivueltas o de precisión (P1) en serie con un resistor R1, por medio del potenciómetro podemos regular el voltaje en la salida de REG1, obteniéndose así los 5 V los cuales pasan por dos capacitores, C2 y C3 que se acoplan para alimentar todos los circuitos del programador, entre el pin de entrada y salida del regulador se dispone de un diodo para protección. En la salida también se dispone de un interruptor electrónico formado por R2 y un transistor Q1 PNP de propósito general, este es activado por la señal que es enviada por U17 cuya salida activa es bajo, del colector de este transistor se obtiene una señal para activar un LED (LD1) que indica que la EPROM está siendo alimentada y también se dirige hacia un pin del relé cuya salida alimenta el pin de voltaje de grabación de la memoria.

La segunda fuente también emplea un regulador de voltaje LM317 (REG2) cuya conexión es similar de la primera fuente, la diferencia radica en el pin de ajuste o regulación en el cual tenemos tres potenciómetros en paralelo P2, P3 y P4, cada uno de los cuales esta controlado por un transistor (Q2, Q3 y Q4) que actúan como interruptores electrónicos, estos son activados por las señales provenientes de U16 que conmutan el voltaje de grabación de 12.5, 21 y 25 Voltios respectivamente, naturalmente para obtener estos voltajes es necesario ajustar los potenciómetros individualmente, la salida del regulador es acoplada por medio de dos capacitores C5 y C6 y luego son dirigidas a otro pin del relé K1 para proveer de Vpp a la memoria.

Como parte de esta sección se dispone del relé K1 el mismo que es utilizado para conmutar en el pin Vpp de la EPROM del voltaje necesario de grabación esto es 12.5, 21 o 25 Voltios si se escoge esta opción, pero si lo que se escoge es una lectura, este pin debe receptar un voltaje de 5 voltios. Para activar el relé se dispone de otro interruptor electrónico formado por R8 y Q5 que es transistor NPN de uso general, cuya señal proviene del segundo flip flop de U17. Entre los pines de alimentación del

relé se coloca un diodo en contrafase con el objeto de proteger al transistor ya que cuando el relé abre sus contactos produce transitorios de corriente muy alta que podrían destruir al mismo.

3.2. DESCRIPCION DEL SDI - 86S

Como ya se dijo en el capítulo I, el SDI - 86S no es otra cosa que una microcomputadora basada en el 8086, con 16K de ROM y 4K de RAM, su diagrama esquemático se muestra en el anexo 2.

El 8086 posee dos modos de funcionamiento, el modo máximo y el modo mínimo. El modo máximo es utilizado cuando es necesario el uso de otro procesador, como por ejemplo un coprocesador matemático, para lo cual el 8086 debe ser provisto de varios integrados adicionales con el fin de obtener todas las señales necesarias para el control del coprocesador; en cambio en modo mínimo, el 8086 al no tener que controlar a otro procesador, sus señales de control son suficientes para el funcionamiento del sistema. Nosotros hacemos uso del modo mínimo.

3.2.1. LA MEMORIA

La memoria se la debe trabajar en 2 bancos, uno para las localidades pares (banco bajo) y otro para las impares (banco alto). Se utilizan 2 memorias EPROM 2764 y 2 RAM 6116, los buses de dirección y datos son demultiplexados por intermedio de dos registros transparentes de 8 bits 74LS373 para dirección y dos registros bidireccionales para los datos.

3.2.1.1. PROBLEMA PRESENTADO EN EL MANEJO DE LA MEMORIA

Nosotros decidimos ubicar la memoria ROM desde la localidad 0000h hasta la 3FFFh y la RAM desde la 4000h hasta la 4FFFh, debido a que en los primeros 1024K se encuentran los vectores de

interrupción y nosotros necesitabamos el vector 2 para la interrupción del teclado, según como definimos nuestra microcomputadora. Pero, al momento de realizar las pruebas nos dimos cuenta que al encender la computadora ésta apuntaba a la memoria RAM (apunta a la localidad de memoria FFFF0H), lo que nos creaba un problema, ya que al encenderla ésta debía empezar en ROM, por cuanto ahí se debía poner el código del sistema operativo, entonces parecería que teníamos que aumentar 2 EPROMs más a partir de la localidad 5000h, para poder trabajar bien con la interrupción del teclado y que la microcomputadora comience a ejecutar las instrucciones del sistema operativo. Esto hacía pensar que debíamos volver a cablear la parte de la memoria y en otras palabras por problemas de espacio empezar el cableado de nuevo, pero encontramos una solución.

3.2.1.1.1. SOLUCION AL PROBLEMA

Utilizamos un circuito combinatorial basado en un contador 74161 utilizado para carga y puertas adicionales, de tal forma que al apuntar el microprocesador a la localidad FFF0h, este direccione las localidades 0FF0h, apuntando a ROM donde estaría el sistema operativo del SDI - 86S. Luego de 16 localidades más cuando el procesador direccione la localidad 0000h, con lo que continuaría en ROM el problema está solucionado. Una observación adicional fue que cuando el microprocesador lee localidades consecutivas cuyo contenido es FFh, se obtienen resultados inesperados. Por ésta razón, nosotros grabamos las 16 localidades con el valor de 90 correspondiente a instrucciones NOP, desde la localidad 0FF0h hasta la 0FFFh, aunque podrían haber sido ceros o bien código del sistema operativo. Debido a esto, pensamos que es la razón por la que el DOS encera todas las localidades de la memoria RAM al momento de la inicialización, tal vez, esto no sería necesario si Intel hubiera definido FFh, como código para una instrucción NOP en su microprocesador 8086. El circuito es el mostrado en la figura 8.

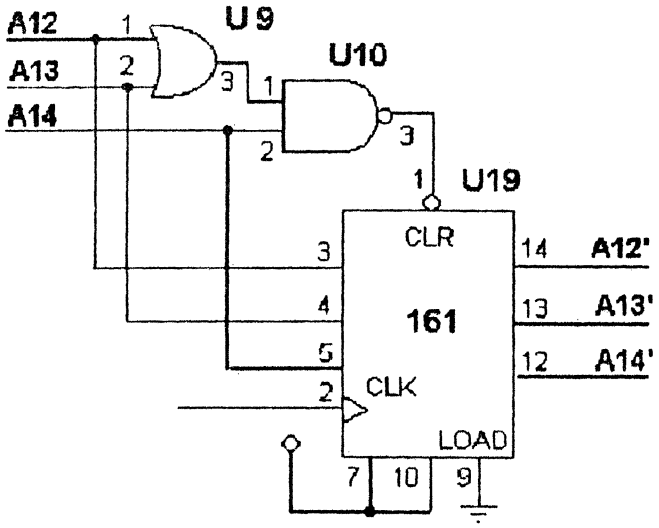


Figura 8: Circuito para cambiar la dirección de apuntamiento de la memoria

3.2.2. GENERACION DE LAS SEÑALES DE CLOCK Y RESET

Para la obtención de estas señales se utiliza el integrado 8284, que posee circuitería interna suficiente para su generación, necesitando solamente unos cuantos componentes discretos para su funcionamiento, para obtener una señal estable de Clock, se utiliza un cristal de 15MHZ, esta señal es divida internamente por el 8284 por un factor de tres, con lo cual tenemos una señal de reloj de 5 MHZ, que es la señal recomendada para el 8086.

3.2.3. COMUNICACIÓN DEL SDI - 86S CON EL EXTERIOR

El SDI - 86S posee un display y teclado alfanúmericos, un puerto de salida de 8 bits y un puerto de entrada de 4 bits, para la comunicación del SDI - 86S con el mundo exterior.

3.2.3.1. DISPLAY LCD HD-44780

El LCD HD-44780 es el display utilizado por el SDI - 86S, el cual posee 2 tipos de memorias RAM, La RAM de datos de visualización (DDRAM), en donde la posición de cada caracter se encuentra asociado a una dirección de ésta memoria y la RAM generadora de caracteres (CGRAM) la cual le permite al usuario definir por programa hasta 8 patrones de caracteres en un formato de 5x7 puntos. También posee una ROM generadora de caracteres (CGROM) en la que se encuentran almacenados 192 caracteres definidos. La ventaja que presenta este módulo LCD, es la de poseer una circuitería de manejo interna, con lo que sólo necesita recibir instrucciones para trabajar. Las principales de este módulo LCD se resumen en la Tabla II.

PRINCIPALES INSTRUCCIONES

RS	R/W	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0	Descripción
0	0	0	0	0	0	0	0	0	1	Borra pantalla, coloca cursor al inicio
0	0	0	0	0	0	0	0	1	X	Cursor 1ra. posición, no afecta DDRAM
0	0	0	0	0	0	0	1	I/D	S	Desplazamiento de cursor y mensaje
0	0	0	0	0	0	1	D	C	B	Activa/Desactiva pantalla, cursor o parpadeo
0	0	0	0	0	1	S/C	R/L	X	X	Mueve cursor y/o desplaza mensaje
0	0	0	0	1	DL	N	F	X	X	Configura el módulo LCD
0	0	0	1	A5	A4	A3	A2	A1	A0	Selecciona posición en CGRAM
0	0	1	A6	A5	A4	A3	A2	A1	A0	Selecciona posición en DDRAM
0	1	BY	A6	A5	A4	A3	A2	A1	A0	Sumunistra estado de BY y contenido de AC
1	0	D7	D6	D5	D4	D3	D2	D1	D0	Escribe en direccion de RAM seleccionada
1	1	D7	D6	D5	D4	D3	D2	D1	D0	Lee dato de dirección de RAM seleccionada

X: Puede ser 0 o 1
I/D: Mueve el cursor a la derecha si es 1, o a la izquierda si es 0
S: Habilita el desplzamiento del mensaje si es 1, o deshabilita si es 0
D: Display activado (D=1), desactivado (D=0)
C: Cursor activado (C=1) o desactivado (C=0)
B: Si es 1, el caracter indicado por el cursor parpadea o no si es 0
S/C: Desplaza el mensaje si es 1 o el cursor si es 0, a la izquierda si (R/L=0) o a la derecha si (R/L=1)
D/L: Interface de 8 bits si es 1, caso contrario de 4 bits
F: Formato de caracteres de 5x10 puntos si es 1, caso contrario de 5x7 puntos

Fuente: Curso básico de Microprocesadores, CEKIT S. A., 1994

Tabla II: Principales instrucciones del display LCD

La DDRAM puede almacenar hasta 80 caracteres, al energizar el display sólo se verá prendido la mitad izquierda (8 caracteres a la izquierda), al configurarlo como dos líneas se logra encender todos los 16 caracteres, en donde las posiciones de 0 a 7 son visibles y las posiciones de 8 a 39 son invisibles para la primera línea, mientras que de la 40 a la 47 son visibles para la segunda línea y de 48 a 80 invisibles. Esto es importante tomar en cuenta al momento de la programación del display.

Este módulo LCD tiene 2 registros seleccionables: El registro de instruciones (IR), en él se almacenan las instrucciones o las direcciones de las dos memorias RAM (DDRAM o CGRAM) a las que se desee acceder, éste no puede ser leído por el microprocesador. El registro de datos (DR), en el cual se almacenan o reciben los datos que se envían entre el microprocesador y la DDRAM o CGRAM. Estos registros pueden ser seleccionados mediante la línea selectora RS (Register Select) y son los únicos que pueden ser controlados directamente por el microprocesador. El LCD HD-44780 utiliza 14 líneas para su funcionamiento, 2 para alimentación, 1 para un voltaje de referencia que maneja el contraste de la pantalla, 3 líneas de control y 8 líneas para datos o instrucciones, estas se detallan en la Tabla III.

NUMERO DE PIN	DESCRIPCION
1	GND
2	VDD
3	Vo
4	RS
5	R/W
6	E
7	DB0
8	DB1
9	DB2
10	DB3
11	DB4
12	DB5
13	DB6
14	DB7

Tabla III: Descripción de los pines del display LCD

3.2.3.1.1. COMUNICACIÓN ENTRE EL LCD HD-44780 Y EL MICROPROCESADOR

La comunicación del LCD HD-44780 con el microprocesador 8086 en el SDI - 86S, se realiza a través del 8255 en modo 0, configurando el puerto A para salida y conectándo sus líneas con los pines desde DB0 (pin7) a DB7 (pin 14) del LCD, por aquí el microprocesador podrá enviar los caracteres a mostrar en pantalla o las instrucciones que debe realizar el display. Adicionalmente, conectándo las líneas del puerto C alto (desde PC4 a PC7) configurado como salida, a las líneas E, R/W y RS, respectivamente. Estas últimas líneas permiten la comunicación de las señales de control que enviará el microprocesador al display, tales como para especificar si el dato envíado es un caracter que debe ser mostrado en pantalla o una instrucción que debe ser ejecutada por el display, por ella también se debe envíar una señal de retardo que sustituya la necesidad de monitorear constantemente la bandera BY (bandera que indica si el display está ocupado), antes de volver a presentar un nuevo dato en pantalla. La siguiente figura (9) muestra la relación entre RS, R/W y E:

RELACION ENTRE RS, R / $\overline{W}$ y E

RS	R / $\overline{W}$	E	Operación
0	0		Escribir en el IR
0	1		Leer BY y AC (*)
1	0		Escribir en el DR
1	1		Leer del DR

IR: Registro de instrucciones
DR: Registro de datos
BY: Bandera de "ocupado"
AC: Contador de direcciones
*: BY→DB7, (AC)→(DB6~DB0)

Fuente: Curso básico de Microprocesadores, CEKIT S. A., 1994

Figura 9: Relación entre RS, R/W y E del dispaly LCD

3.2.3.2. EL TECLADO

Se utiliza uno del tipo matricial sensible al toque de 48 teclas extraplano y sin partes móviles, posee 26 teclas alfabéticas, 10 numéricas y 12 de signo y de control, sus salidas no producen una señal de tipo ASCII, por lo cual fue necesario incorporar una rutina de conversión, ésta matriz está direccionada por dos grupos de líneas, una de 6 y otra de 8, con lo cual se forman las 48 teclas, como particularidad éste teclado no corresponde al clásico QWERTY, pero su distribución de teclas no produce incomodidad de operación.

3.2.3.2.1. COMUNICACIÓN DEL TECLADO CON EL MICROPROCESADOR

El teclado se comunica con el microprocesador 8086 en el SDI - 86S mediante el controlador visual y de teclado 8279, configurado en modo 0 (teclado codificado con la activación de 2 teclas) y operando a una velocidad interna de 100 KHZ que es programada mediante la palabra de control con un valor de 50, ya que se usa una señal de reloj de 5MHZ. Para obtener las señales del teclado se utiliza un decodificador 74LS138 cuyas entradas de selección provienen de las líneas de rastreo del 8279, estas son: SL0, SL1 y SL2, de las ocho salidas del 138 solamente se utilizan 6, que van al grupo de 6 líneas del teclado, mientras que el grupo de 8 líneas son controladas por las señales RL0 a RL7 del 8279.

3.2.3.3. PUERTOS

El SDI - 86S posee un puerto de salida de 8 bits, que corresponde al puerto B del 8255 y un puerto de entrada de 4 bits, que corresponde al puerto C bajo. En la configuración del 8255 se estableció en modo 0 además los puertos A y C alto como salidas para el display y el teclado respectivamente, esta combinación es programada mediante la palabra de control con un valor de 81H.

Estos dos puertos B y C bajo están implementados físicamente en el SDI - 86S por un conector DB25 hembra, siguiendo la distribución de señales del puerto paralelo, pero obviamente si el grupo de señales de control que éste posee.

3.2.4. MAPA DE MEMORIA DEL SISTEMA OPERATIVO DEL SDI - 86S

El mapa de memoria del SDI - 86S, es el que se muestra en la figura 10.

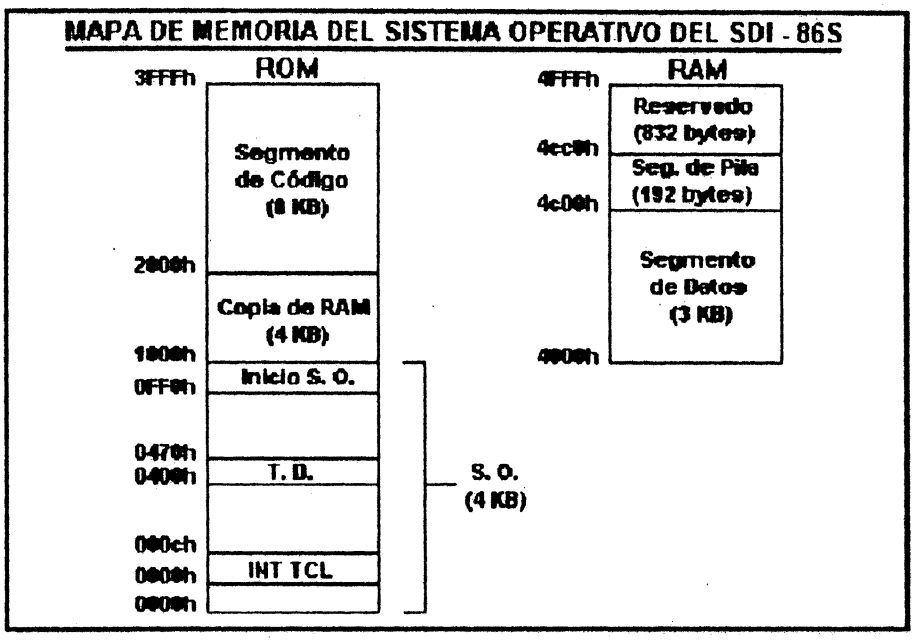


Figura 10: Mapa de memoria del S.O. del SDI-86

Como se puede notar, el sistema operativo abarca los primeros 4 KB de la memoria ROM, el inicio del sistema operativo empieza en la localidad 0FF0h, continuando luego en la localidad 0000h después de la localidad 0FFFh. En el sistema operativo se hace un salto de 4 localidades desde la 0008h a la 000Ch, debido a que en éstas se encuentra el vector de interrupción del teclado. La rutina de servicio del teclado se encuentra en la localidad 0034h, de tal forma que en el vector de interrupción se tiene definido el segmento 0000h y el desplazamiento 0034h. La tabla de datos del

display está ubicada desde la localidad 0400h hasta la 0470h. De los 4 KB destinados para el sistema operativo solamente se utilizan 778 bytes, quedando el resto para expansión del mismo. Existe una copia de ROM en RAM de 4 KB correspondientes al segmento de datos, pila y un área reservada de 832 bytes para mejoras futuras. En estos momentos, solamente se realiza una copia del segmento de datos de la ROM a la RAM, pero más adelante se puede utilizar la parte de RAM reservada y será necesario también moverla desde la ROM, en el próximo capítulo se explicará en detalle en que se puede utilizar esta área reservada. El segmento de código abarca desde la localidad 2000h a la 3FFFh, mientras que el segmento de datos desde la 4C00h a la 4CC0, quedando reservado los 832 bytes últimos.

3.2.5. SISTEMA OPERATIVO DEL SDI - 86S

Los diagramas de flujo junto con el código fuente del sistema operativo del SDI – 86S se encuentran disponibles en el anexo 3.

La subrutina retardo se utiliza para no necesitar averiguar por la bandera BY para saber si el display se encuentra ocupado o no, también sirve para permitir que el mensaje sea visualizado en la pantalla a una velocidad adecuada. La subrutina *mcar* se utiliza para presentar un dato en la pantalla del display, mientras que con la subrutina *palintr* se puede enviar una palabra de instrucción para que sea ejecutada luego por el módulo LCD HD-44780. La subrutina *lntcl* se encarga de obtener el caracter tecleado y convertirlo a código ASCII para que pueda ser interpretado por el display. La subrutina fd0 realiza la configuración del display y borra la pantalla. La subrutina *ftl* espera hasta que se teclee un caracter.

El programa principal define los segmentos extra y datos en la localidad 0000h y pila desde la 4C00h a la 4CC0h. Se define el segmento extra y datos en la posición 0000h porque en un inicio sólo trabajabamos con el segmento de datos para presentar los mensajes del sistema operativo, el cual teníamos luego que relocalizar a la localidad 4000h donde realmente debe empezar, sin embargo

hubo la necesidad de utilizar los primeros 5 datos de la tabla del display para generar la función de borrar pantalla, al haber relocalizado el segmento de datos la función de borrar pantalla no funcionaría, porque la tabla del display ya no forma parte del segmento de datos, a menos que se direccione el segmento extra con el cual sí podemos acceder a la tabla del display. Luego el programa principal, configura el 8279, 8255 y el display, seguidamente presenta el mensaje1 (SDI - 86 ver. 1.0) después de que todo esto ha sucedido, aparece el prompt (-) en el mensaje, en éste momento el sistema operativo llama a la suburtina *ft1* y espera a que el usuario presione la tecla 'S', al suceder esto llama a la subrutina *fd0* para borrar la pantalla y mostrar el prompt (-). En este instante, el sistema operativo puede receptar los comandos denominados "caracteres del sistema operativo". Al recibir un caracter, el programa principal compara si es 'Do', 'H' o 'Q'. En caso de ser 'Do', muestra el mensaje2 (Copiando RAM), luego copia la RAM desde la ROM, después presenta el mensaje3 (Programa), relocalizando luego el segmento de datos en localidad 4000h y ejecutando el programa, al terminar su ejecución regresa a recibir otro comando, relocalizando nuevamente el segmento de datos en 0000h. Si se recibe el carácter 'H', se presenta la ayuda de los caracteres del sistema operativo en pantalla, luego de cada línea de ayuda espera que el usuario presione el caracter "flecha a la derecha" que permite avanzar la siguiente línea. Si se recibe el caracter 'Q' el sistema operativo finaliza, presentando el mensaje "FIN", hasta que se presione el botón "reset" o se apague el SDI – 86S.

3.2.6. PRUEBAS AL SISTEMA

La aplicación que nosotros hemos construído se trata de un juego doble, el primero es un juego de palabras utilizado para probar el display y el teclado, mientras que el segundo es un juego de luces programable por el usuario, que sirve para probar el puerto de entrada y el de salida. Contiene un menú que permitirá al usuario seleccionar el juego 1 o 2, al inicio se presentarán las opciones y para continuar se debe presionar la letra "C". En el momento que sale el mensaje "ELIJA OPCION:", el usuario puede presionar "1" si desea el juego de palabras y "2" si desea el juego de luces

programables que continuará indefinidamente hasta que se ingrese la combinación binaria del "7" por el puerto de entrada. Cuando el juego de palabras termina se presentará el mensaje "Otro jugador", mientras que al finalizar el juego de luces se verá el mensaje "Fin de secuencia". Cada que termina uno de los dos juegos el usuario puede presionar "1" o "2" si desea continuar o "S" si desea salir.

El código fuente de la aplicación se encuentra en el anexo 4, consta de un programa principal y 3 subrutinas. La subrutina "premen" se utiliza para presentar un mensaje, la cual es llamada varias veces por el programa principal para mostrar todos los mensajes de la aplicación. La subrutina "leercar" se utiliza para el juego de palabras, ésta se encarga de permitir la lectura de un caracter desde el teclado y visualizarlo en la primera posición y la subrutina "retcomp" es un retardo utilizado para presentar el cambio en las luces a través del puerto de salida del SDI - 86S, que corresponde al puerto B del 8255.

3.2.7. SIMULACION EN EL SDI - 86P

Para la simulación de un archivo se deben seguir ciertas reglas fáciles de entender, se tiene 1 función para el manejo del teclado y 2 para el manejo del display. Estas son las siguientes:

- *tclfnc0: Se utiliza para capturar una tecla desde el teclado*
- *dspfnc0: Se utiliza para borrar la pantalla*
- *dspfnc1: se utiliza para imprimir un caracter en la posición especificada en col.*

Estas subrutinas deben ser llamadas en el programa del usuario, tomado en cuenta el funcionamiento que se ha descrito, no se debe escribir el código para estas subrutinas en el programa, ya que el SDI - 86P se encargará de anexar estas subrutinas a él, con el código necesario para la simulación del funcionamiento descrito. Esto hará que se pueda ver el funcionamiento del SDI - 86S, tal como se lo vería si se lo implementara en la realidad.

3.2.8. CREACION DE UN ARCHIVO COMPATIBLE PARA EL SDI - 86S

Al crear un archivo compatible con el SDI - 86S. El SDI - 86P, reemplazará las subrutinas de los códigos de simulación con los necesarios para el funcionamiento real del SDI - 86S, manteniendo los nombres anteriores y logrando de ésta manera la compatibilización entre lo simulado y lo real.

3.2.8.1. PROBLEMA PRESENTADO EN LA COMPATIBILIZACION DE UN ARCHIVO

Inicialmente pensamos simular todas las instrucciones del 8086, para poder ver el cambio de registros y banderas paso a paso. Además, crear nuestra propia forma de obtener el código máquina de un programa, pero leímos en la página 75 del libro LOS MICROPROCESADORES INTEL de Barry B. Brey que "hay más de 20.000 variantes en el lenguaje de máquina para los microprocesadores 8086-80486", lo que hace que simularlas todas resulte demasiado complicado. Esto nos llevo a pensar en otra solución, la cual la adoptamos, ésta se trataba de utilizar el Turbo Ensamblador de Borland para la generación del código máquina y en cuanto a la solución para poder el ver el cambio de registros y banderas paso a paso, implementar un programa escrito en lenguaje ensamblador que se añada al programa que se quiere depurar, haciéndo que éste sea programado, añadido y ejecutado de una manera automática por el SDI - 86P, la cual finalmente logramos. Sin embargo, pensamos que si se implementaba una microcomputadora y se grababa el código máquina generado por el Turbo Ensamblador en las EPROMs conectadas a ella, el programa almacenado funcionaría sin problemas, lo cual no fue cierto. Resulta que cuando se tiene en un registro de 16 bits un dato como 4A5Bh por ejemplo, el byte menos significativo, en este caso 5Bh se almacena primero y el byte más significativo, en este caso 4Ah se almacena después. Es decir se invierte el orden en los bytes, cada que sucede una transferencia de un registro de 16 bits a memoria, ocurriendo el proceso inverso en la transferencia de memoria a registro de 16 bits. Si se observa un archivo *.lst generado por el Turbo Ensamblador, se puede notar que no se toma en cuenta esto, al menos en el

listado no se observa esta inversión de bytes, ya que ella debe realizarse cuando el archivo hace la transferencia a memoria, lo que presentó el incoveniente de invertir manualmente cada vez que se encontraba con una instrucción de este tipo como "mov cx,52", "call game1", "jmp ComSO" (salto largo), "inc col", etc. Esto significa, que si vamos a crear una aplicación con el 8086 y construímos una microcomputadora para realizarla, debemos de tomar en cuenta al momento de grabar el código máquina de un archivo *.lst en las EPROMs , estas consideraciones, caso contrario la aplicación no funcionará. Pero nosotros ya solucionamos este problema en nuestra tesis.

3.2.8.1.1 SOLUCION AL PROBLEMA

El realizar estas inversiones manualmente hace que haya mucha posibilidad de equivocarse, a más de lo dificultoso que resulta ingresar el código en lenguaje máquina y comprobarlo para ver si no se ha equivocado. Sin embargo, si nuestra tesis va a ingresar el código máquina automáticamente a las EPROMs, a diferencia del SDK-86, un sistema de desarrollo sacado por Intel hace algunos años atrás. Entonces, porque no hacer que estas inversiones se realicen automáticamente por la computadora, por lo que nosotros al código que obtenemos del archivo *.lst generado por el Turbo Ensamblador, le hacemos un pequeño tratamiento. Buscamos por los códigos de las instrucciones que requieren estas inversiones, e invertimos los dos bytes subsiguientes, lo mismo que se debe hacer cuando se encuentra con dos bytes que representan un dato de desplazamiento de una variable definida en el segmento de datos. Cabe destacar, que previamente a esto se debe separar el código máquina del texto y separar los bytes en un arreglo llamado DataSegData ()

3.2.9. TRATAMIENTO DE UN ARCHIVO *.LST POR EL SDI - 86P

Primeramente hay que señalar que debido a que el sistema hace uso del Ensamblador de la Borland para el proceso de compilación y enlace, tenemos que se crearán 4 diferentes tipos de archivos con

las extensiones de archivo ASM, OBJ, LST, MAP y EXE. De todos ellos el más importante para el proceso que pretendemos realizar es el archivo de extensión LST. En el mismo tendremos que se presenta una combinación de código máquina y código fuente y del cual haremos el proceso de separación de ellos para ubicarlos en las respectivas celdas del control personalizado GRID.VBX. Al analizar este archico se observó que tanto el código máquina y el código fuente se diferenciaban por la presencia de un caracter de tabulación (código ASCII 9), entonces se procedió a leer el archivo de extensión LST línea a línea, lo cual se conseguía por medio del siguiente código:

```
Open OnlyName + ".LST" For Input As #1
Do
  n = n + 1
  Line Input #1, LineTemp
Loop Until EOF(1)
  Close #1
```

Donde OnlyName contiene el nombre del archivo a procesar y el lazo Do - Loop Until EOF(1) permite leer al archivo desde el inicio al final (en el cual encontraremos el caracter especial EOF (End Of File) y Line Input #1, LineTemp permitirá leer una línea la cual está delimitada por el caracter retorno de carro. Teniendo en cuenta que una línea completa esta almacenada en la variable LineTemp. Por medio de la instrucción CodeMach$ = Mid$(LineTemp, temp1, temp3 - temp1 + 1) capturaremos la parte que involucra el código máquina, donde temp1 = InStr(1, LineTemp, TL) + 1, temp2 = InStr(temp1, LineTemp, TL) y temp3 = InStr(temp1, LineTemp, "+"). La sentencia ProgSrc$ = Trim$(Mid$(LineTemp, temp3 + 1, Len(LineTemp) - temp3 + 1)) captura la parte relativa al código fuente. Una vez hecho esto asignaremos a las celdas respectivas por medio de frmCode!GridCode.Col = 1, frmCode!GridCode.Row = i y frmCode!GridCode.Text = CodeMach$ para el caso del código máquina y frmCode!GridCode.Col = 2, frmCode!GridCode.Row = i y frmCode!GridCode.Text = ProgSrc$ para el código fuente; frmCode.Show mostrará la forma que contiene el contro GRID.VBX y sus valores correspondientemente asignados.

Luego del proceso anterior nos toca separar en celdas individuales el código máquina para posteriormente guardarlo en una memoria EPROM. Teniendo en cuenta que en las celdas direccionadas por frmCode!GridCode.Col = 1 y frmCode!GridCode.Row = i se encuentra el mencionado código, es sobre este que debemos actuar. Adicionalmente las sentencias del ensamblador DATA SEGMENT - DATA ENDS y CODE SEGMENT - CODE ENDS nos indicarán el inicio y el fin de los bloques del segmento de DATOS y CODIGO, entonces por medio de:

```
For i = 1 To n
    frmCode!GridCode.Row = i
    If UCase$(Mid$(frmCode!GridCode.Text, 1, Len(DatName))) = DatName Then inicio = i: Exit
For
Next i
For i = inicio + 1 To n
   frmCode!GridCode.Row = i
   If UCase$(Mid$(frmCode!GridCode.Text, 1, Len(DatName))) = DatName Then fin = i: Exit For
Next i
```

Obtendremos el incio y el final de los mencionados bloques. Luego el proceso de separación es relativamete fácil ya que los datos o código se encuetran separados por cadenas de espacios en blanco y por medio del siguiente código:

```
Pos = InStr(frmCode!GridCode.Text, " ")
While Pos <> 0
      indice = indice + 1
      DataSeg Data(indice) = Mid$(frmCode!GridCode.Text, 1, Pos - 1)
      frmCode!GridCode.Text = Mid$(frmCode!GridCode.Text, Pos + 1,
Len(frmCode!GridCode.Text))
      Pos = InStr(frmCode!GridCode.Text, " ")
```

Wend

If Len(frmCode!GridCode.Text) <> 0 Then indice = indice + 1: DataSeg Data(indice) =

Mid$(frmCode!GridCode.Text, 1, Len(frmCode!GridCode.Text))

Habremos separado el código máquina. Hay ciertas singularidades que se presentan en el proceso anteriormente mencionado y que el algoritmo se encarga de procesarlas por lo que se recomienda analizar el código fuente escrito en Visual Basic para mayor detalle. Como último detalle señalaremos que hay ciertos casos en que se hace necasario invertir los bytes antes de proceder a grabarlos y se describen por:

If Mid$(frmCode!GridCode.Text, 1, 2) = "B8" Or Mid$(frmCode!GridCode.Text, 1, 2) = "B9" Or

Mid$(frmCode!GridCode.Text, 1, 2) = "BA" Or Mid$(frmCode!GridCode.Text, 1, 2) = "BB" Or

Mid$(frmCode!GridCode.Text, 1, 2) = "BC" Or Mid$(frmCode!GridCode.Text, 1, 2) = "BD" Or

Mid$(frmCode!GridCode.Text, 1, 2) = "BE" Or Mid$(frmCode!GridCode.Text, 1, 2) = "BF" Or

Mid$(frmCode!GridCode.Text, 1, 2) = "E8" Or Mid$(frmCode!GridCode.Text, 1, 2) = "E9" Then

Finalmente, después de estos procedimientos tendremos dos arreglos: ROM y RAM, el primero de los cuales contiene el segmento de código del programa del usuario, y el segundo el segmento de datos de dicho programa.

3.2.10. SUBRUTINAS ANEXADAS A UNA APLICACIÓN

Las subrutinas anexadas a una aplicación deben realizar lo mismo que se ha simulado en el SDI – 86P, se puede observar en el código de la aplicación que se encuentra en el anexo 6, que la subrutina tclfnc0 permite el ingreso de un caracter desde el teclado, para esto encera el registro AL y espera hasta que tome un valor, lo cual puede ocurrir únicamente al presionar una tecla. Debido a que la subrutina del sistema operativo Intcl se encarga de capturar la tecla presionada y convertir el valor

recibido a ASCII, está garantizado que el valor al que cambió el registro AL es el valor ASCII de la tecla presionada. La subrutina dspfnc0 realiza el borrado de pantalla cada vez que es llamada, para ello guarda los datos temporalmente de los registros BX, CX, DX, SI y DI para que no sean alterados, lee los 5 primeros datos de la tabla del display, llamando luego a la subrutina palintr que se encarga de enviar las instrucciones al display cada vez que es llamada, finalmente se devuelven los datos temporalmente almacenados a los registros BX, CX, DX, SI y DI. La subrutina dspfnc1 reliza la impresión de un caracter en la posición especificada en col, primero guarda temporalmente los datos de los registros BX, CX, DX, SI y DI, en variables definidas en el segmento de datos, para que no sean alterados luego de realizada esta función, luego coloca el cursor en la primera posición y cuenta las columnas hasta la posición deseada en donde se presentará el carácter, previamente debe hacer el cambio respectivo para que un caracter desde la posición 7, no caiga en un rango invisible de pantalla; finalmente, presenta el carácter en la posición especificada en col y devuelve los datos temporalmente almacenados a los registros BX, CX, DX, SI y DI.

3.2.11. IMPLEMENTACION DE LA APLICACIÓN EN EL SDI - 86S

Luego de creado el archivo compatible, guárdelo con el nombre que desee y elija la opción Ver Código Máquina del menú Herramientas, después elija la opción Grabar Código Máquina del mismo menú, junto con la opción de la memoria EPROM que va ha grabar (par o impar) y el voltaje de grabación adecuado (12.5, 21 o 25 V). Debe grabar las 2 EPROMs, conectándo previamente a la elección de las opciones el SDI - 86G al SDI - 86P, mediante el puerto paralelo. El SDI - 86P grabará el sistema operativo, el segmento de datos y de código de la aplicación en ambas EPROMs, luego de esto, debe colocar las dos EPROMs en sus sockets correspondientes en el SDI - 86S y la aplicación deberá funcionar como en la simulación.

CAPITULO IV

CONCLUSIONES Y RECOMENDACIONES

En este capítulo se abarcará el desarrollo de los circuitos impresos de los componentes de hardware del SDI – 86, así como las mejoras que se pueden realizar para mejorar diversos aspectos del SDI – 86, entre las que incluímos mejoras al entorno de depuración del SDI – 86P, a la simulación del SDI – 86S, al sistema operativo del SDI – 86S, entre otras. Finalmente exponiendo las conclusiones y observaciones realizadas en nuestro trabajo.

4.1. SOBRE LOS CIRCUITOS IMPRESOS

Para el desarrollo de los circuitos impresos del SDI - 86G y del SDI - 86S (ver anexo 4), se utilizó el programa Tango PCB de Accel Technologies, el cual permite crear y editar PCBs (Printed Circuit Board), éste es un programa para ambiente DOS, por lo tanto no permite realizar con una mayor fluidez el desarrollo de un circuito, además la versión utilizada corresponde al año de 1989, por tanto es una versión anticuada, pero que cumple con las necesidades mínimas requeridas para la elaboración de circuitos, cabe recalcar que se hizo uso de la opción de la creación de los circuitos en forma automática, y no manual como se había difundido este programa en seminarios internos de la facultad, lo que sí se hizo fue una edición obviamente manual de las pistas que el programa no pudo llevar a cabo, pues en el diseño fue necesario realizarlo a doble cara debido a la complejidad del circuito.

Como observación final sobre este tópico hay que indicar que la fabricación de un circuito impreso a doble cara en nuestro medio es decepcionante, pues los lugares en donde los fabrican son empíricos y no permiten realizar estos con todas las características que debieran poseer, es decir las *pistas* no pueden ser delgadas y lo mismo con respecto a las *vías*, que son agujeros que conectan la pista de una cara con otra no son metalizados, como consecuencia hay que editar el circuito impreso de tal forma que sus dimensiones no son eficientes, pero que puede servir para elaborarlo artesanalmente.

4.2. MEJORAS AL SDI-86P

Entre las mejoras que se pueden hacer al SDI-86P tenemos una mejora a la presentación del entorno de depuración, ésta se puede realizar utilizando las funciones de video en el programa de depuración 8086dep.asm. También se debe ampliar para que la depuración sea no solo para el 8086, sino también para los demás procesadores de esta familia, como el 80286, 80386, 80486, etc. Esto se consigue ampliando el programa 8086dep.asm. Además, se puede mejorar el chequeo de sistaxis para que la depuración se realice bajo el entorno de Windows en lo posible.

Una buena alternativa para no tener que usar las funciones de video en el programa de depuración y necesitar programarlas en ensamblador sería codificando el SDI - 86P en Visual C++ de Microsoft, en lugar de Visual Basic del mismo fabricante. La razón es que la comunicación entre Visual C++ y ensamblador sería más sencilla de lo que fue al usar Visual Basic, por lo que se podría programar un entorno visual muy agradable en lenguaje C, dejando la depuración y el programa del usuario en ensamblador. El principal limitante para no haber podido crear un entorno visual más agradable en la depuración fue porque en Basic no se puede tener acceso a programar en ensamblador, cosa que si se puede hacer en lenguaje C, lo que elimina este problema.

En cuanto a la simulación del SDI - 86S que realiza el SDI - 86P, esta puede realizarse sacando las entradas y salidas virtuales que simulan el funcionamiento del SDI - 86S por el puerto paralelo y realizar una simulación que permita que el usuario pueda construír circuitos electrónicos que se

conecten al SDI - 86P y que funcionen de la misma manera como si se lo conectara al SDI - 86S, sin la necesidad de tenerlo en su casa, haciendo un poco más flexible el uso del SDI - 86. Para lograr esto se debe crear una pequeña interface entre los circuitos electrónicos que van a ser controlados y el PC.

4.3. MEJORAS AL SDI-86G

Entre las mejoras que se pueden aplicar a éste módulo, tenemos que podría modificarse el diseño de la fuente de alimentación, ya que la presente tiene el inconveniente de disipar bastante calor en el regulador integrado, aún cuando se empleó un disipador de dimensiones grandes, pero este inconveniente no afecta la estabilidad del voltaje.

También es recomendable el empleo de un socket tipo ZIF (Zero Insertion Force), para la memoria EPROM, nosotros no lo ubicamos porque no pudimos conseguirlo en el mercado local. Este socket es importante para no estropear la memoria tanto en la inserción como en el retiro.

4.4. MEJORAS AL SDI-86S

Se puede sacar en el circuito impreso del SDI-86S las líneas de dirección, datos y control del microprocesador a través de un conector, esto sirve para facilitar ampliaciones futuras que se quieran realizar en él o en el caso que se desee ampliar los puertos de entrada y salida, poder conectar fácilmente otros chips 8255 los cuales deben ser decodificados y configurados en el sistema operativo o el programa de aplicación. Nosotros no lo pusimos porque en el Tango de Accel Technologies, no encontramos un conector adecuado para esto, el conector recomendable puede ser un bus ISA de 16 bits. Al igual que en el SDI - 86G, es recomendable la ubicación de dos sockets tipo ZIF para las memorias, con el fin de no estropearlas.

4.5. MEJORAS AL SISTEMA OPERATIVO DEL SDI - 86S

El sistema operativo del SDI - 86S maneja una función para el teclado, ésta es suficiente para la captura de teclas pulsadas una por una, las cuales pueden ser almacenadas en una parte de la memoria antes de su tratamiento. Sin embargo, si se desea se puede añadir otra para capturar un grupo de caracteres en un búfer destinado para el teclado. Las funciones del display incorporadas son dos: una para borrar la pantalla y otra para mostrar un caracter en la columna especificada en la variable *col*, éstas son suficientes para el manejo del display, pero se puede anexar otra para realizar desplazamientos de caracteres lo cual también se puede hacer con las dos anteriores, pero con ellas no se aprovecha la instrucción incorporada en el módulo LCD que realiza ésto automáticamente, además se puede permitir que el usuario disponga de funciones para manejar directamente las instrucciones internas del display, lo que no se hizo porque cada función incorporada debe ser simulada en lenguaje ensamblador y por problema de tiempo no fue realizado. Esto sería muy sencillo si se codificara el SDI - 86P en *Visual C++* de Microsoft en reemplazo del *Visual Basic* del mismo fabricante, permitiendo incluso obtener una presentación más agradable.

4.6. OBSERVACIONES

Se ha observado que si se quiere implementar un sistema digital basado en un microprocesador 8086, se debe construir una microcomputadora basada en éste procesador, realizar el código fuente tanto del sistema operativo como de la aplicación, debiendo someterlos al lenguaje ensamblador para la obtención del código máquina. Este código debe ser grabado en 2 memorias EPROMs una para el banco alto (localidades impares) y otra para el banco bajo (localidades pares). Pero se debe tomar en cuenta que si se usa el Turbo Ensamblador de Borland (tal vez en los otros también ocurre lo mismo), hay que tomar en cuenta que el código máquina que se encuentra en el archivo *.lst, *no debe copiarse a las EPROMs tal como se lo ve*, sino que hay que tomar en cuenta que deben invertirse los 2 bytes consecutivos a un código que realice una transferencia de una palabra de la memoria a un

registro de 16 bits, o viceversa. Así como, cuando se trabaja con variables definidas en el segmento de datos. Debe tomarse en cuenta que el código máquina puede ubicarse en cualquier lugar en la memoria y este funcionará igual, gracias a que los microprocesadores de Intel, poseen la característica de la *relocalización de memoria*. Además, para comunicar el sistema operativo con el programa se deben realizar cálculos manuales, tomando en cuenta el *funcionamiento de salto* que se emplee.

Según lo que se ha visto, la relocalización de memoria se puede realizar debido a que los saltos ocurridos por una instrucción de salto o una llamada a subrutina, no son a una dirección específica, si no más bien relativa, siendo saltos a una localidad *n* veces más o menos la localidad de la próxima instrucción, donde *n* depende si el salto es corto, cercano o largo. Además, la definición de los registros de segmentos y desplazamientos hacen que para relocalizar un segmento de memoria resulte tan fácil como cambiar el contenido del registro del segmento que se desea, con el valor que se quiere. Hay que tomar en cuenta que los registros de segmentos no pueden ser cambiados directamente, sino que se lo debe hacer a través de otro registro. Es importante notar que como el registro de código (CS) no puede ser destino, la pregunta es cómo se puede relocalizar el segmento de código, tal vez la respuesta sea con interrupciones de software.

Se ha notado que cuando el procesador lee consecutivamente localidades de memoria cuyo contenido es FFh se obtienen resultados inesperados, por eso es recomendable que en las localidades no usadas se almacene el contenido de 90h o 00h, ésta debe ser la razón por la que el D.O.S. de Microsoft encera las localidades en la inicialización. Nuestro sistema operativo no accede a localidades no ocupadas, por lo que no fue necesario encerarlas.

Al momento de crear el archivo de depuración 8086dep.asm nos dimos cuenta que no se puede llamar a una macro *dentro de un lazo de programación*, cuando se trabaja con código nativo del 8086. Esto se debe a que las instrucciones de salto *Jnnn* alcanzan a una dirección corta por medio de un desplazamiento de un byte que limita a una distancia de –128 a 127 bytes. Sólo las instrucciones

JMP y CALL pueden lograr un salto cercano en el mismo segmento utilizando 2 bytes para esto, logrando un salto de -32768 a 32767 o un lejano a otro segmento. Por esto es que al llamar una macro desde una subrutina no hay ningún problema, incluso si se usa el código nativo del 8086. Cabe destacar que para microprocesadores 80386 en adelante, las instrucciones de salto sí pueden realizar saltos cercanos, en estos sí se puede llamar macros dentro un lazo de programación creado con ellos, pero hasta un cierto límite. Algo importante de notar es que la instrucción *LOOP* no puede realizar saltos cercanos, peor lejanos; razón por la que no se pueden llamar macros dentro de lazos formados con esta instrucción. Esto se debe tomar en cuenta al momento de realizar una aplicación con el SDI-86 ya que "no debe usar esta instrucción cuando depure un programa paso a paso", pero esto no es problema porque con la instrucción *jne* y decrementando un valor puede lograr lo mismo que realiza la instrucción LOOP y con un mayor alcance.

4.7. CONCLUSIONES

Fue difícil la realización de ésta tesis, por cuanto no se contó con mucha información al respecto de ciertos aspectos importantes en su desarrollo, hemos abarcado dos campos importantes en nuestra formación profesional, el manejo de circuitos electrónicos y la programación. Hemos logrado una interacción eficiente entre el hardware y el software, que nos permita realizar lo que nos propusimos. Cabe destacar que las ideas básicas las tomamos de nuestra bibliografía, pero el desarrollo de la tesis involucró mucha imaginación y creatividad para resolver los problemas que encontramos en el camino, habiendo encontrado respuestas a ellos junto con el descubrimiento de conocimientos que no teníamos documentados en la bibliografía o que no habíamos alcanzado a revisar. Hemos experimentado cómo la programación hace que los circuitos electrónicos conectados al microprocesador funcionen de la manera esperada, la importancia de un sistema operativo para lograr que el hardware quede listo para la ejecución de una aplicación.

El sistema de desarrollo realizado es diferente a los que conocíamos, nació de observar que para poder crear una aplicación en una computadora se la debe programar, guardando éste programa en un

dispositivo de almacenamiento secundario como un diskette, pudiéndo ejecutar ésta aplicación introduciendo el diskette en la computadora e invocando al sistema operativo (D.O.S.) para que nos permita ejecutar el programa almacenado en él. Como sabemos, el programa es cargado en RAM y luego ejecutado. Por esta razón, pensamos que así como el D.O.S. mueve un programa de un diskette a la RAM, nosotros también podríamos mover un programa pero ésta vez de ROM a RAM, ya que el objetivo de almacenar un programa en ROM es no sólo la de no tener que volver a escribir el programa de la aplicación, sino también de no dejar que el usuario tenga la molestia de ingresar el código de su aplicación manualmente, como lo hacían algunos sistemas de desarrollo como el SDK-86 fabricado por Intel hace algunos años. Para realizar esto, debímos diseñar un grabador de EPROM (SDI – 86G) que se conecte al SDI – 86P para grabar el programa del usuario, junto con el sistema operativo que se encargaría de dejar listo el programa del usuario para su ejecución en el sistema de aplicaciones (SDI – 86S).

ANEXOS

ANEXO I
ARCHIVOS GENERADOS POR EL SDI - 86P

Archivo 8086dep.asm

```
CURSOR MACRO F,C    ;Macro para posicionar el cursor en la fila o columna especificada.
        MOV AH,02H
        MOV BH,00H
        MOV DH,F
        MOV DL,C
        INT 10H
ENDM
CLRSCR MACRO        ;Macro para limpiar pantalla.
        MOV AX,0600H
        MOV BH,07H
        MOV CX,0000H
        MOV DX,184FH
        INT 10H
ENDM
VEREG  MACRO ET,R  ;Macro para presentar el contenido del registro especificado.
MOV AH, 9
MOV DX,OFFSET ET
INT 21H
MOV AX, R
CALL DESP
ENDM
;Macro para depurar la linea de programa especificada.
DEPURAR MACRO
C1,C2,C3,C4,C5,C6,C7,C8,C9,C10,C11,C12,C13,C14,C15,C16,C17,C18,C19,C20
LOCAL  NOSAL
PUSHF
POP RF
MOV RAX, AX
MOV RBX, BX
MOV RCX, CX
MOV RDX, DX
MOV RSP, SP
MOV RBP, BP
MOV RDI, DI
MOV RSI, SI
MOV RCS, CS
MOV RDS, DS
MOV RES, ES
MOV RSS, SS
VEREG ETIQAX, RAX
CALL ESPACIO
VEREG ETIQBX, RBX
CALL ESPACIO
VEREG ETIQCX, RCX
CALL ESPACIO
VEREG ETIQDX, RDX
CALL ESPACIO
VEREG ETIQSP, RSP
CALL ESPACIO
VEREG ETIQBP, RBP
```

```
CALL ESPACIO
VEREG ETIQDI, RDI
CALL NUEVO
VEREG ETIQSI, RSI
CALL ESPACIO
VEREG ETIQCS, RCS
CALL ESPACIO
VEREG ETIQDS, RDS
CALL ESPACIO
VEREG ETIQES, RES
CALL ESPACIO
VEREG ETIQSS, RSS
CALL ESPACIO
VEREG ETIQF, RF
CALL NUEVO
CALL NUEVO
MOV AH, 6
MOV DL, C1
INT 21H
MOV DL, C2
INT 21H
MOV DL, C3
INT 21H
MOV DL, C4
INT 21H
MOV DL, C5
INT 21H
MOV DL, C6
INT 21H
MOV DL, C7
INT 21H
MOV DL, C8
INT 21H
MOV DL, C9
INT 21H
MOV DL, C10
INT 21H
MOV DL, C11
INT 21H
MOV DL, C12
INT 21H
MOV DL, C13
INT 21H
MOV DL, C14
INT 21H
MOV DL, C15
INT 21H
MOV DL, C16
INT 21H
MOV DL, C17
INT 21H
MOV DL, C18
INT 21H
MOV DL, C19
INT 21H
MOV DL, C20
INT 21H
```

```
CALL NUEVO
CALL NUEVO
MOV AH, 9
MOV DX,OFFSET MENS2
INT 21H
CALL TECLA
CMP AL,1BH
JNE NOSAL
MOV AH,4CH
INT 21H
NOSAL:
CLRSCR
CURSOR 0, 0
PUSH RF
POPF
MOV AX, RAX
MOV BX, RBX
MOV CX, RCX
MOV DX, RDX
MOV SP, RSP
MOV BP, RBP
MOV DI, RDI
MOV SI, RSI
;MOV CS,RCS
MOV DS, RDS
MOV ES, RES
MOV SS, RSS
ENDM
;Definición del segmento de pila
STAC   SEGMENT STACK
       DW 1024 DUP (?)
STAC ENDS
;Datos del programa del usuario y del 8086dep.asm
DATOS SEGMENT

MENS2 DB '                PRESIONE UNA TECLA PARA CONTINUAR$'
ETIQF DB 'FL = $'
ETIQAX DB 'AX = $'
ETIQBX DB 'BX = $'
ETIQCX DB 'CX = $'
ETIQDX DB 'DX = $'
ETIQSP DB 'SP = $'
ETIQBP DB 'BP = $'
ETIQSI DB 'SI = $'
ETIQDI DB 'DI = $'
ETIQCS DB 'CS = $'
ETIQDS DB 'DS = $'
ETIQES DB 'ES = $'
ETIQSS DB 'SS = $'
RF   DW 0
RAX   DW 0
RBX   DW 0
RCX   DW 0
RDX   DW 0
RSP   DW 0
RBP   DW 0
RSI   DW 0
```

```
RDI    DW 0
RCS    DW 0
RDS    DW 0
RES    DW 0
RSS    DW 0
C1     DB 0
C2     DB 0
C3     DB 0
C4     DB 0
C5     DB 0
C6     DB 0
C7     DB 0
C8     DB 0
C9     DB 0
C10    DB 0
C11    DB 0
C12    DB 0
C13    DB 0
C14    DB 0
C15    DB 0
C16    DB 0
C17    DB 0
C18    DB 0
C19    DB 0
C20    DB 0
MES1   DB 13,10,'$'
DATOS ENDS
CODE SEGMENT
        ASSUME CS:CODE,DS:DATOS,SS:STAC
        MAIN   PROC FAR
               MOV AX,DATOS
MOV DS,AX
.386
DEPURAR "M","O","V"," ","D","L",",","0","5","H","","","","","","","","","",""
               MOV DL,05H
DEPURAR "E","T","I","Q",":","","","","","","","","","","","","","","",""
        ETIQ:
DEPURAR "M","O","V"," ","A","X",",","0","1","2","3","H","","","","","","","",""
               MOV AX,0123H
DEPURAR "A","D","D"," ","A","X",",","0","0","2","5","H","","","","","","","",""
               ADD AX,0025H
DEPURAR "M","O","V"," ","B","X",",","A","X","","","","","","","","","","",""
               MOV BX,AX
DEPURAR "A","D","D"," ","B","X",",","A","X","","","","","","","","","","",""
               ADD BX,AX
DEPURAR "M","O","V"," ","C","X",",","B","X","","","","","","","","","","",""
               MOV CX,BX
DEPURAR "S","U","B"," ","C","X",",","A","X","","","","","","","","","","",""
               SUB CX,AX
DEPURAR "S","U","B"," ","A","X",",","A","X","","","","","","","","","","",""
               SUB AX,AX
DEPURAR "D","E","C"," ","D","L","","","","","","","","","","","","","",""
               DEC DL
DEPURAR "J","N","Z"," ","E","T","I","Q","","","","","","","","","","","",""
               JNZ ETIQ
DEPURAR "N","O","P","","","","","","","","","","","","","","","","",""
               NOP
```

```
DEPURAR "M","O","V"," ","A","H",",","4","C","H","","","","","","","","","","","","","","
                MOV AH,4CH
                INT 21H
        NUEVO   PROC NEAR    ;Subrutina para mostrar un mensaje en pantalla
                MOV AH,9
                MOV DX,OFFSET MES1
                INT 21H
                RET
        NUEVO ENDP
        TECLA   PROC NEAR    ;Subrutina para capturar una tecla pulsada
                MOV AH,6
                MOV DL,0FFH
                INT 21H
                JE TECLA
                RET
        TECLA ENDP
        ESPACIO   PROC NEAR ;Subrutina para mostrar un espacio en blanco
                MOV AH,6
                MOV DL,''
                INT 21H
                RET
        ESPACIO ENDP
        DESP   PROC NEAR      ;Subrutina para presentar el valor almacenado en cada registro.
                XOR CX,CX
                MOV BX,10H
        DESP1: XOR DX,DX
                DIV BX
                PUSH DX
                INC CX
                OR AX,AX
                JNE DESP1
                MOV AH,6
        DESP2: POP DX
                CMP DL,0AH
                JB BASC
                ADD DL,07H
        BASC:   ADD DL,'0'
                INT 21H
                LOOP DESP2
                RET
        DESP ENDP
        CODE   ENDS
        MAIN ENDP
END     MAIN
```

Archivo tesis.bat

```
@ECHO OFF
TASM C:\VB\SAMPLES\SDI-86\EJCOM.ASM /I.
TLINK C:\VB\SAMPLES\SDI-86\EJCOM.OBJ
OUTSCR
ECHO.
```

Archivo Sim8086.asm

```
;Macro para mostrar el valor de la salida.
ValSal  macro f,colr,s7,s6,s5,s4,s3,s2,s1,s0
        cursor f,16h
        led s7, colr, 1
        cursor f,18h
        led s6, colr, 1
        cursor f,1ah
        led s5, colr, 1
        cursor f,1ch
        led s4, colr, 1
        cursor f,1eh
        led s3, colr, 1
        cursor f,20h
        led s2, colr, 1
        cursor f,22h
        led s1, colr, 1
        cursor f,24h
        led s0, colr, 1
endm
ValIn   macro f,colr,s3,s2,s1,s0    ;Macro para mostrar el valor de la entrada.
        cursor f,2eh
        led s3, colr, 1
        cursor f,30h
        led s2, colr, 1
        cursor f,32h
        led s1, colr, 1
        cursor f,34h
        led s0, colr, 1
endm
Led     macro car,color,numl        ;Macro para simular un LED.
local   mcol
        mov ah,09h
        mov al, car
        mov bh,00h
        mov bl,00h
        cmp al,1fh
        je mcol
        mov bl, color
mcol:   mov cx, numl
        int 10h
endm
EntSal  macro               ;Macro para dibujar las entradas y salidas.
        cursor 12,10h
        pintar '',2bh
        cursor 13,10h
        pintar '',2bh
```

```
        cursor 14,10h
        pintar ' ',2bh
        cursor 15,10h
        led ' ',61h,2bh
        cursor 16,10h
        led ' ',61h,2bh
        cursor 17,10h
        led ' ',61h,2bh
        cursor 18,10h
        led ' ',61h,2bh
        cursor 19,10h
        led ' ',61h,2bh
        cursor 20,10h
        led ' ',61h,2bh
        ValSal 15,61h,'|','|','|','|','|','|','|','|'
        ValSal 16,09h,'0','0','0','0','0','0','0','0'
        ValSal 17,61h,'|','|','|','|','|','|','|','|'
        ValSal 18,47h,1fh,1fh,1fh,1fh,1fh,1fh,1fh,1fh
        ValSal 19,61h,'|','|','|','|','|','|','|','|'
        ValSal 20,61h,'=','=','=','=','=','=','=','='
        ValIn 15,61h,'^','^','^','^'
        ValIn 16,61h,'|','|','|','|'
        ValIn 17,61h,'|','|','|','|'
        ValIn 18,9,'0','0','0','0'
Endm
etiqdsp macro etqdsp        ;Macro para presentar la etiqueta del display.
local   otroc
        mov dl,20h
        mov di,11
        mov si,offset etqdsp
otroc:  cursor 9,dl
        mov ah,09h
        mov al,[si]
        mov bh,00h
        mov bl,81h
        mov cx, 1
        int 10h
        inc dl
        inc si
        dec di
        jnz otroc
endm
pintar  macro car,numc      ;Macro para pintar un carácter con el color y posición especificados
        mov ah,09h
        mov al,car
        mov bh,00h
        mov bl,81h
        mov cx,numc
        int 10h
endm
dsp     macro etqd          ;Macro para presentar el display.
        cursor 2,10h
        pintar ' ',2bh
        cursor 3,10h
        pintar ' ',2bh
        cursor 4,10h
        pintar ' ',2bh
```

```
        cursor 5,10h
        pintar '',05h
        cursor 6,10h
        pintar '',05h
        cursor 7,10h
        pintar '',05h
        cursor 5,36h
        pintar '',05h
        cursor 6,36h
        pintar '',05h
        cursor 7,36h
        pintar '',05h
        cursor 8,10h
        pintar '',2bh
        cursor 9,10h
        pintar '',2bh
        cursor 10,10h
        pintar '',2bh
        etiqdsp etqd
        cursor 6,16h
        pintar '',01h
        cursor 6,18h
        pintar '',01h
        cursor 6,1Ah
        pintar '',01h
        cursor 6,1Ch
        pintar '',01h
        cursor 6,1Eh
        pintar '',01h
        cursor 6,20h
        pintar '',01h
        cursor 6,22h
        pintar '',01h
        cursor 6,24h
        pintar '',01h
        cursor 6,26h
        pintar '',01h
        cursor 6,28h
        pintar '',01h
        cursor 6,2Ah
        pintar '',01h
        cursor 6,2Ch
        pintar '',01h
        cursor 6,2Eh
        pintar '',01h
        cursor 6,30h
        pintar '',01h
        cursor 6,32h
        pintar '',01h
        cursor 6,34h
        pintar '',01h
endm
Cursor  macro f,c       ;Macro para borrar pantalla.
        mov ah,02h
        mov bh,00h
        mov dh,f
        mov dl,c
```

```
        int 10h
endm
Clrscr  macro           ;Macro para limpiar pantalla.
        mov ax,0600h
        mov bh,07h
        mov cx,0000h
        mov dx,184fh
        int 10h
endm
stac    segment stack
        dw 1024 dup (?)
stac ends
datos segment   ;Datos de la aplicación del usuario y del archivo Sim8086.asm
m1      db 'PROGRAMA:<1> JUEGO 1<2> JUEGO 2<$> SALIRELIJA OPCION$'
m2      db 'Adivine palabra$'
m3      db 'Otro jugador$'
m4      db 'Elija secuencia$'
m5      db '7 para terminar$'
m6      db 'Fin de secuencia$'
wrd     db 'POLITECNICA$'
raya    db 5fh
nr      db 0
ni      db 0
nl      db 0
d0s1    db 81h
d1s1    db 42h
d2s1    db 24h
d3s1    db 18h
d4s1    db 3ch
d5s1    db 7eh
d6s1    db 0ffh
d7s1    db 0e7h
d0s2    db 18h
d1s2    db 24h
d2s2    db 42h
d3s2    db 81h
d4s2    db 18h
d5s2    db 24h
d6s2    db 42h
d7s2    db 81h
d0s3    db 81h
d1s3    db 42h
d2s3    db 24h
d3s3    db 18h
d4s3    db 81h
d5s3    db 42h
d6s3    db 24h
d7s3    db 18h
d0s4    db 7fh
d1s4    db 3fh
d2s4    db 1fh
d3s4    db 0fh
d4s4    db 07h
d5s4    db 03h
d6s4    db 01h
d7s4    db 00h
etqdsp  db 'LCD - 44780$'
```

```
car     db 0
col     db 0
sd0     db 0
sd1     db 0
sd2     db 0
sd3     db 0
sd4     db 0
sd5     db 0
sd6     db 0
sd7     db 0
tbx     dw 0
tcx     dw 0
tdx     dw 0
tsi     dw 0
tdi     dw 0
datos   ends
code segment
        assume cs:code,ds:datos,ss:stac
        main    proc far
                mov ax,datos
                mov ds,ax
                EntSal          ;Llamada a macro para mostrar el entorno de simulación
                call dspfnc0    ;Aplicación del usuario
                mov cx,52       ;Presenta información sobre el funcionamiento del programa
                mov si,offset m1
        sigcar1: mov al,[si]
                mov car,al
                call dspfnc1
                inc col
                cmp si,8
                jz Stop1        ;Para luego del mensaje "PROGRAMA:"
                cmp si,19
                jz Stop1        ;Para luego del mensaje "<1> JUEGO 1"
                cmp si,30
                jz Stop1        ;Para luego del mensaje "<2> JUEGO 2"
                cmp si,39
                jnz NoStop      ;Para luego del mensaje "<S> SALIR"
        Stop1:  call tclfnc0
                cmp al,'C'      ;Continua si se presiona la letra C
                jnz Stop1       ;Para luego del mensaje "ELIJA OPCION:"
                call dspfnc0
        NoStop: inc si
                loop sigcar1
        g1:     call tclfnc0
                cmp al,'1'      ;Compara si se presionó 1
                jnz g2          ;Si no se presionó 1 salta a g2
                call dspfnc0
                mov cx,15
                mov si,offset m2
                call premen
        Stop2:  call tclfnc0
                cmp al,'C'
                jnz Stop2
                call game1      ;Ejecuta juego 1
                call dspfnc0
                mov cx,12       ;Presenta mensaje "Adivine palabra"
                mov si,offset m3
```

```
            call premen
g2:         cmp al,'2'          ;Compara si se presionó 2
            jnz nogame
            call dspfnc0
            mov cx,15           ;Presenta mensaje "Elija secuencia"
            mov si,offset m4
            call premen
Stop3:      call tclfnc0
            cmp al,'C'          ;Continua si se presiona la letra C
            jnz Stop3
            call dspfnc0
            mov cx,15
            mov si,offset m5    ;Presenta mensaje "7 para terminar"
            call premen
            call game2          ;Ejecuta juego 2
            call dspfnc0
            mov cx,16
            mov si,offset m6    ;Presenta mensaje "Fin de secuencia"
            call premen
nogame:     cmp al,'S'
            jnz g1
            mov ah,4ch                  ;salir al DOS
            int 21h
            game1   proc near;Juego de adivinar palabra
            call dspfnc0
            mov nr,0bh          ;Carga nr con el número de '_'
            mov col,03h
repet:      mov car,'_'
            call dspfnc1
            inc col             ;Incrementa en 1 la posición del próximo caracter '_'
            dec nr              ;Decrementa el número de '_' a imprimir
            jne repet           ;Repite mientras hayan más '_' a imprimir
            mov col,0fh         ;Carga la posición donde se imprimirá el 9
            mov car,'9'
            call dspfnc1
            mov ni,09h          ;Carga el número de intentos en ni
denuevo:    mov col,00h         ;Carga en col la posión inicial
            call leercar        ;Llama a la subrutina para leer caracter
            mov col,03h         ;Carga en col la posición de inicio de impresión de la palabra
            mov nl,0bh          ;Carga en nl el número de letras de la palabra
            mov si,offset wrd ;Carga la dirección del caracter inicial de la palabra
            mov al,car
comp:       cmp al,[si]         ;Compara el caracter ingresado con cada caracter de la palabra
            jne salto           ;Si son diferentes continua en salto
            call dspfnc1
salto:      inc col             ;Incrementa la posición de impresión
            inc si              ;Incrementa la dirección del próximo caracter de la palabra
            dec nl              ;Decrementa el número de letras de la palabra
            jne comp            ;Continua en comp hasta que nl sea cero
            add ni,2fh          ;Convierte el número de intentos a su valor ASCII menos 1
            mov col,0fh         ;Carga en col la posición 0fh
            mov al,ni
            mov car,al
            call dspfnc1
            sub ni,2fh          ;Regresa ni a su valor original
            dec ni              ;Decrementa en 1 el número de intentos
            jne denuevo         ;Continua en denuevo mientras el número de intentos no es cero
```

```
                ret
        endp
        game2   proc near;Juego de luces programable
repsec:
        call InSDI
        cmp al,07h          ;Compara si se ingresado 7 por el puerto de entrada
        jz Stop
        and al,03h
        cmp al,00h
        jnz salt1
        mov si,offset d0s1;Selecciona secuencia 1 si se ingresó 0
salt1:  cmp al,01h
        jnz salt2
        mov si,offset d0s2;Selecciona secuencia 1 si se ingresó 1
salt2:  cmp al,02h
        jnz salt3
        mov si,offset d0s3;Selecciona secuencia 1 si se ingresó 2
salt3:  cmp al,03h
        jnz salt4
        mov si,offset d0s4;Selecciona secuencia 1 si se ingresó 3
salt4:  mov cx,08h
ndat:   mov al,[si]
        call OutSDI         ;Muestra secuencia seleccionada
        call retcomp
        inc si
        loop ndat
        jmp repsec
Stop:   ret
        endp
premen  proc near           ;Subrutina para presentar mensaje.
        mov al,[si]
        mov car,al
        call dspfnc1
        inc col
        inc si
        loop premen
        ret
premen  endp
leercar proc near           ;Subrutina para leer carácter.
        call tclfnc0
        mov col,00h
        mov car,al
        call dspfnc1
        ret
leercar endp
retcomp proc near;Subrutina de retardo.
        mov di, 40
newret: mov bx,0ffffh
ret1:   dec bx
        jnz ret1
        dec di
        jnz newret
        ret
retcomp endp
InSDI   proc near           ;Subrutina para simular el puerto de entrada del SDI – 86S.
        mov tbx, bx
        mov tcx, cx
```

```
        mov tdx, dx
        mov tsi, si
        mov tdi, di
        Call tecla
        cmp al,30h
        jne slt0
        ValIn 18,9,'0','0','0','0'
        mov al,00h
slt0:   cmp al,31h
        jne slt1
        ValIn 18,9,'0','0','0','1'
        mov al,01h
slt1:   cmp al,32h
        jne slt2
        ValIn 18,9,'0','0','1','0'
        mov al,02h
slt2:  cmp al,33h
        jne slt3
        ValIn 18,9,'0','0','1','1'
        mov al,03h
slt3:   cmp al,34h
        jne slt4
        ValIn 18,9,'0','1','0','0'
        mov al,04h
        slt4:     cmp al,35h
        jne slt5
        ValIn 18,9,'0','1','0','1'
        mov al,05h
slt5:   cmp al,36h
        jne slt6
        ValIn 18,9,'0','1','1','0'
        mov al,06h
slt6:   cmp al,37h
        jne slt7
        ValIn 18,9,'0','1','1','1'
        mov al,07h
slt7:   cmp al,38h
        jne slt8
        ValIn 18,9,'1','0','0','0'
        mov al,08h
slt8:   cmp al,39h
        jne slt9
        ValIn 18,9,'1','0','0','1'
        mov al,09h
slt9:   cmp al,41h
        jne slt10
        ValIn 18,9,'1','0','1','0'
        mov al,0ah
slt10:  cmp al,42h
        jne slt11
        ValIn 18,9,'1','0','1','1'
        mov al,0bh
slt11:  cmp al,43h
        jne slt12
        ValIn 18,9,'1','1','0','0'
        mov al,0ch
slt12:  cmp al,44h
```

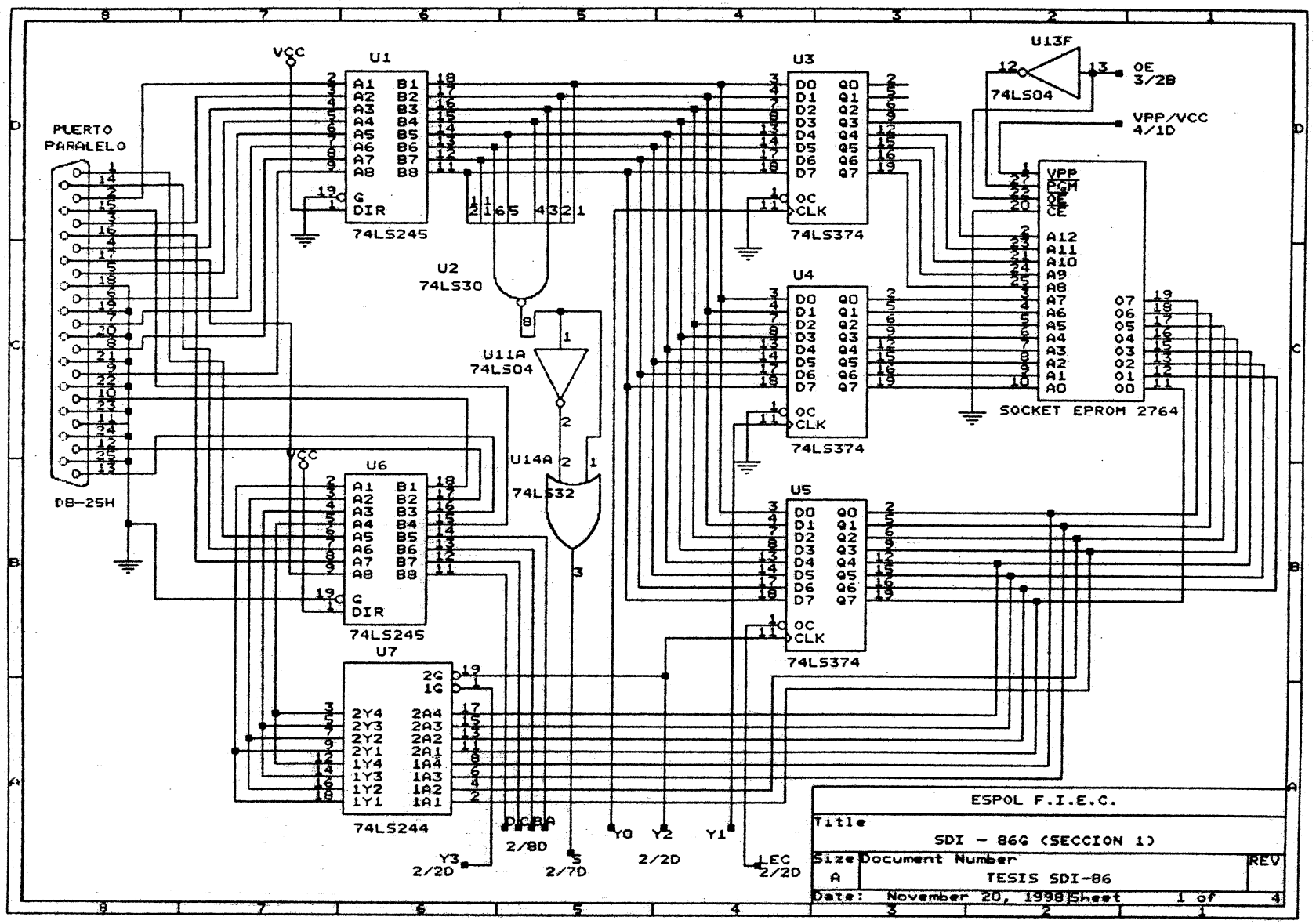
```
        jne slt13
        ValIn 18,9,'1','1','0','1'
        mov al,0dh
slt13:  cmp al,45h
        jne slt14
        ValIn 18,9,'1','1','1','0'
        mov al,0eh
slt14:  cmp al,46h
        jne slt15
        ValIn 18,9,'1','1','1','1'
        mov al,0fh
slt15:  mov ah,00h
        cursor 24, 0
        mov bx, tbx
        mov cx, tcx
        mov dx, tdx
        mov si, tsi
        mov di, tdi
        ret
endp
OutSDI  proc near          ;Subrutina para simular el puerto de salida del SDI – 86S.
        mov tbx, bx
        mov tcx, cx
        mov tdx, dx
        mov tsi, si
        mov tdi, di
        mov ah, 0
        mov dh, 2
        div dh
        cmp al, 1
        ja slto0
        mov sd0, al
slto0:  mov sd0, ah
        div dh
        cmp al, 1
        ja slto1
        mov sd1, al
slto1:  mov sd1, ah
        div dh
        cmp al, 1
        ja slto2
        mov sd2, al
slto2:  mov sd2, ah
        div dh
        cmp al, 1
        ja slto3
        mov sd3, al
slto3:  mov sd3, ah
        div dh
        cmp al, 1
        ja slto4
        mov sd4, al
slto4:  mov sd4, ah
        div dh
        cmp al, 1
        ja slto5
        mov sd5, al
```

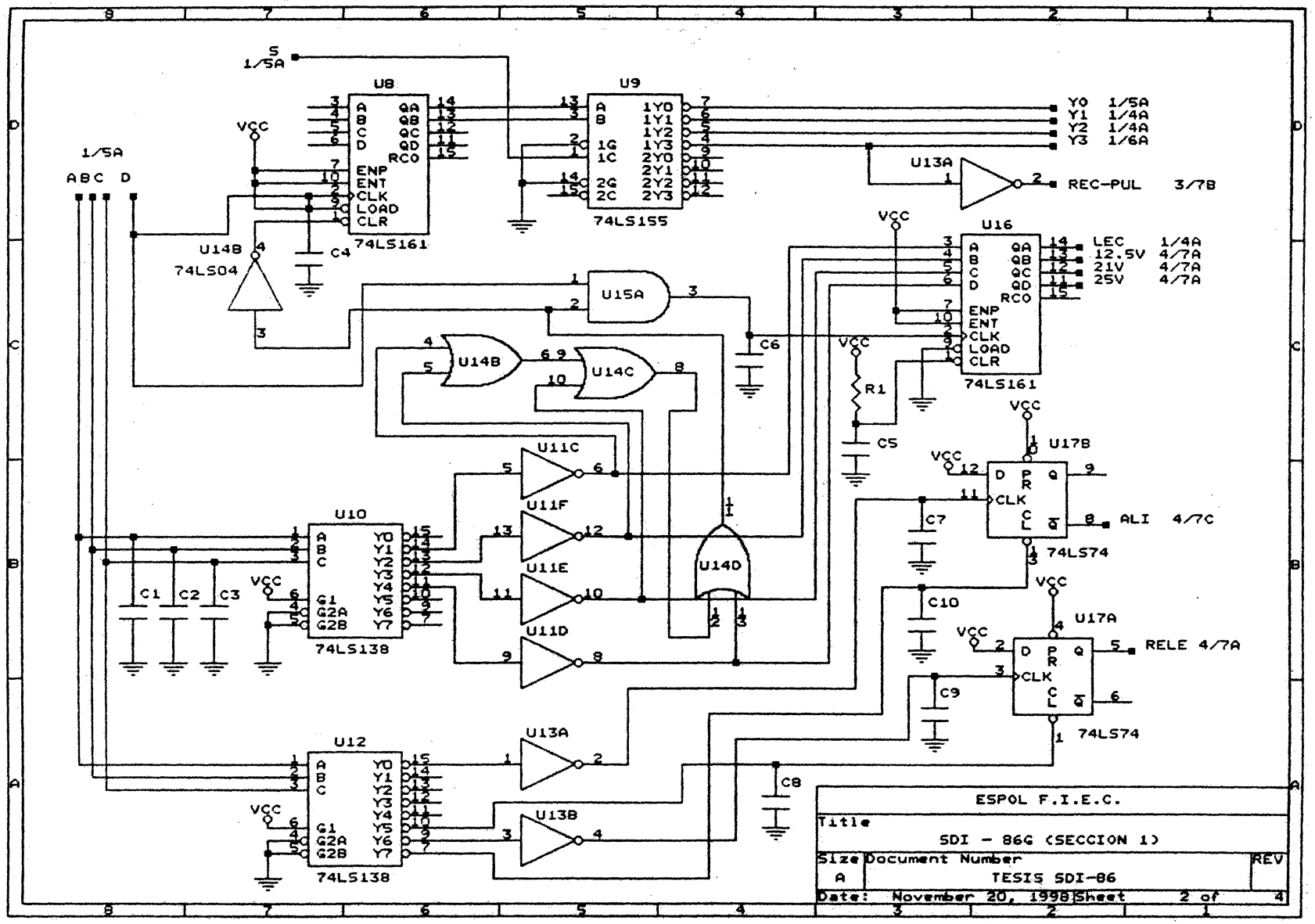
```
slto5:  mov sd5, ah
        div dh
        cmp al, 1
        ja slto6
        mov sd6, al
slto6:  mov sd6, ah
        div dh
        cmp al, 1
        ja slto7
        mov sd7, al
slto7:  mov sd7, ah
        add sd0,1fh
        add sd1,1fh
        add sd2,1fh
        add sd3,1fh
        add sd4,1fh
        add sd5,1fh
        add sd6,1fh
        add sd7,1fh
        ValSal 18,47h,sd7,sd6,sd5,sd4,sd3,sd2,sd1,sd0
        add sd0,11h
        add sd1,11h
        add sd2,11h
        add sd3,11h
        add sd4,11h
        add sd5,11h
        add sd6,11h
        add sd7,11h
        ValSal 16,09h,sd7,sd6,sd5,sd4,sd3,sd2,sd1,sd0
        cursor 24, 0
        mov bx, tbx
        mov cx, tcx
        mov dx, tdx
        mov si, tsi
        mov di, tdi
        ret
endp
tclfnc0 proc near       ;Subrutina para simular la función de captura de tecla pulsada.
captura: mov ah, 6
        mov dl,0ffh
        int 21h
        je captura
        ret
endp
dspfnc0 proc near       ;Subrutina para simular la función de borrar pantalla.
        pushf
        mov tbx, bx
        mov tcx, cx
        mov tdx, dx
        mov tsi, si
        mov tdi, di
        dsp etqdsp
        cursor 6,16h
        pintar '',01h
        mov bx, tbx
        mov cx, tcx
        mov dx, tdx
```

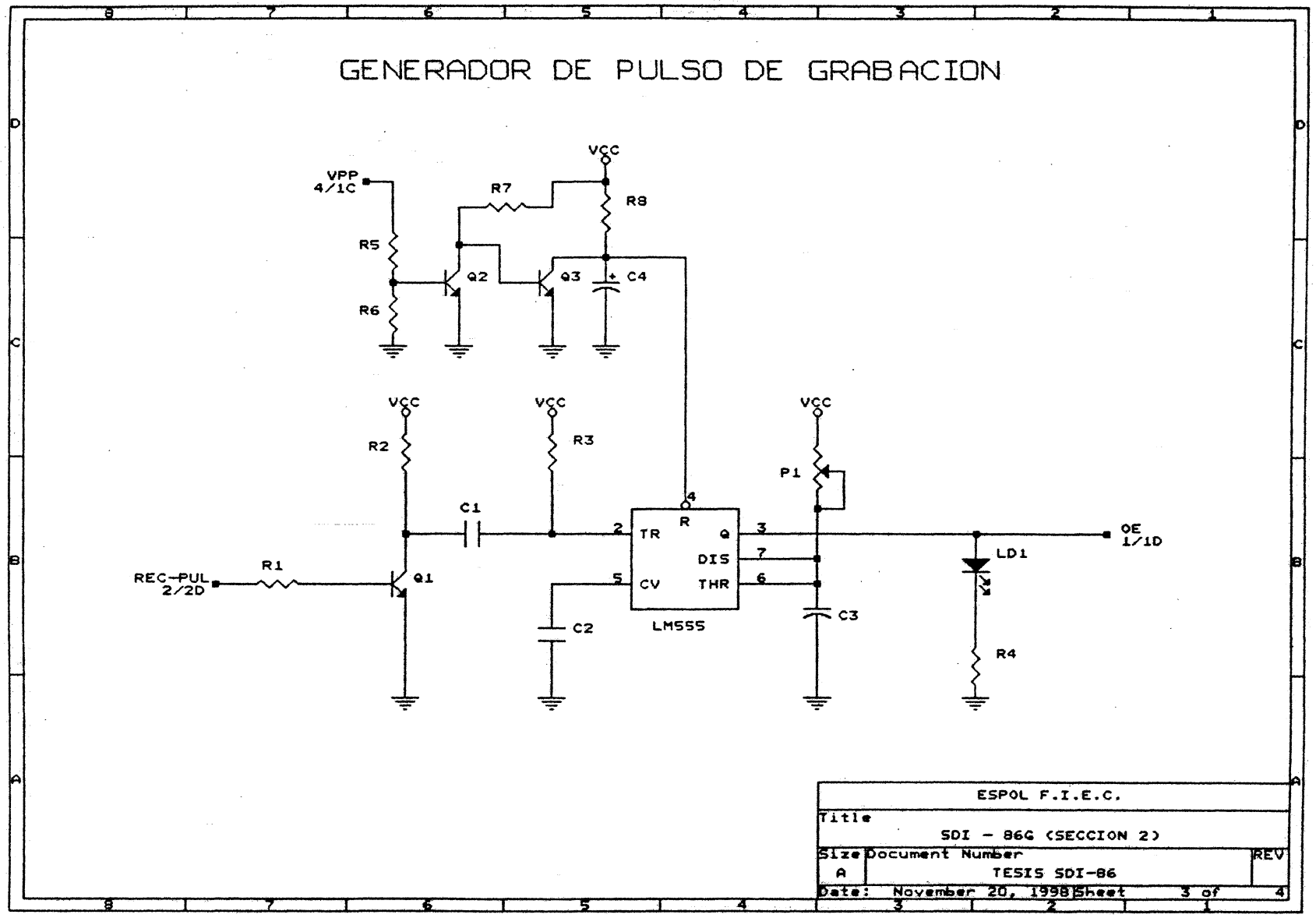
```
        mov si, tsi
        mov di, tdi
        mov col,0
        popf
        ret
endp
dspfnc1 proc near       ;Subrutina para simular la función de mostrar carácter en la columna
especificada.
        pushf
        mov tbx, bx
        mov tcx, cx
        mov tdx, dx
        mov tsi, si
        mov tdi, di
        cmp col,00h
        jne st0
        cursor 6,16h
        pintar car,01h
st0:    cmp col,01h
        jne st1
        cursor 6,18h
        pintar car,01h
st1:    cmp col,02h
        jne st2
        cursor 6,1Ah
        pintar car,01h
st2:    cmp col,03h
        jne st3
        cursor 6,1ch
        pintar car,01h
st3:    cmp col,04h
        jne st4
        cursor 6,1eh
        pintar car,01h
st4:    cmp col,05h
        jne st5
        cursor 6,20h
        pintar car,01h
st5:    cmp col,06h
        jne st6
        cursor 6,22h
        pintar car,01h
st6:    cmp col,07h
        jne st7
        cursor 6,24h
        pintar car,01h
st7:    cmp col,08h
        jne st8
        cursor 6,26h
        pintar car,01h
st8:    cmp col,09h
        jne st9
        cursor 6,28h
        pintar car,01h
st9:    cmp col,0ah
        jne st10
        cursor 6,2ah
```

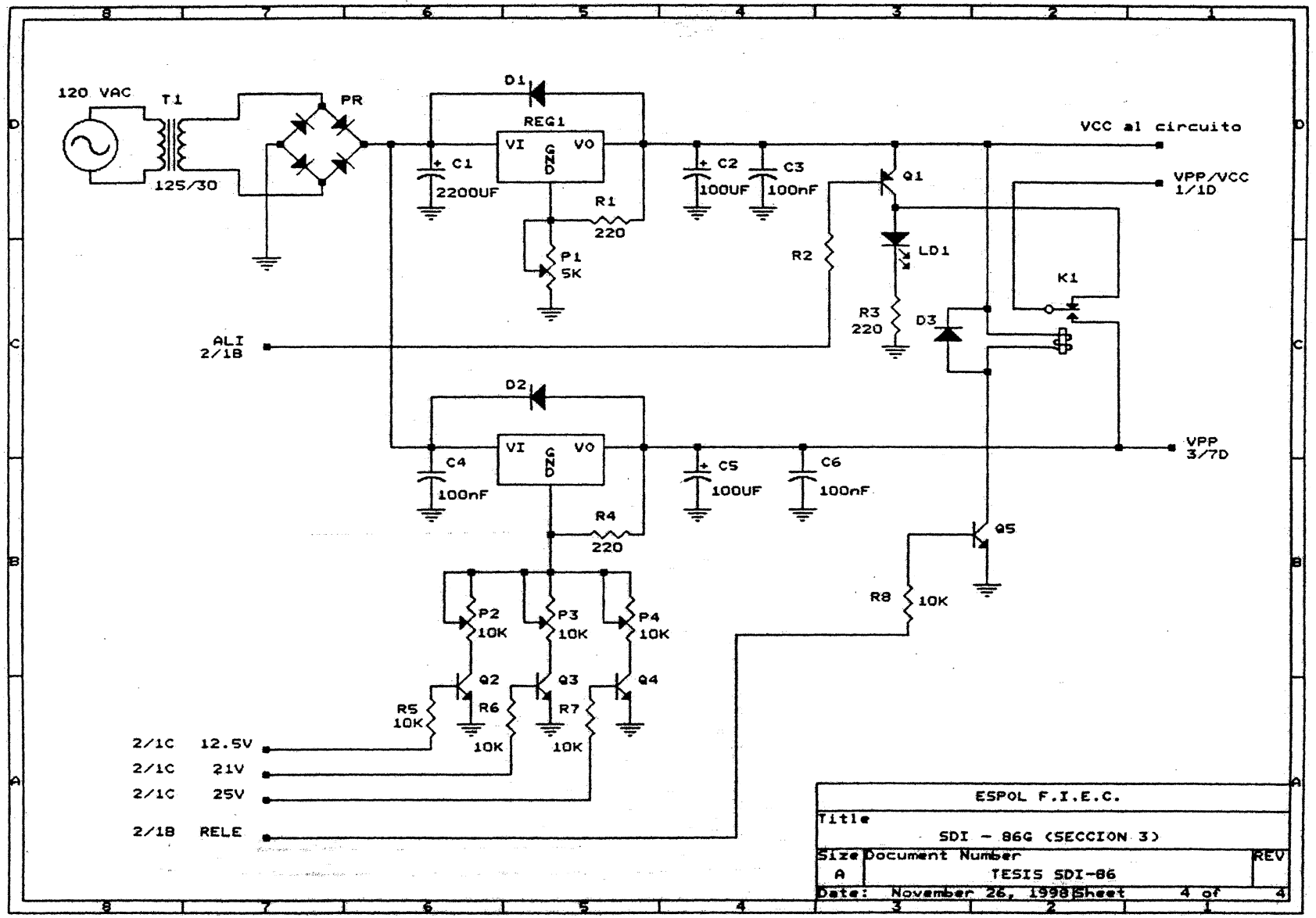
```
        pintar car,01h
st10:   cmp col,0bh
        jne st11
        cursor 6,2ch
        pintar car,01h
st11:   cmp col,0ch
        jne st12
        cursor 6,2eh
        pintar car,01h
st12:   cmp col,0dh
        jne st13
        cursor 6,30h
        pintar car,01h
st13:   cmp col,0eh
        jne st14
        cursor 6,32h
        pintar car,01h
st14:   cmp col,0fh
        jne st15
        cursor 6,34h
        pintar car,01h
st15:   nop
        mov bx, tbx
        mov cx, tcx
        mov dx, tdx
        mov si, tsi
        mov di, tdi
        popf
        ret
endp
Tecla   proc near          ;Subrutina para capturar tecla pulsada.
        mov ah,6
        mov dl,0FFh
        int 21h
        je tecla
        ret
Tecla   endp
        code    ends
        main endp
end   main
```

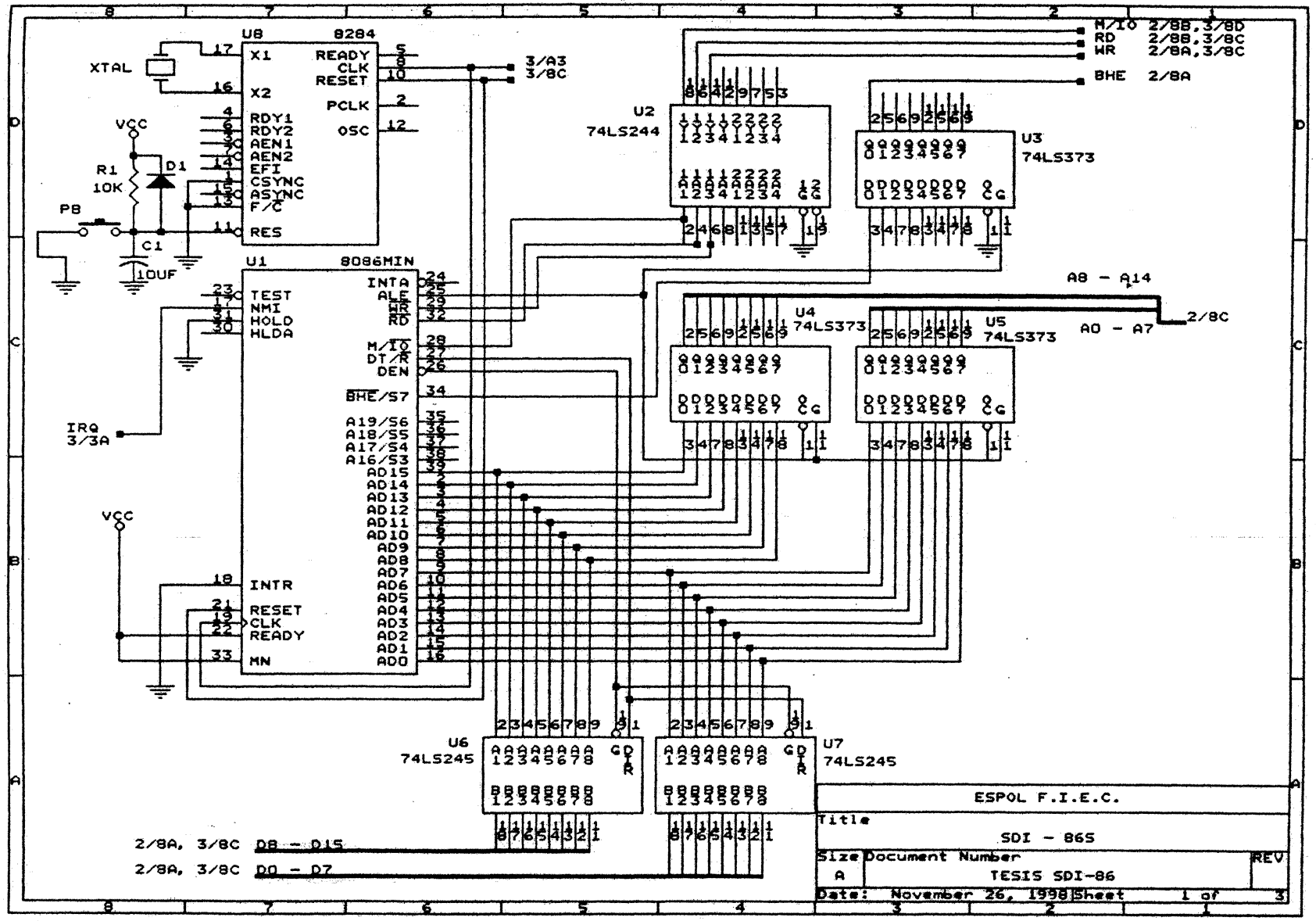
ANEXO II
DIAGRAMAS ESQUEMATICOS DEL SDI - 86G Y DEL SDI - 86S

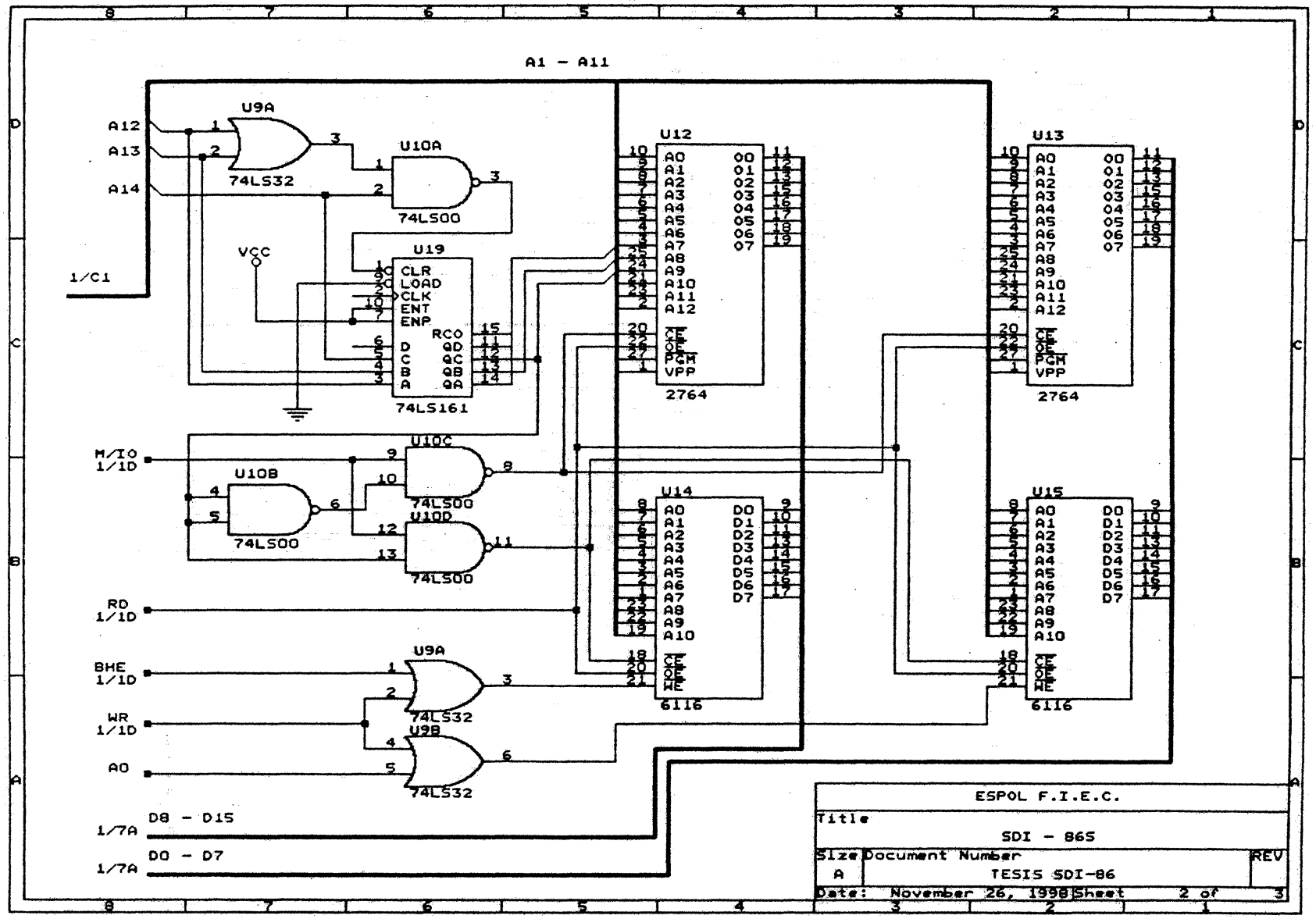

PUERTO
PARALELO
DB-25H
VCC
U1
74LS245
U2
74LS30
U11A
74LS04
U14A
74LS32
U6
74LS245
U7
74LS244
U3
74LS374
U4
74LS374
U5
74LS374
U13F
74LS04
OE
3/2B
VPP/VCC
4/1D
SOCKET EPROM 2764
2/8D
2/7D
2/2D
2/2D
2/2D
ESPOL F.I.E.C.
Title
SDI - 86G (SECCION 1)
Size
A
Document Number
TESIS SDI-86
REV
Date: November 20, 1998
Sheet 1 of 4

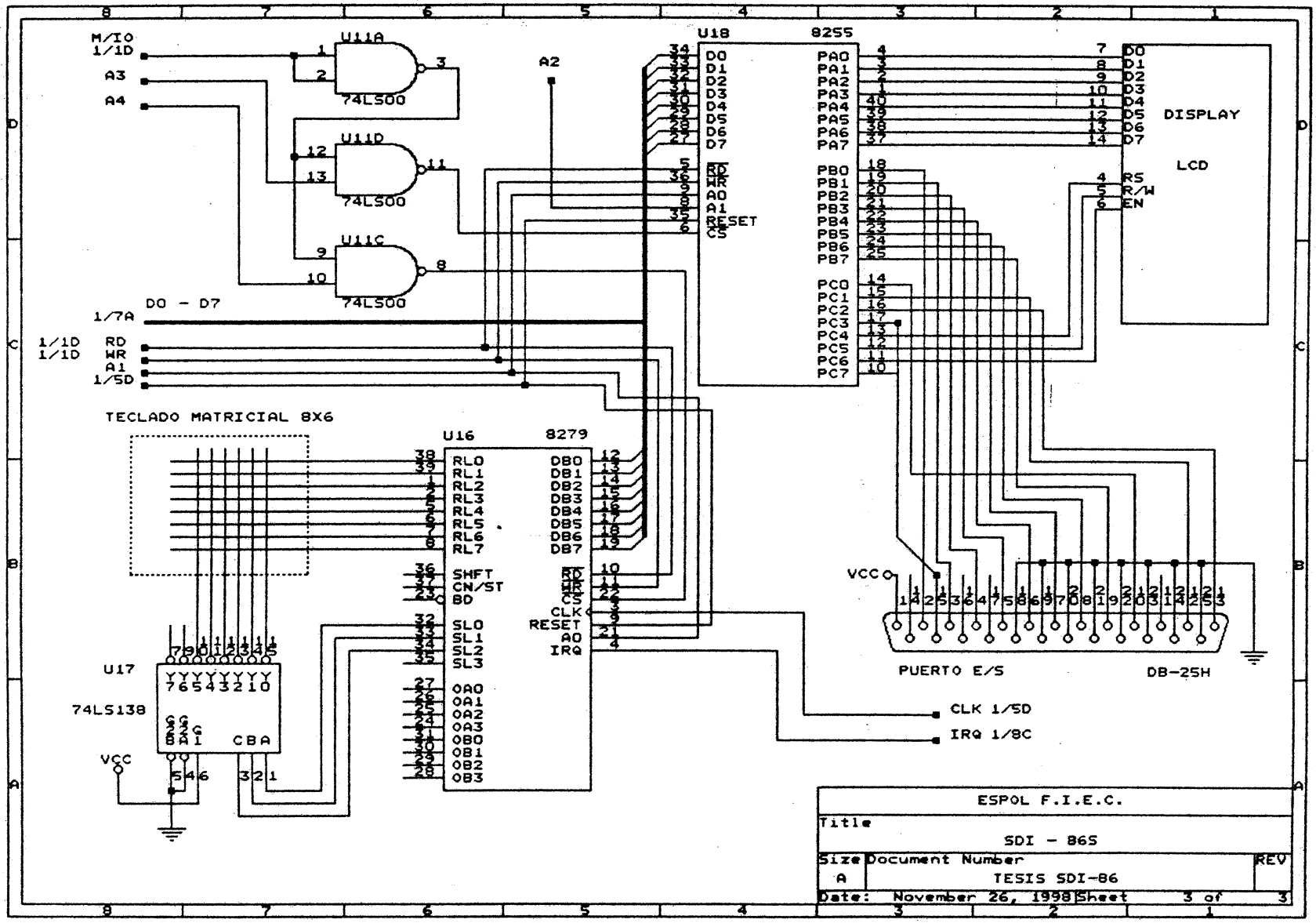

U8
U9
U16
U10
U11C
U11F
U11E
U11D
U12
U13A
U13B
U14B
U14C
U14D
U15A
U17A
U17B
74LS161
74LS155
74LS04
74LS138
74LS74
C1
C2
C3
C4
C5
C6
C7
C8
C9
C10
R1
VCC
1/5A
Y0 1/5A
Y1 1/4A
Y2 1/4A
Y3 1/6A
REC-PUL 3/7B
LEC 1/4A
12.5V 4/7A
21V 4/7A
25V 4/7A
ALI 4/7C
RELE 4/7A
ESPOL F.I.E.C.
SDI - 86G (SECCION 1)
TESIS SDI-86
November 20, 1998
Sheet 2 of 4


GENERADOR DE PULSO DE GRABACION
VCC
VPP
4/1C
R7
R8
R5
Q2
Q3
C4
R6
VCC
VCC
VCC
R2
R3
P1
C1
4
R
2
TR
Q
3
7
DIS
REC-PUL
2/2D
R1
Q1
5
CV
THR
6
OE
1/1D
LD1
C2
LM555
C3
R4
ESPOL F.I.E.C.
Title
SDI - 86G (SECCION 2)
Size
A
Document Number
TESIS SDI-86
REV
Date: November 20, 1998
Sheet 3 of 4

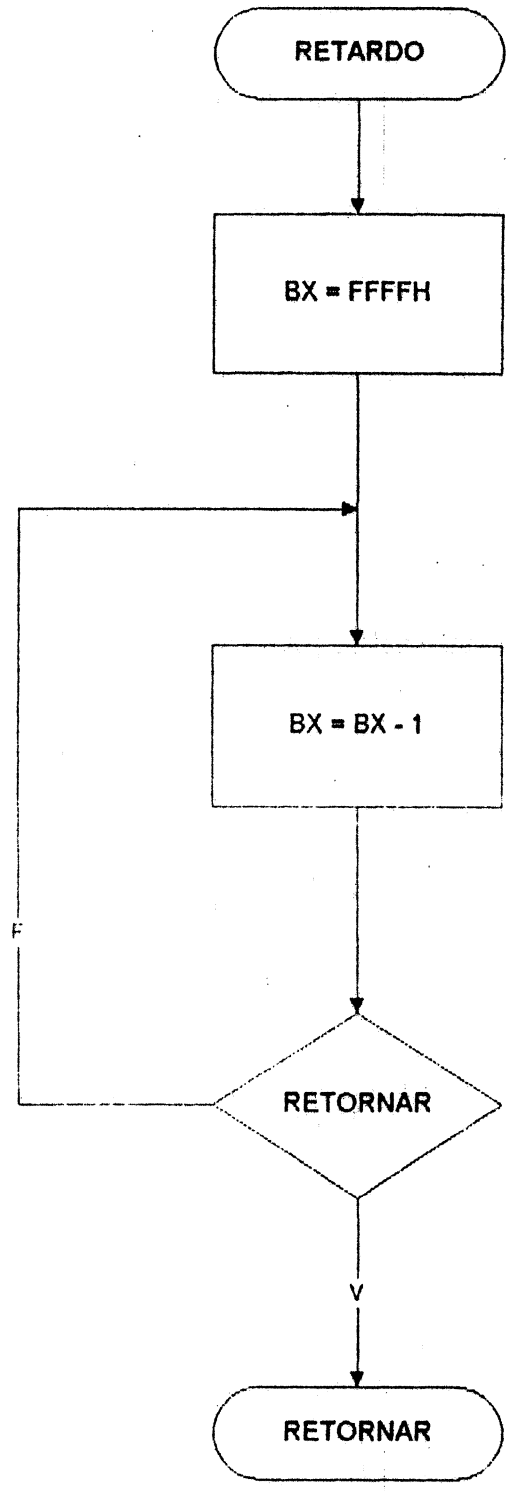

120 VAC
T1
125/30
PR
D1
REG1
VI
VO
GND
C1
2200UF
R1
220
P1
5K
C2
100UF
C3
100nF
VCC al circuito
Q1
VPP/VCC
1/1D
LD1
R2
R3
220
D3
K1
ALI
2/1B
D2
C4
100nF
C5
100UF
C6
100nF
VPP
3/7D
R4
220
Q5
R8
10K
P2
10K
P3
10K
P4
10K
Q2
Q3
Q4
R5
10K
R6
10K
R7
10K
2/1C 12.5V
2/1C 21V
2/1C 25V
2/1B RELE
ESPOL F.I.E.C.
Title
SDI - 86G (SECCION 3)
Size
A
Document Number
TESIS SDI-86
REV
Date: November 26, 1998
Sheet 4 of 4

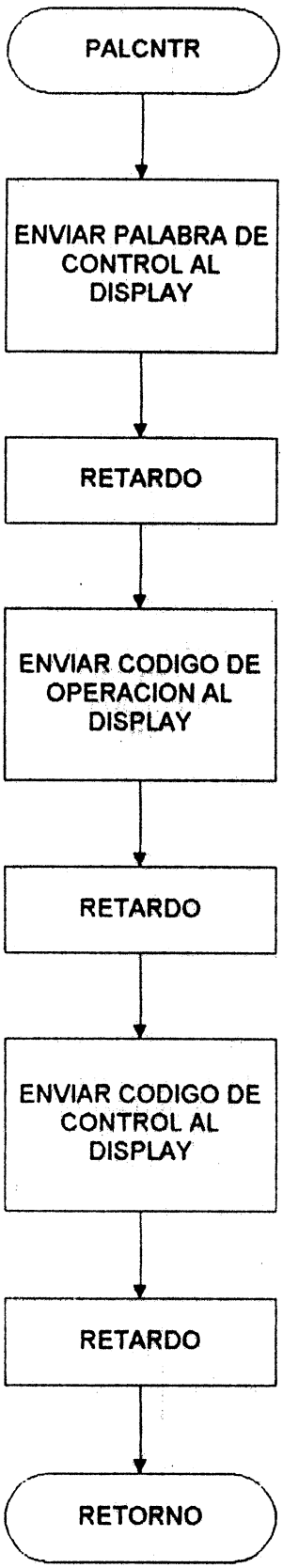

U8 8284
XTAL
X1
X2
READY
CLK
RESET
PCLK
OSC
RDY1
RDY2
AEN1
AEN2
EFI
CSYNC
ASYNC
F/C
RES
VCC
R1
10K
D1
PB
C1
10UF
3/A3
3/8C
M/IO 2/8B,3/8D
RD 2/8B,3/8C
WR 2/8A,3/8C
BHE 2/8A
U2
74LS244
U3
74LS373
U1 8086MIN
TEST
NMI
HOLD
HLDA
INTA
ALE
WR
RD
M/IO
DT/R
DEN
BHE/S7
A19/S6
A18/S5
A17/S4
A16/S3
AD15
AD14
AD13
AD12
AD11
AD10
AD9
AD8
AD7
AD6
AD5
AD4
AD3
AD2
AD1
AD0
IRQ
3/3A
A8 - A14
A0 - A7
2/8C
U4
74LS373
U5
74LS373
INTR
RESET
CLK
READY
MN
U6
74LS245
U7
74LS245
2/8A, 3/8C D8 - D15
2/8A, 3/8C D0 - D7
ESPOL F.I.E.C.
Title
SDI - 86S
Size A
Document Number
TESIS SDI-86
REV
Date: November 26, 1998 Sheet 1 of 3

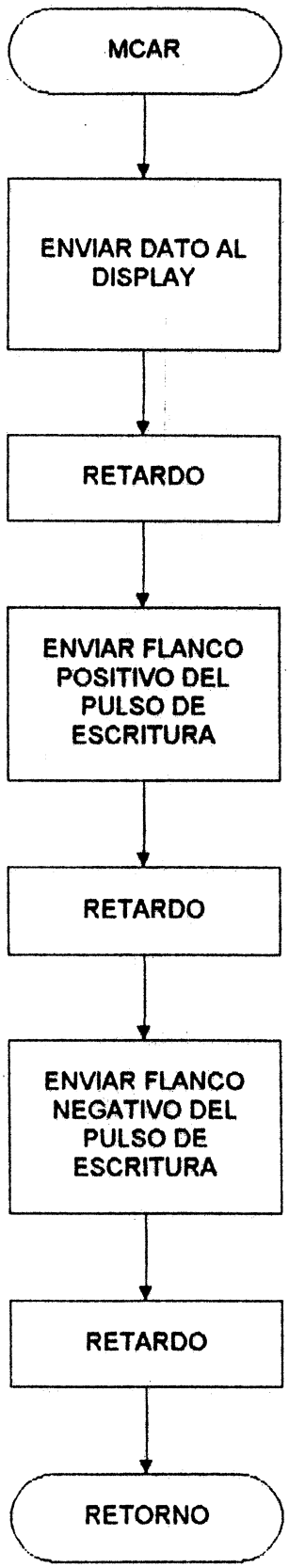

A1 - A11
A12
A13
A14
U9A
74LS32
U10A
74LS00
U19
CLR
LOAD
CLK
ENT
ENP
RCO
QD
QC
QB
QA
D
C
B
A
74LS161
VCC
1/C1
U12
U13
2764
M/IO
1/1D
U10B
U10C
U10D
74LS00
U14
U15
6116
RD
1/1D
BHE
1/1D
WR
1/1D
A0
U9A
U9B
74LS32
1/7A
D8 - D15
D0 - D7
ESPOL F.I.E.C.
Title
SDI - 865
Size
A
Document Number
TESIS SDI-86
REV
Date: November 26, 1998
Sheet 2 of 3

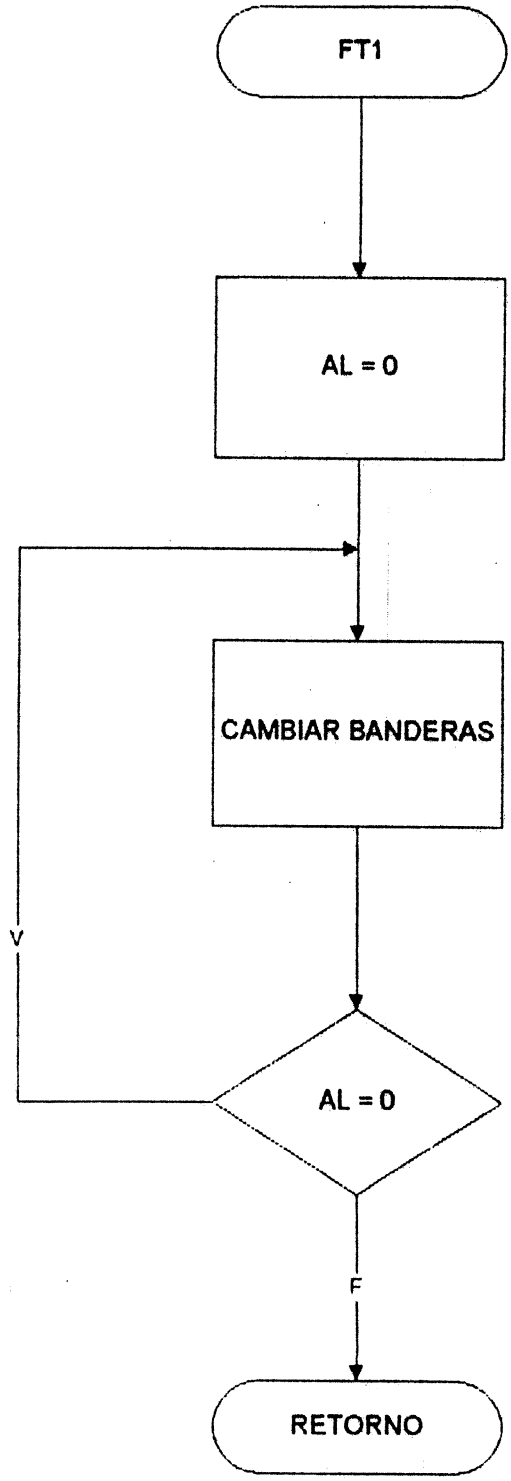

M/IO
1/1D
A3
A4
U11A
74LS00
U11D
74LS00
U11C
74LS00
A2
U18
8255
DISPLAY
LCD
RS
R/W
EN
D0 - D7
1/7A
1/1D RD
1/1D WR
A1
1/5D
TECLADO MATRICIAL 8X6
U16
8279
U17
74LS138
VCC
PUERTO E/S
DB-25H
CLK 1/5D
IRQ 1/8C
ESPOL F.I.E.C.
Title
SDI - 86S
Size
A
Document Number
TESIS SDI-86
REV
Date: November 26, 1998
Sheet 3 of 3

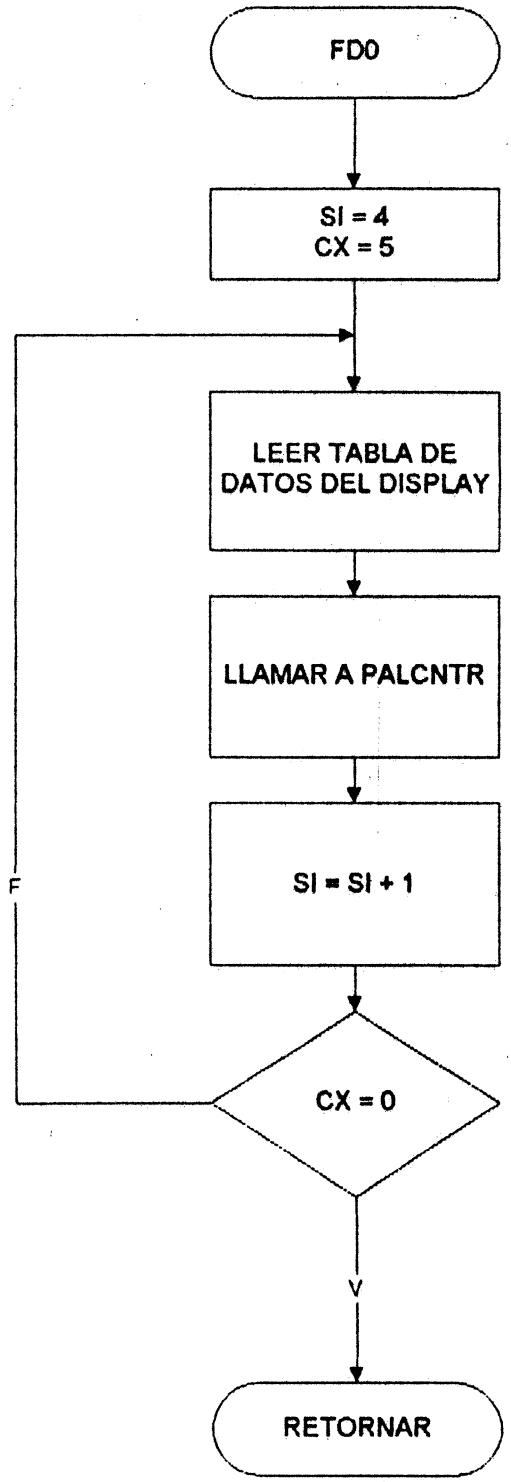
ANEXO 3
SISTEMA OPERATIVO DEL SDI - 86S

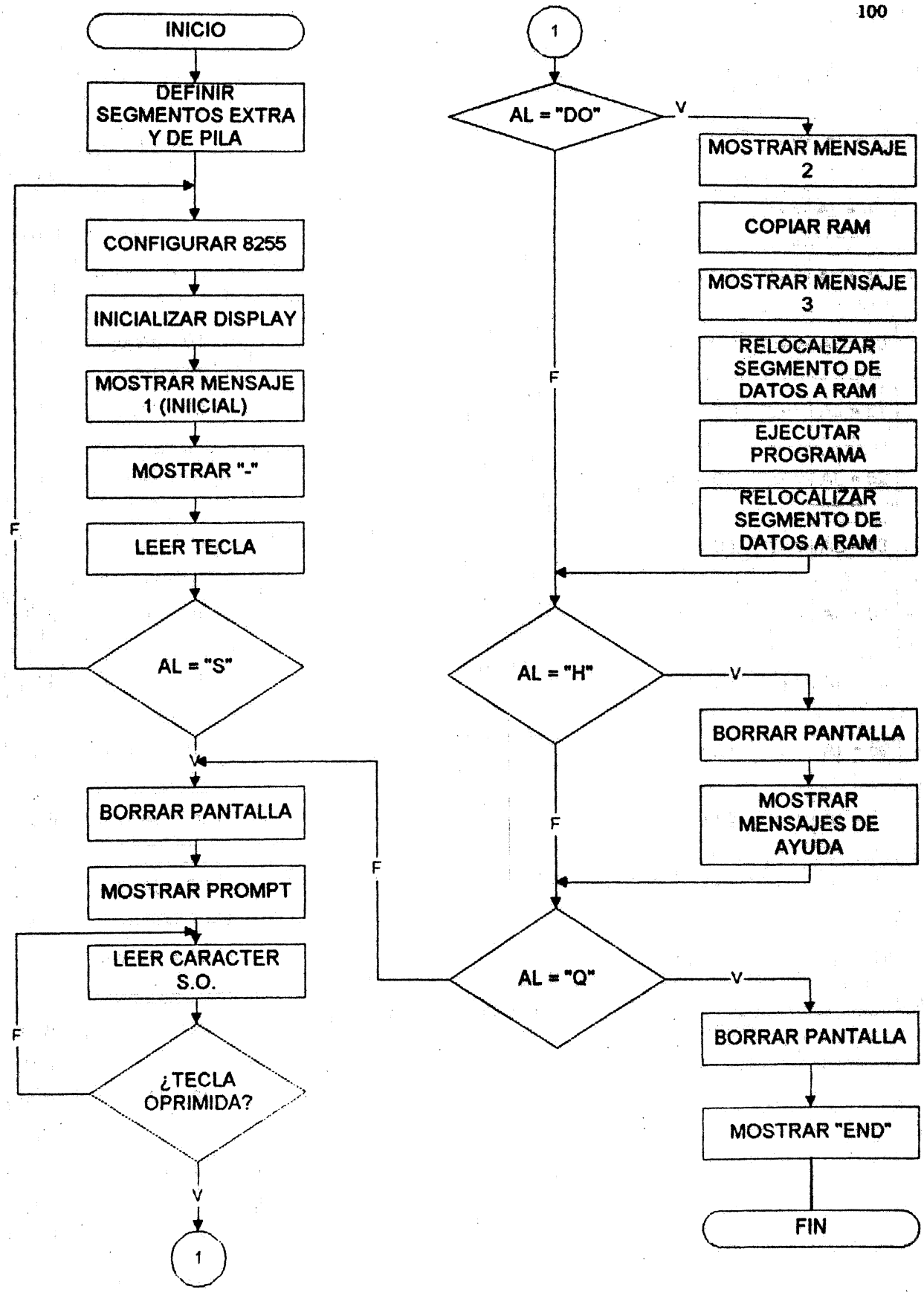

RETARDO
BX = FFFFH
BX = BX - 1
F
RETORNAR
V
RETORNAR

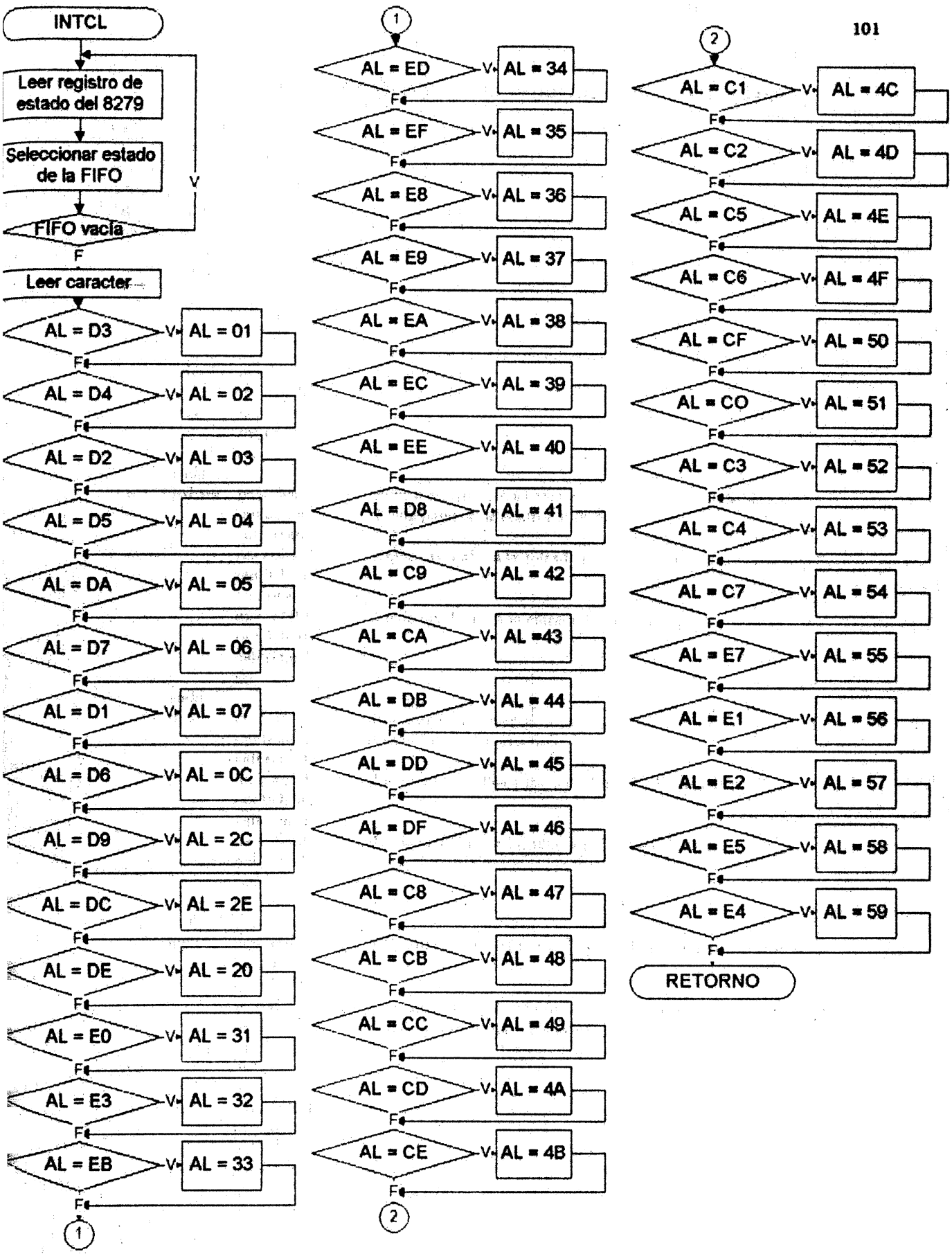

PALCNTR
ENVIAR PALABRA DE CONTROL AL DISPLAY
RETARDO
ENVIAR CODIGO DE OPERACION AL DISPLAY
RETARDO
ENVIAR CODIGO DE CONTROL AL DISPLAY
RETARDO
RETORNO


MCAR
ENVIAR DATO AL DISPLAY
RETARDO
ENVIAR FLANCO POSITIVO DEL PULSO DE ESCRITURA
RETARDO
ENVIAR FLANCO NEGATIVO DEL PULSO DE ESCRITURA
RETARDO
RETORNO


FT1
AL = 0
CAMBIAR BANDERAS
V
AL = 0
F
RETORNO


FD0
SI = 4
CX = 5
LEER TABLA DE DATOS DEL DISPLAY
LLAMAR A PALCNTR
SI = SI + 1
CX = 0
F
V
RETORNAR


INICIO
DEFINIR SEGMENTOS EXTRA Y DE PILA
CONFIGURAR 8255
INICIALIZAR DISPLAY
MOSTRAR MENSAJE 1 (INIICIAL)
MOSTRAR "-"
LEER TECLA
AL = "S"
F
V
BORRAR PANTALLA
MOSTRAR PROMPT
LEER CARACTER S.O.
¿TECLA OPRIMIDA?
F
V
1
1
AL = "DO"
V
F
MOSTRAR MENSAJE 2
COPIAR RAM
MOSTRAR MENSAJE 3
RELOCALIZAR SEGMENTO DE DATOS A RAM
EJECUTAR PROGRAMA
RELOCALIZAR SEGMENTO DE DATOS A RAM
AL = "H"
V
F
BORRAR PANTALLA
MOSTRAR MENSAJES DE AYUDA
AL = "Q"
F
V
BORRAR PANTALLA
MOSTRAR "END"
FIN


INTCL
Leer registro de estado del 8279
Seleccionar estado de la FIFO
FIFO vacía
V
F
Leer caracter
AL = D3
AL = 01
AL = D4
AL = 02
AL = D2
AL = 03
AL = D5
AL = 04
AL = DA
AL = 05
AL = D7
AL = 06
AL = D1
AL = 07
AL = D6
AL = 0C
AL = D9
AL = 2C
AL = DC
AL = 2E
AL = DE
AL = 20
AL = E0
AL = 31
AL = E3
AL = 32
AL = EB
AL = 33
1
AL = ED
AL = 34
AL = EF
AL = 35
AL = E8
AL = 36
AL = E9
AL = 37
AL = EA
AL = 38
AL = EC
AL = 39
AL = EE
AL = 40
AL = D8
AL = 41
AL = C9
AL = 42
AL = CA
AL =43
AL = DB
AL = 44
AL = DD
AL = 45
AL = DF
AL = 46
AL = C8
AL = 47
AL = CB
AL = 48
AL = CC
AL = 49
AL = CD
AL = 4A
AL = CE
AL = 4B
2
AL = C1
AL = 4C
AL = C2
AL = 4D
AL = C5
AL = 4E
AL = C6
AL = 4F
AL = CF
AL = 50
AL = CO
AL = 51
AL = C3
AL = 52
AL = C4
AL = 53
AL = C7
AL = 54
AL = E7
AL = 55
AL = E1
AL = 56
AL = E2
AL = 57
AL = E5
AL = 58
AL = E4
AL = 59
RETORNO

Código fuente del Sistema Operativo del SDI – 86S

```
stac    segment stack
        dw 1024 dup (?)
stac ends
datos segment
datos   ends
code segment
        assume cs:code,ds:datos,ss:stac
        main    proc far
                mov ax,datos
                mov ds,ax
                nop
                jmp finfunc     ;Salta a finfunc para no ejecutar subrutinas al inicio.
        retardo proc near       ;Subrutina retardo.
                mov bx,0ffffh
        ret1:   dec bx
                jne ret1
                ret
        endp
        palcntr proc near       ;Subrutina para enviar una palabra de control al display.
                out 08h,al      ;Envia palabra de control al display.
                call retardo
                mov al,10h      ;Envia nivel alto a la entrada E en el display.
                out 0ch,al
                call retardo
                mov al,00h      ;Envia nivel bajo a la entrada E en el display.
                out 0ch,al
                call retardo
                ret
        endp
        mcar    proc near       ;Subrutina para mostrar caracter.
                out 08h,al      ;Envia caracter a presentar.
                call retardo
                mov al,50h      ;Envia niveles altos a las entradas RS y E en el display.
                out 0ch,al
                call retardo
                mov al,40h      ;Envia nivel bajo a la entrada E en el display.
                out 0ch,al
                call retardo
                ret
        endp
        Intcl   proc near       ;Subrutina para capturar tecla pulsada y convertirla a ASCII.
                in al,12h
                test al,07h
                jz Intc         ;Lazo para detectar si se ha pulsado un caracter.
                in al,10h       ;Captura caracter pulsado en al.
                cmp al,0d3h     ;Convierte a ASCII según la tecla pulsada.
                jnz car1
                mov al,01h
        car1:   cmp al,0d4h
                jnz car2
                mov al,02h
        car2:   cmp al,0d2h
                jnz car3
                mov al,03h
```

```
car3:   cmp al,0d5h
        jnz car4
        mov al,04h
car4:   cmp al,0dah
        jnz car5
        mov al,05h
car5:   cmp al,0d7h
        jnz car6
        mov al,06h
car6:   cmp al,0d1h
        jnz car7
        mov al,07h
car7:   cmp al,0d6h
        jnz car8
        mov al,0ch
car8:   cmp al,0d9h
        jnz car9
        mov al,2ch
car9:   cmp al,0dch
        jnz car10
        mov al,2eh
car10:  cmp al,0deh
        jnz car11
        mov al,20h
car11:  cmp al,0e0h
        jnz car12
        mov al,31h
car12:  cmp al,0e3h
        jnz car13
        mov al,32h
car13:  cmp al,0ebh
        jnz car14
        mov al,33h
car14:  cmp al,0edh
        jnz car15
        mov al,34h
car15:  cmp al,0efh
        jnz car16
        mov al,35h
car16:  cmp al,0e8h
        jnz car17
        mov al,36h
car17:  cmp al,0e9h
        jnz car18
        mov al,37h
car18:  cmp al,0eah
        jnz car19
        mov al,38h
car19:  cmp al,0ech
        jnz car20
        mov al,39h
car20:  cmp al,0eeh
        jnz car21
        mov al,30h
car21:  cmp al,0d8h
        jnz car22
        mov al,41h
```

```
car22:  cmp al,0c9h
        jnz car23
        mov al,42h
car23:  cmp al,0cah
        jnz car24
        mov al,43h
car24:  cmp al,0dbh
        jnz car25
        mov al,44h
car25:  cmp al,0ddh
        jnz car26
        mov al,45h
car26:  cmp al,0dfh
        jnz car27
        mov al,46h
car27:  cmp al,0c8h
        jnz car28
        mov al,47h
car28:  cmp al,0cbh
        jnz car29
        mov al,48h
car29:  cmp al,0cch
        jnz car30
        mov al,49h
car30:  cmp al,0cdh
        jnz car31
        mov al,4ah
car31:  cmp al,0ceh
        jnz car32
        mov al,4bh
car32:  cmp al,0c1h
        jnz car33
        mov al,4ch
car33:  cmp al,0c2h
        jnz car34
        mov al,4dh
car34:  cmp al,0c5h
        jnz car35
        mov al,4eh
car35:  cmp al,0c6h
        jnz car36
        mov al,4fh
car36:  cmp al,0cfh
        jnz car37
        mov al,50h
car37:  cmp al,0c0h
        jnz car38
        mov al,51h
car38:  cmp al,0c3h
        jnz car39
        mov al,52h
car39:  cmp al,0c4h
        jnz car40
        mov al,53h
car40:  cmp al,0c7h
        jnz car41
        mov al,54h
```

```
car41:  cmp al,0e7h
        jnz car42
        mov al,55h
car42:  cmp al,0e1h
        jnz car43
        mov al,56h
car43:  cmp al,0e2h
        jnz car44
        mov al,57h
car44:  cmp al,0e5h
        jnz car45
        mov al,58h
car45:  cmp al,0e4h
        jnz car46
        mov al,59h
car46:  cmp al,0e6h
        jnz car47
        mov al,5ah
car47:  iret
endp
ft1     proc near       ;Subrutina para capturar una tecla.
        mov al,0
ptcl:   test al,0ffh
        jz ptcl
        ret
endp
fd0     proc near       ;Subrutina para inicializar display
        mov si,0004h
        mov cx,0500h    ;Lee los primeros 5 datos de la tabla de parámetros del display.
cont1:  mov al,[si]
        call palcntr
        inc si
        loop cont1
        ret
endp
finfunc: mov ax,0       ;Inicio del Sistema Operativo.
        mov es,ax       ;Localiza el segmento extra a partir de la localidad 0.
        mov ds,ax       ;Localiza el segmento de datos a partir de la localidad 0.
        mov ax,0c004h
        mov ss,ax       ;Localiza el segmento de pila a partir de la localidad 4C0h.
        mov sp,0c000h   ;Ubica el puntero de pila en C0h
        mov ax,0
        mov al,3eh      ;Inicializa el reloj del 8279.
        out 12h,al
        call retardo
        mov al,00h      ;Programa el modo del 8279.
        out 12h,al
        call retardo
        mov al,50h      ;Lee el estado de la FIFO y el teclado.
        out 12h,al
        mov al,81h      ;Inicializa el 8255 en modo 0.
        out 0eh,al
        call fd0        ;Inicializa el display LCD 44780.
        mov cl,0Fh      ;Envia mensaje "SDI-86 Ver. 1.0"
cont2:  cmp cl,07h      ;Compara si no se pasa de la mitad de la pantalla.
        jnz NoCP1
        mov al,0c0h     ;Direcciona la DDRAM con 40h
```

```
        call palcntr
NoCP1:  mov al,[si]
        call mcar       ;Muestra caracter a caracter el mensaje.
        inc si
        loop cont2
        mov al,'-'      ;Muestra el prompt del sistema.
        call mcar
TclS:   call ft1        ;Continua si se presiona 'S'.
        cmp al,'S'
        jnz TclS
ComSO:  call fd0        ;Borra la pantalla.
        mov al,'-'      ;Coloca el prompt del sistema.
        call mcar
        call ft1        ;Espera por un caracter de control del sistema operativo.
        mov dl,al
        call mcar       ;Muestra el caracter pulsado.
        mov al,dl
        cmp al,01h      ;Verifica si se presionó "Do".
        jnz NotclDo
        call fd0        ;Borra pantalla.
        mov cx,0c00h    ;Presenta mensaje "Copiando RAM".
        mov si,6504h
sigcar4: cmp cl,04h
        jnz NoCP3
        mov al,0c0h     ;Direcciona la DDRAM con 40h.
        call palcntr
NoCP3:  mov al,[si]
        call mcar
        inc si
        loop sigcar4
        mov cx,000ch    ;Realiza copia de RAM en ROM a RAM.
        mov di,0010h    ;Direcciona inicio de Copia de RAM en ROM.
        mov si,0040h    ;Direcciona inicio de RAM.
mram:   mov ah,[di]     ;Copia localidades de memoria.
        mov [si],ah
        inc si          ;Incrementa localidades de memoria.
        inc di
        loop mram
        call fd0        ;Borra pantalla.
        mov cx,0800h    ;Presenta mensaje "Programa"
        mov si,5D04h
sigcar3: mov al,[si]
        call mcar
        inc si
        loop sigcar3
        mov ax,0004h    ;Relocaliza segmento de datos en 400h
        mov ds,ax
        mov ax,0
        call retardo
        mov ax,0        ;Relocaliza segmento de datos en 0.
        mov ds,ax
NotclDo: cmp al,'H'     ;Verifica si se presionó 'H'
        jnz NoH
        call fd0        ;Borra la pantalla.
        mov dx,0
        mov cx,4600h    ;Muestra mensajes de ayuda del sistema operativo.
        mov si,1404h
```

```
        sigcar1: cmp dl,08h
                 jnz NoCP2
                 mov al,0c0h        ;Direcciona la DDRAM con 40h.
                 call palcntr
        NoCP2:   mov al,[di]
                 call mcar
                 inc dl
                 cmp dh,0eh
                 jz Stop
                 cmp dh,19h
                 jz Stop
                 cmp dh,22h
                 jz Stop
                 cmp dh,2fh
                 jz Stop
                 cmp dh,3ch
                 jnz NoStop
        Stop:    mov dl,00h         ;Espera hasta que se presione C.
                 call fl1
                 cmp al,04h
                 jnz Stop
                 push cx
                 call fd0           ;Borra pantalla.
                 pop cx
        NoStop:  inc dh
                 inc di
                 loop sigcar1
        NoH:     cmp al,51h         ;Verifica si se presionó Q.
                 jnz NotclEx
                 call fd0           ;Borra pantalla.
                 mov cx,0300h       ;Muestra mensaje "END"
                 mov si,5A04h
        sigcar2: mov al,[si]
                 call mcar
                 inc si
                 loop sigcar2
        fin:     jmp fin            ;El programa se detiene
        NotclEx:jmp ComSO           ;Continua a pedir una nueva instrucción.
        code     ends
        main endp
end     main
```

ANEXO 4

Código fuente de la aplicación "JUEGO DOBLE CON MICROPROCESADOR 8086"

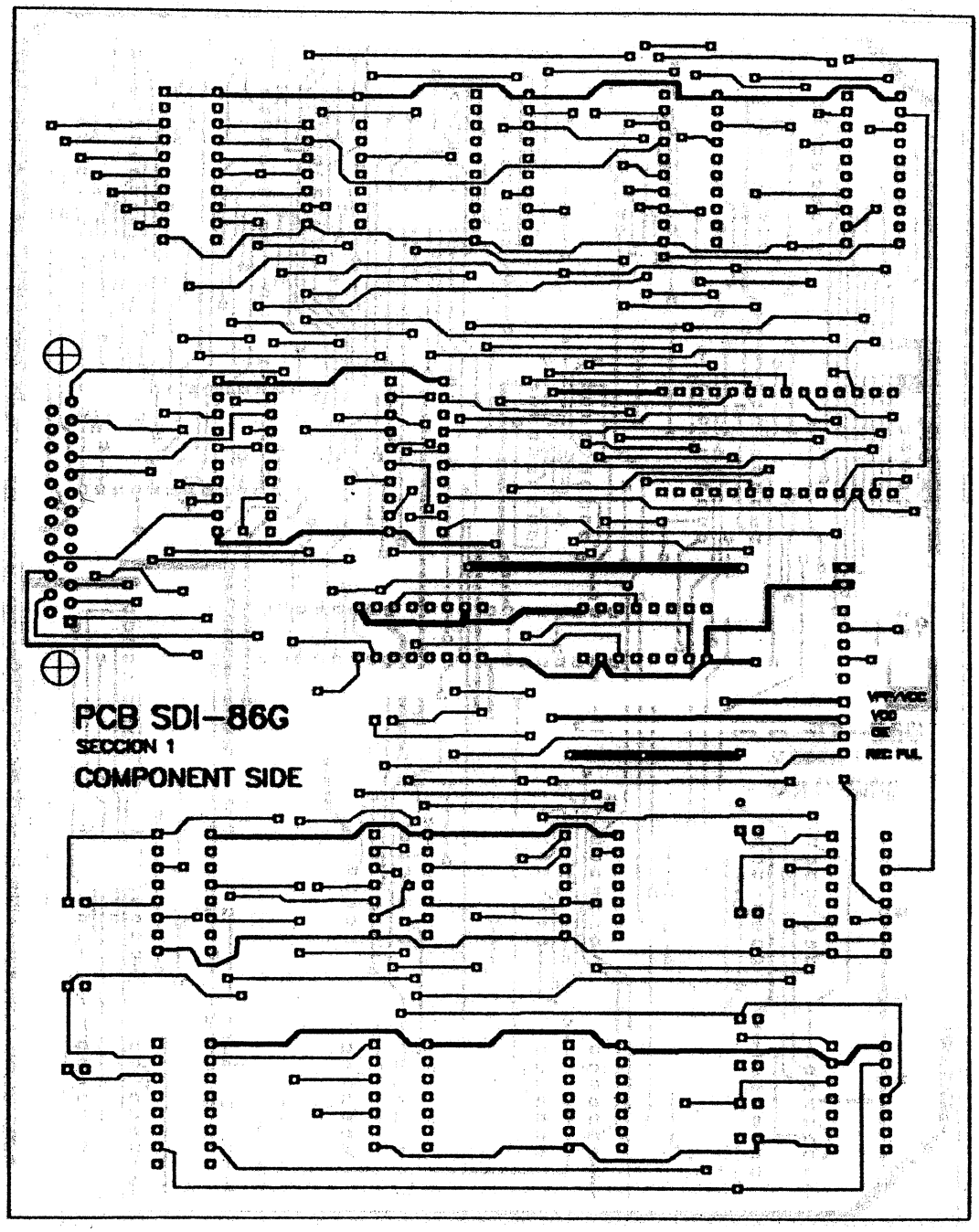
```
stac    segment stack
        dw 1024 dup (?)
stac ends
datos segment
m1      db 'PROGRAMA:<1> JUEGO 1<2> JUEGO 2<S> SALIRELIJA OPCION$'
m2      db 'Adivine palabra$'
m3      db 'Otro jugador$'
m4      db 'Elija secuencia$'
m5      db '7 para terminar$'
m6      db 'Fin de secuencia$'
wrd     db 'POLITECNICA$'
raya    db 5fh
nr      db 0
ni      db 0
nl      db 0
d0s1    db 81h
d1s1    db 42h
d2s1    db 24h
d3s1    db 18h
d4s1    db 3ch
d5s1    db 7eh
d6s1    db 0ffh
d7s1    db 0e7h
d0s2    db 18h
d1s2    db 24h
d2s2    db 42h
d3s2    db 81h
d4s2    db 18h
d5s2    db 24h
d6s2    db 42h
d7s2    db 81h
d0s3    db 81h
d1s3    db 42h
d2s3    db 24h
d3s3    db 18h
d4s3    db 81h
d5s3    db 42h
d6s3    db 24h
d7s3    db 18h
d0s4    db 7fh
d1s4    db 3fh
d2s4    db 1fh
d3s4    db 0fh
d4s4    db 07h
d5s4    db 03h
d6s4    db 01h
d7s4    db 00h
datos   ends
```

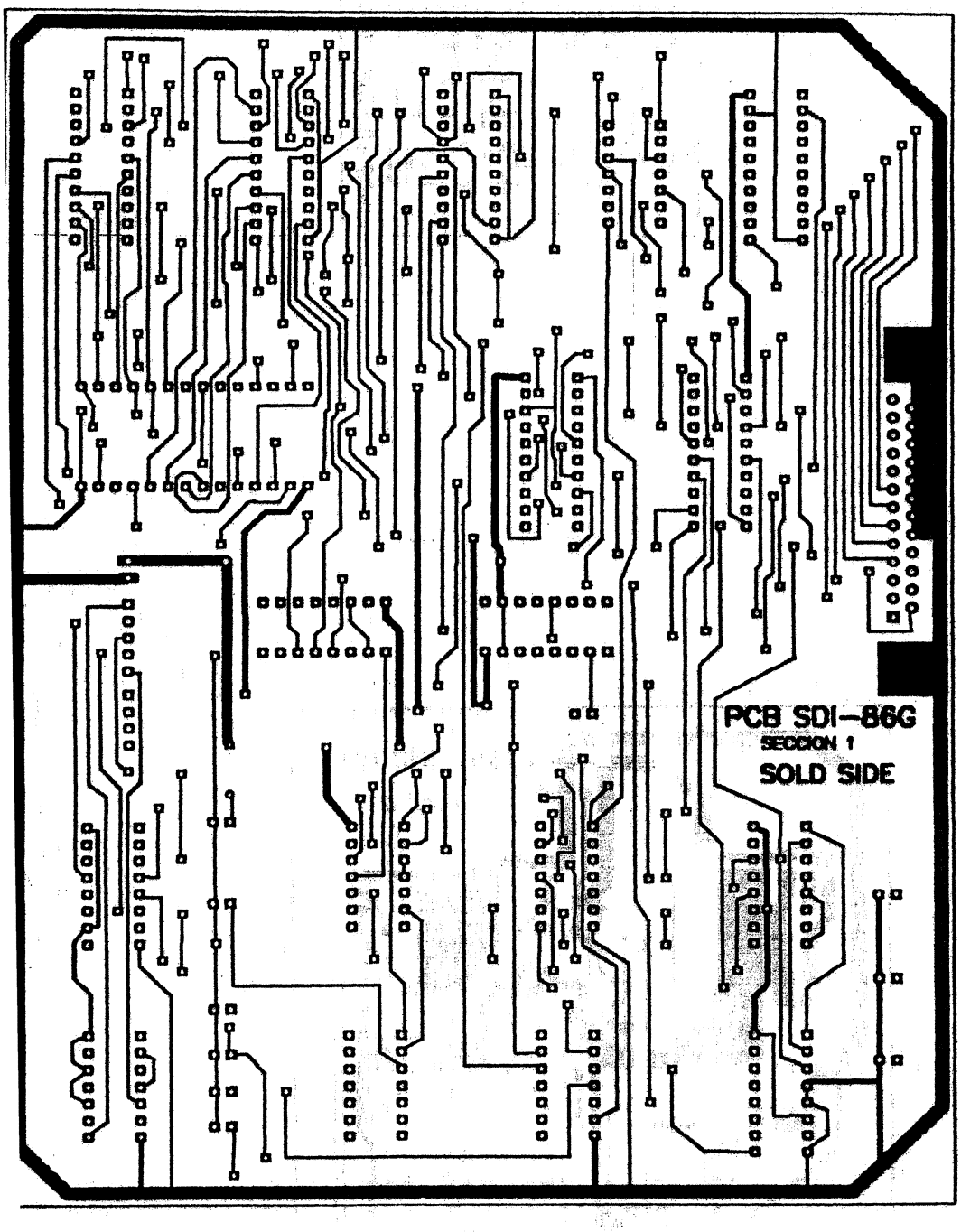
```
code segment
        assume cs:code,ds:datos,ss:stac
        main    proc far
                mov ax,datos
                mov ds,ax
                call dspfnc0        ;Borra la pantalla
                mov cx,52           ;Presenta información sobre el funcionamiento del programa
                mov si,offset m1
        sigcar1: mov al,[si]
                mov car,al
                call dspfnc1
                inc col
                cmp si,8
                jz Stop1            ;Para luego del mensaje "PROGRAMA:"
                cmp si,19
                jz Stop1            ;Para luego del mensaje "<1> JUEGO 1"
                cmp si,30
                jz Stop1            ;Para luego del mensaje "<2> JUEGO 2"
                cmp si,39
                jnz NoStop          ;Para luego del mensaje "<S> SALIR"
        Stop1:  call tclfnc0
                cmp al,'C'          ;Continua si se presiona la letra C
                jnz Stop1           ;Para luego del mensaje "ELIJA OPCION:"
                call dspfnc0
        NoStop: inc si
                loop sigcar1
        g1:     call tclfnc0
                cmp al,'1'          ;Compara si se presionó 1
                jnz g2              ;Si no se presionó 1 salta a g2
                call dspfnc0
                mov cx,15
                mov si,offset m2
                call premen
        Stop2:  call tclfnc0
                cmp al,'C'
                jnz Stop2
                call game1          ;Ejecuta juego 1
                call dspfnc0
                mov cx,12           ;Presenta mensaje "Adivine palabra"
                mov si,offset m3
                call premen
        g2:     cmp al,'2'          ;Compara si se presionó 2
                jnz nogame
                call dspfnc0
                mov cx,15           ;Presenta mensaje "Elija secuencia"
                mov si,offset m4
                call premen
        Stop3:  call tclfnc0
                cmp al,'C'          ;Continua si se presiona la letra C
                jnz Stop3
                call dspfnc0
                mov cx,15
                mov si,offset m5    ;Presenta mensaje "7 para terminar"
                call premen
                call game2          ;Ejecuta juego 2
                call dspfnc0
                mov cx,16
```

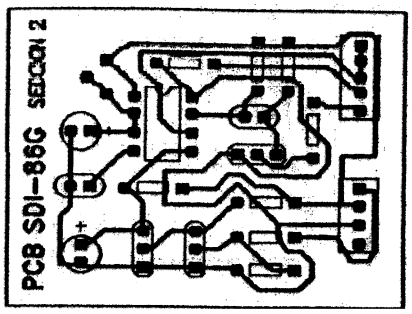
```
                mov si,offset m6  ;Presenta mensaje "Fin de secuencia"
                call premen
        nogame: cmp al,'S'
                jnz g1
        mov ah,4ch          ;salir al DOS
        int 21h
        game1   proc near;Juego de adivinar palabra
        call dspfnc0
        mov nr,0bh          ;Carga nr con el número de '_'
        mov col,03h
repet:  mov car,'_'
        call dspfnc1
        inc col             ;Incrementa en 1 la posición del próximo caracter '_'
        dec nr              ;Decrementa el número de '_' a imprimir
        jne repet           ;Repite mientras hayan más '_' a imprimir
        mov col,0fh         ;Carga la posición donde se imprimirá el 9
        mov car,'9'
        call dspfnc1
        mov ni,09h          ;Carga el número de intentos en ni
denuevo:mov col,00h         ;Carga en col la posión inicial
        call leercar        ;Llama a la subrutina para leer caracter
        mov col,03h         ;Carga en col la posición de inicio de impresión de la palabra
        mov nl,0bh          ;Carga en nl el número de letras de la palabra
        mov si,offset wrd ;Carga la dirección del caracter inicial de la palabra
        mov al,car
comp:   cmp al,[si]         ;Compara el caracter ingresado con cada caracter de la palabra
        jne salto           ;Si son diferentes continua en salto
        call dspfnc1
salto:  inc col             ;Incrementa la posición de impresión
        inc si              ;Incrementa la dirección del próximo caracter de la palabra
        dec nl              ;Decrementa el número de letras de la palabra
        jne comp            ;Continua en comp hasta que nl sea cero
        add ni,2fh          ;Convierte el número de intentos a su valor ASCII menos 1
        mov col,0fh         ;Carga en col la posición 0fh
        mov al,ni
        mov car,al
        call dspfnc1
        sub ni,2fh          ;Regresa ni a su valor original
        dec ni              ;Decrementa en 1 el número de intentos
        jne denuevo         ;Continua en denuevo mientras el número de intentos no es cero
                ret
        endp
        game2   proc near;Juego de luces programable
repsec:
        call InSDI
        cmp al,07h          ;Compara si se ingresado 7 por el puerto de entrada
        jz Stop
        and al,03h
        cmp al,00h
        jnz salt1
        mov si,offset d0s1;Selecciona secuencia 1 si se ingresó 0
salt1:  cmp al,01h
        jnz salt2
        mov si,offset d0s2;Selecciona secuencia 1 si se ingresó 1
salt2:  cmp al,02h
        jnz salt3
        mov si,offset d0s3;Selecciona secuencia 1 si se ingresó 2
```

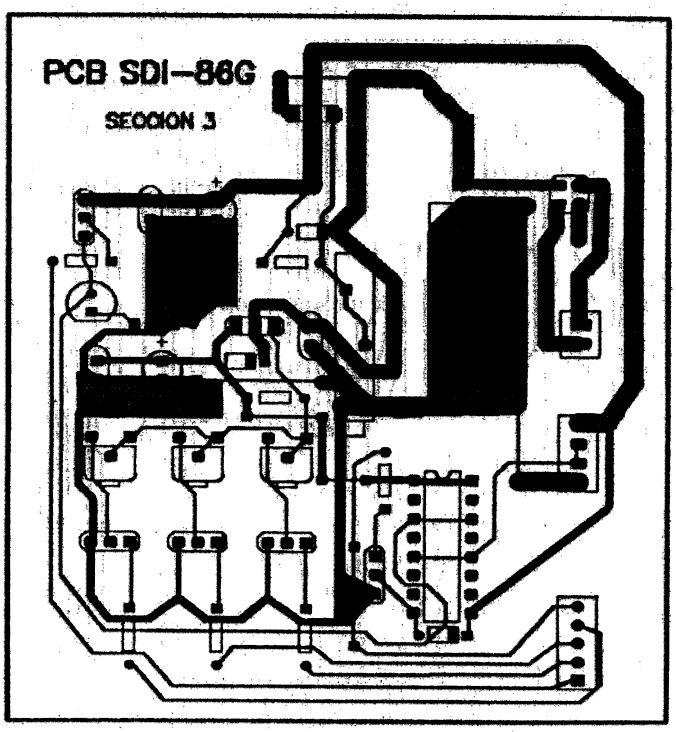
```
salt3:   cmp al,03h
         jnz salt4
         mov si,offset d0s4;Selecciona secuencia 1 si se ingresó 3
salt4:   mov cx,08h
ndat:    mov al,[si]
         call OutSDI        ;Muestra secuencia seleccionada
         call retcomp
         inc si
         loop ndat
         jmp repsec
Stop:    ret
         endp
premen   proc near         ;Subrutina para presentar mensaje
         mov al,[si]
         mov car,al
         call dspfnc1
         inc col
         inc si
         loop premen
         ret
premen   endp
leercar  proc near         ;Subrutina para presentar leer caracter
         call tclfnc0
         mov col,00h
         mov car,al
         call dspfnc1
         ret
leercar endp
retcomp proc near;Subrutina de retardo
         mov di, 40
newret:  mov bx,0ffffh
ret1:    dec bx
         jnz ret1
         dec di
         jnz newret
         ret
retcomp endp
         code     ends
         main endp
end    main
```

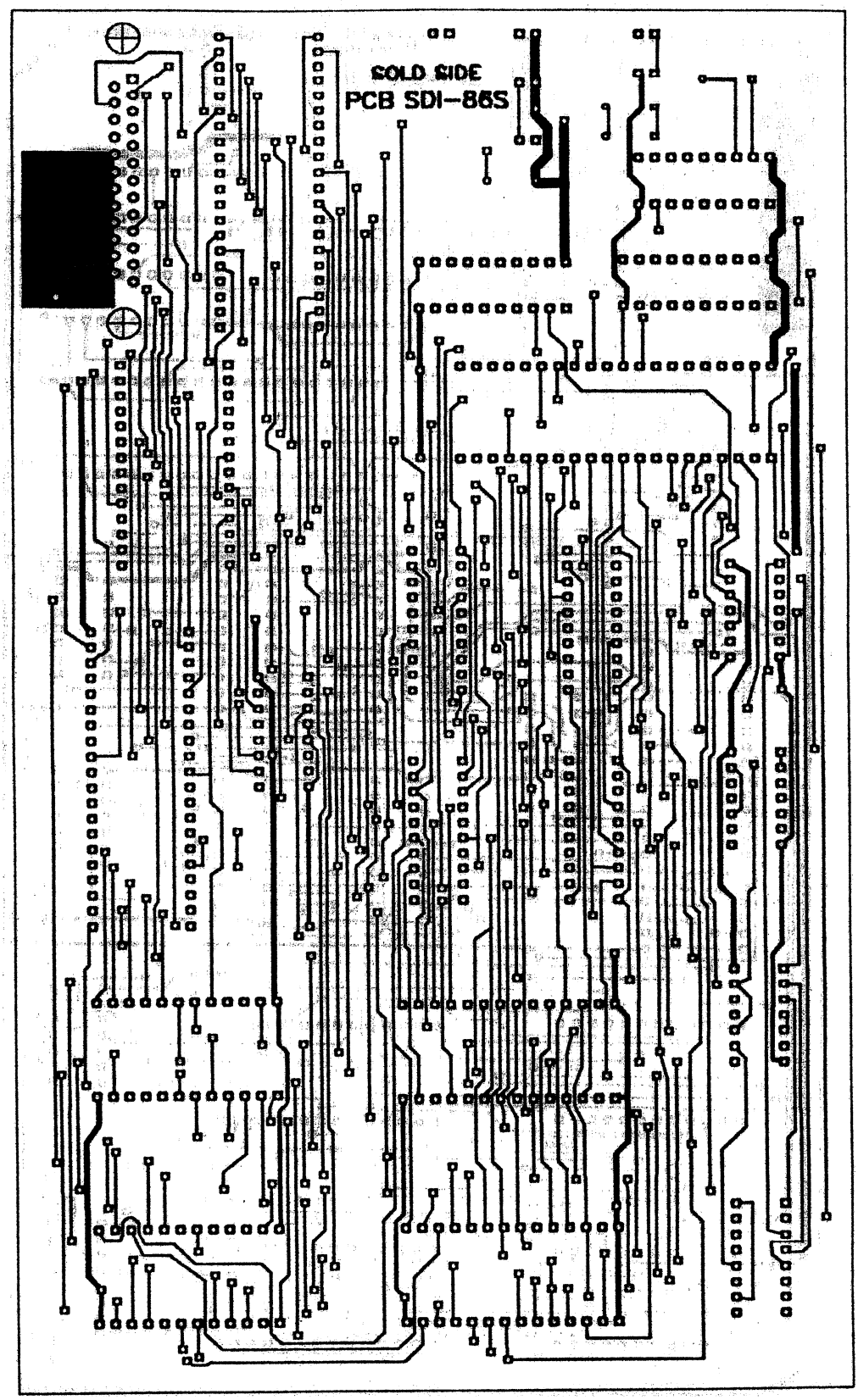
ANEXO 5
CIRCUITOS IMPRESOS DEL SDI - 86 G Y DEL SDI - 86S

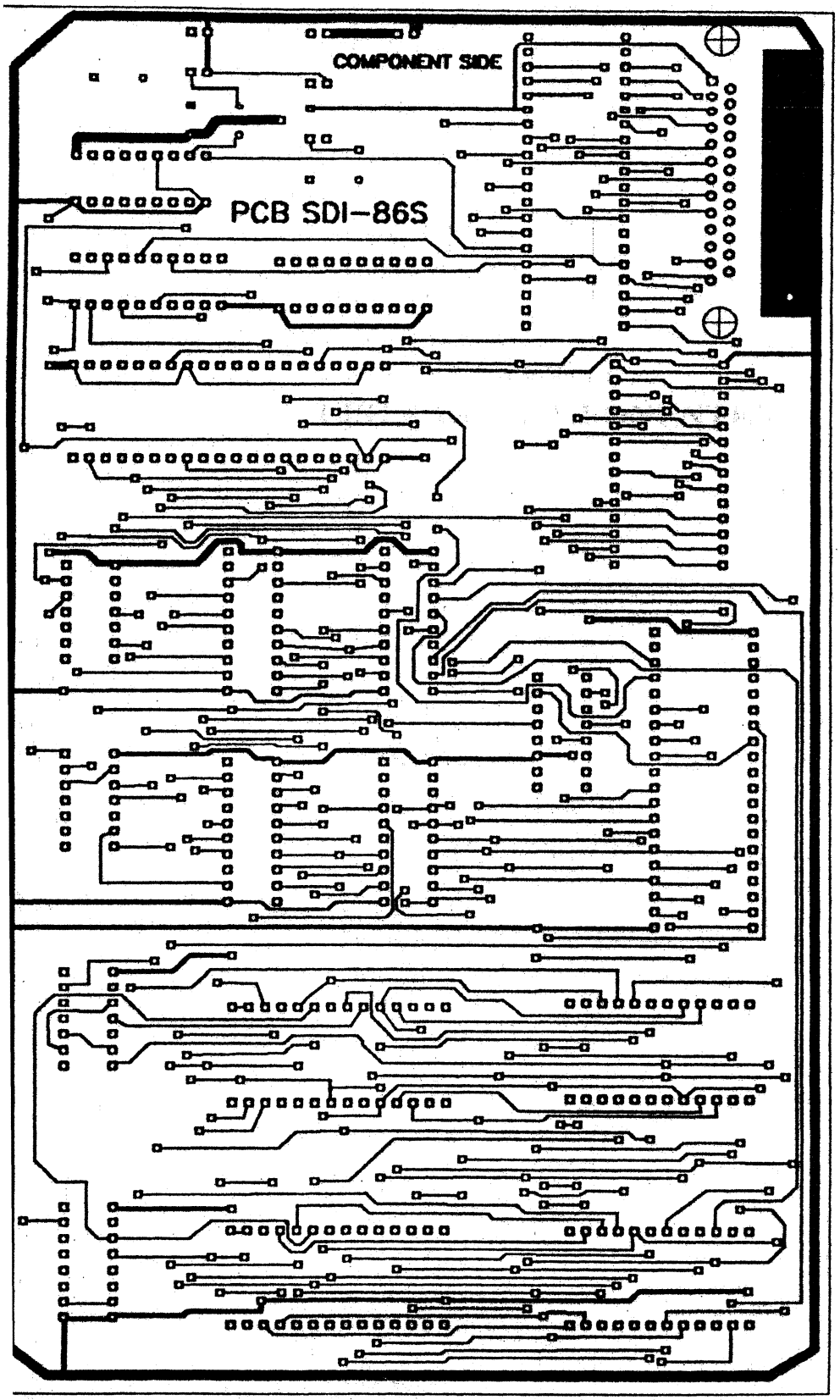

PCB SDI-86G
SECCION 1
COMPONENT SIDE


PCB SDI-86G
SECCION 1
SOLD SIDE


PCB SDI-86G SECCION 2


PCB SDI-86G
SECCION 3


SOLD SIDE
PCB SDI-865


COMPONENT SIDE
PCB SDI-86S

ANEXO 6
LISTA DE COMPONENTES

LISTA DE COMPONENTES PARA EL SDI-86G

CANTIDAD	*DESCRIPCION*
2	74LS245
3	74LS374
1	74LS244
1	74LS30
1	74LS32
1	74LS08
2	74LS04
1	74LS155
2	74LS161
2	74LS138
1	74LS74
2	LM317
1	LM555
7	Trans. NPN 2N3904
1	Trans. PNP 2N3906
1	Puente Rectificador
3	diodos 1N4007
2	les pequeños rojo y verde
1	Rele de 5V 2P2C
4	Pot. Precisión 10K
1	Pot. Precisión 5k
15	Cap. Cerámicos de 100nF
1	Cap. Elec. de 2200uF, 50V
2	Cap. Elec. de 100uF, 25V
1	Cap. Elec. de 10uF, 10V
1	Cap. Elec. de 1uF, 10V
6	Resistores de 10K ½W
5	Resistores de 220Ω ½W
4	Resistores de 1K ½W
1	Resistor de 45K ½W
1	Resistor de 22K ½W
1	Conector DB-25H
1	Transformador de 2A 125/30V
1	Disipador grande
1	Cable con conectores DB-25M
3	Tarjetas de Circuito Impreso
1	Caja metálica

LISTA DE COMPONENTES PARA EL SDI-86S

CANTIDAD	*DESCRIPCION*
1	Microprocesador INTEL 8086
1	8284
1	8255
1	8279
1	74LS244
3	74LS373
2	74LS245
1	74LS32
2	74LS00
1	74LS161
1	74LS138
2	RAM 6116 (2KB)
2	EPROM 2764 (8KB)
1	Teclado matricial 8x6
1	Display alfanumérico LCD
1	Conector DB25H
1	Cristal de 15mhz
2	Resistores de 1K
1	Diodo 1N4007
2	Cap. Elect. De 10uF, 10V
1	Led rojo pequeño
1	Botonera antirebote
1	Interruptor
2	Sockets de 28 pines
1	Tarjeta de circuito impreso
1	Fuente de 5 Vdc
1	Caja metálica

BIBLIOGRAFIA

PETER ABEL, Lenguaje Ensamblador y Programación para IBM PC y Compatibles, Tercera Edición, 1996. PRENTICE HALL HISPANOAMERICANA S.A.

BARRY B. BREY, Los Microprocesadores INTEL 8086-80486, Tercera Edición, 1995. PRENTICE HALL HISPANOAMERICANA S.A.