CENTRO DE INVESTIGACION Y DE ESTUDIOS AVANZADOS

DEL

INSTITUTO POLITECNICO NACIONAL

DEPARTAMENTO DE INGENIERIA ELECTRICA

SECCION DE COMPUTACION

SISTEMA PARA DIAGNOSTICO Y DESARROLLO DE APLICACIONES CON MICROPROCESADORES BASADO EN COMPUTADORAS PERSONALES

Tesis que presenta el Ing. Armando Jimenez Flores para obtener el grado de **MAESTRO EN CIENCIAS** en la especialidad de **INGENIERIA ELECTRICA**. Trabajo dirigido por el Dr. Armando Maldonado Talamantes y el Dr. Manuel E. Guzman.

Becario del CONACYT

México D.F., Agosto de 1988

Dedico con cariño esta tesis

A mis padres

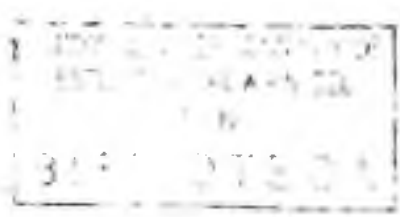
A mi esposa

A mis hermanos

A mis abuelitos

A mis tíos

Y especialmente a mis hijas Angélica y Diana Paola, por los momentos de diversión y convivencia que les robé durante el desarrollo del presente trabajo.



Mi especial agradecimiento:

Al Dr. Armando Maldonado Talamantes, por su valiosa labor de asesoria y por su firme apoyo, como Jefe del Area de Sistemas Digitales del Departamento de Electronica de la Universidad Autonoma Metropolitana, durante el desarrollo de este trabajo.

Al Dr. Manuel E. Guzmán Renteria, por su estimable labor de asesoramiento y por su gran apoyo, como Jefe de la Seccion de Computacion del Departamento de Ingenieria del Centro de Investigacion y de Estudios Avanzados, durante la realizacion de la presente tesis.

Al Dr. José Luis Leyva Montiel, por el tiempo dedicado a la revision de este trabajo.

Al M. en C. Héctor Ruiz Barradas, por la gran ayuda y paciencia que me brindo durante las etapas de diseño y pruebas.

Al Centro de Investigacion y de Estudios Avanzados del Instituto Politécnico Nacional y a la Universidad Autonoma Metropolitana, por todos los beneficios recibidos y las facilidades concedidas.

Al Consejo Nacional de Ciencia y Tecnologia, por la beca que me otorgo para sustentar mis estudios de maestria.

A mis amigos, por el apoyo moral que recibí de ellos.

A todas las personas que de alguna forma colaboraron en la realización de este trabajo.

I N D I C E

RESUMEN

El presente trabajo consiste en el diseño y construcción de un **Sistema para Diagnóstico y Desarrollo de Aplicaciones con Microprocesadores basado en Computadoras Personales**, el cual se divide funcionalmente en dos partes fundamentales:

1.- Un **Grabador Universal de Memorias EPROM.**

2.- Un **Emulador de Sistemas Digitales.**

Los componentes del grabador son: un módulo digital basado en un microprocesador Z-80, el cual se puede comunicar con cualquier computadora personal a través de un puerto serie RS-232; y un modulo de programación, escrito en lenguaje C, que permite aprovechar los recursos "software" de la computadora personal. El grabador puede leer y programar memorias EPROM de los tipos 2716, 2732, 2732A, 2764, 27128, y 27256. Además de las funciones de lectura y escritura, el grabador puede: comparar el contenido de un archivo contra el de una memoria, separar un archivo hexadecimal en sus partes par e impar, y desplegar y modificar archivos bajo el formato Intel. Adicionalmente, el grabador puede copiar directamente, de memoria fuente a memoria destino, sin hacer uso de la computadora.

El emulador de sistemas digitales es propiamente un **emulador para los microprocesadores 8086 y 8088**. Esta constituido por un módulo digital basado en dichos microprocesadores, y de un módulo de programación que aprovecha los recursos "software" de la computadora. De igual forma que el grabador, se comunica con la computadora mediante un puerto serie, y además utiliza un cable de 40 hilos para conectarse al prototipo del usuario. El emulador permite lanzar la ejecución de un programa de manera interrumpida (paso a paso), o ininterrumpida (tiempo real), en el prototipo del usuario. Asimismo, permite cargar archivos en memoria RAM, compartir algunos recursos con el prototipo del usuario, y leer y/o modificar memorias y puertos.

I N T R O D U C C I O N

Con el objeto de promover la generación de tecnología nacional, en el campo de diagnóstico y desarrollo de aplicaciones con microprocesadores, el presente trabajo nace, en 1988, como un **proyecto conjunto** entre la **Universidad Autónoma Metropolitana (Unidad Azcapotzalco)**, y el **Centro de Investigación y de Estudios Avanzados del Instituto Politécnico Nacional**. El trabajo fue realizado en el área de Sistemas Digitales y Computadoras del Departamento de Electrónica de la Universidad, y diseñado bajo las directrices trazadas por los asesores de tesis.

Para alcanzar el objetivo mencionado, se propone el diseño y realización de un ambiente "hardware-software", alrededor de computadoras personales, que proporcione la infraestructura necesaria, y a bajo costo, para desarrollar prototipos basados en microprocesadores y diagnosticar la operación de sistemas digitales.

El trabajo se divide en dos fases principales. La primera consiste en diseñar y construir un **Grabador Universal de Memorias EPROM** que permita leer y grabar las memorias: 2716, 2732, 2732A, 2764, 27128, y 27256, consideradas de uso más común por los diseñadores de sistemas digitales. La segunda fase comprende el diseño y construcción de un **Emulador de Sistemas Digitales**, enfocado a los microprocesadores 8086 y 8088.

Este reporte está constituido por cinco capítulos y cinco apéndices. El capítulo I describe la importancia que tienen actualmente los sistemas de desarrollo para los diseñadores de sistemas digitales con microprocesadores, y las ventajas que ofrece el sistema de desarrollo que se presenta en este trabajo de tesis. El capítulo II se dedica al Grabador de Memorias EPROM, y en él se describe su principio de funcionamiento, caracterización, y diseño. En el capítulo III se describe al Emulador de Sistemas Digitales. En este capítulo se muestra el principio de funcionamiento, la caracterización, y el diseño, tanto de la circuitería como de la programación del

emulador. En el capitulo IV se describen las pruebas realizadas al grabador de memorias y al emulador. Asimismo, se indica cuales han sido los resultados obtenidos. El capitulo V está dedicado a las conclusiones del trabajo. Los apéndices A y B muestran los diagramas eléctricos del grabador y del emulador, respectivamente. El apéndice C contiene la codificacion, en lenguaje ensamblador, del programa monitor del grabador de memorias. En el apéndice D se muestra la codificacion, en lenguage PLM-86, del programa monitor del emulador. Finalmente, el apéndice E muestra el codigo fuente, escrito en lenguaje C, correspondiente al módulo de interaccion con el usuario del emulador.

C A P I T U L O I

LOS SISTEMAS DE DESARROLLO

Durante mucho tiempo, principalmente en la década pasada, el desarrollo de "hardware" y "software" para productos basados en microprocesadores fué acompañado por dos esfuerzos claramente separados; el del diseñador de hardware y el de software. Esto trajo como consecuencia un gran desgaste de esfuerzos y pérdidas de tiempo debido a las inconsistencias entre las especificaciones del "hardware" y el diseño del "software", a nivel de prototipo.

Actualmente, para los diseñadores de sistemas digitales basados en microprocesadores son ya insuficientes las herramientas convencionales tales como: osciloscopios, analizadores de estados lógicos, y otras. Usualmente el diseñador de sistemas digitales utiliza una técnica de realización apoyada en la prueba-error-corrección, mediante la cual realiza una serie de pruebas hasta alcanzar la mejor solución. Este es un proceso que por su naturaleza iterativa, normalmente consume la mayor parte del tiempo de desarrollo del prototipo. Para acelerar dicho proceso, diversas compañías fabricantes de microprocesadores han desarrollado herramientas hardware/software comunmente conocidas como sistemas de desarrollo para microprocesadores. Tales herramientas, permiten la realizacion simultánea tanto de la circuiteria como de la programación, y además coadyuvan a depurar el producto final.

Con los sistemas de desarrollo que se ofrecen actualmente, el diseñador de "software", que puede ser el mismo que diseña el "hardware", trabaja con la circuiteria del prototipo conforme ésta se vaya diseñando. Por otra parte, el diseñador del "hardware" puede construir la circuiteria, trabajando con la programacion disponible, y facilitando el dignóstico conforme el desarrollo del "hardware" vaya progresando. Con estos productos, el diseñador de sistemas digitales dispone de una poderosa herramienta de diseño para soportar desarrollos altamente especializados, basados en microprocesadores.

Es evidente que estos productos satisfacen cabalmente las necesidades de todo diseñador de sistemas digitales con microprocesadores, debido a sus capacidades de diagnostico tales como: desplegar y alterar los registros del microprocesador, leer y modificar el contenido de memorias y puertos, simular una memoria de tipo ROM con una memoria facilmente alterable como la RAM, y detener la ejecución de un programa en puntos predefinidos para desplegar y alterar el estado del sistema. Sin embargo los costos de estas herramientas son altos debido al bajo volumen de venta, y consecuentemente estan fuera del alcance de muchos usuarios potenciales.

El grado de evolución en el desarrollo de tecnologias con semiconductores ha permitido disponer actualmente de microcircuitos con muy alta densidad de componentes integrados. Tal es el caso de los microprocesadores, con cuyo advenimiento se ha generalizado el uso de las computadoras personales en muy diversos campos, haciendo atractiva la idea de usar una de ellas como parte de un sistema de desarrollo para microprocesadores, dandole asi una capacidad adicional de servicio, ya que no se requiere dedicación exclusiva.

Tomando la idea anterior, se estableció como meta para este trabajo, **agregar a una computadora personal las capacidades de un sistema de desarrollo de sistemas digitales**, con las ventajas que ofrecen los productos comerciales pero a un costo verdaderamente accesible para muchos usuarios potenciales que dispongan de una computadora personal.

En la figura 1.1 se muestra el esquema conceptual de la solución que se propone, a la cual se le denomina como Sistema para Diagnostico y Desarrollo de Sistemas Digitales Basados en Microprocesadores.

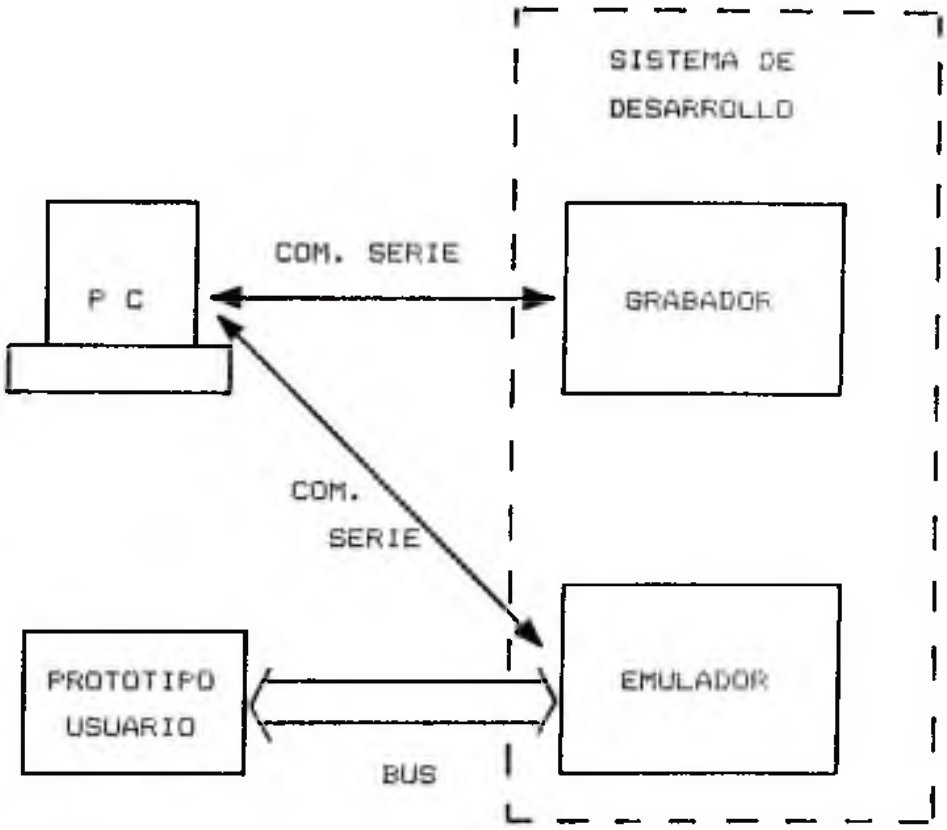


FIG. 1.1 ESQUEMA CONCEPTUAL DEL SISTEMA DE DESARROLLO

El sistema propuesto consta de un **Emulador de Sistemas Digitales** orientado a los microprocesadores 8086 y 8088, y un **Grabador Universal de Memorias EPROM** .El grabador de memorias se incluye en el sistema propuesto, con el objeto de hacerlo más completo y poderoso.

El Emulador de Sistemas Digitales

El Emulador de Sistemas Digitales es propiamente un Emulador para los Microprocesadores 8086 y 8088. Debe tener las características deseables por cualquier diseñador de sistemas digitales, tales como: ejecución interrumpida (paso a paso), ejecución ininterrumpida (tiempo real), lectura y modificacion de

los registros del procesador, lectura y modificacion de la memoria y puertos, carga de archivos en memoria RAM, y salvaguarda de los mismos. Adicionalmente se propone que el emulador pueda compartir, de manera opcional, algunos recursos con el prototipo del usuario.

El Grabador Universal de Memorias EPROM

Un grabador de memorias de lectura exclusiva (EPROM), representa una herramienta de valiosa utilidad para los diseñadores de sistemas digitales, por lo que se puede decir que el grabador de memorias es al diseñador de sistemas digitales como el cautin al técnico en electrónica. Ambas herramientas son indispensables para cada uno, y ahorran tanto tiempo como versátiles y prácticas lo sean en su operación.

Tradicionalmente se han ofrecido en el mercado dos tipos de grabadores: los que están integrados a un sistema de desarrollo y ofrecen características ventajosas, pero a muy alto costo; y aquellos cuyo objetivo es que sean economicamente accesibles, pero que sus limitaciones inherentes a su bajo costo los hacen impotentes o poco prácticos para muchas aplicaciones que hacen uso de cierta programación.

El grabador que se propone como parte del Sistema de Desarrollo, debe reunir en lo posible las ventajas de aquellos que estan integrados a sistemas comercializados, eliminando sus desventajas en el costo. Para ello se establecen los siguientes requerimientos: Versatilidad para leer y programar las memorias EPROM de los tipos 2716, 2732, 2732A, 27c64, 27128, 27256; Habilidad para realizar copiados directos de memoria fuente a memoria destino, sin requerir de una computadora para realizar esta función; Factibilidad de conexión con cualquier computadora personal o sistema con puerto serie, sin importar su sistema operativo, ni el tipo de procesador que maneje; Portabilidad y bajo costo con respecto a los grabadores comerciales.

C A P I T U L O I I

DISEÑO DE UN GRABADOR UNIVERSAL DE MEMORIAS EPROM

El objetivo planteado en esta parte es: diseñar un grabador de memorias que reuna en lo posible las ventajas de los casos mencionados en el capitulo anterior, eliminando sus desventajas. Para ello se establecen los siguientes requerimientos:

Versatilidad para programar adecuadamente las memorias EPROM: 2716, 2732, 27324, 2764, 27128, y 27256, sin riesgo de dañarlas o dañar el grabador con memorias dañadas.

Habilidad para realizar copiados directos de memoria fuente a memoria destino, sin requerir de una computadora para realizar esta función, y especificando mediante un interruptor el tipo de memoria a copiar.

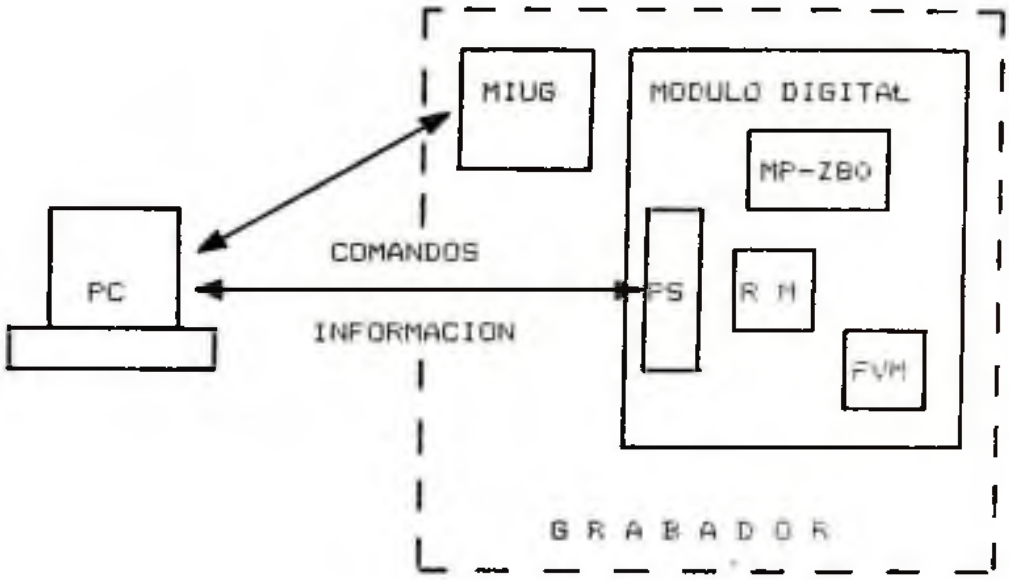
Factibilidad de conexión con cualquier computadora personal o sistema con puerto serie, sin importar su sistema operativo ni el tipo de procesador que maneje.

Portabilidad y bajo costo con respecto a los grabadores comerciales.

PRINCIPIO DE FUNCIONAMIENTO DEL GRABADOR

En la figura 2.1 se muestra el diseño conceptual del grabador. Unicamente cuando el grabador opera en forma autónoma (copiado directo), el esquema conceptual mostrado no tiene validez; ya que, en este caso no se requiere la computadora.

El módulo de interaccion con el usuario del grabador (MIUG) se comunica con el módulo digital, a traves de la computadora, bajo un esquema maestro-esclavo. El modulo de interaccion genera órdenes al módulo digital por medio de comandos o funciones. El módulo digital sigue las órdenes (grabar o leer alguna memoria en particular), y envia al módulo de interaccion la informacion necesaria para que éste la maneje y despliegue.



PC - Computadora Personal
MIUG - Módulo de Interacción con el Usuario del Grabador
PS - Puerto serie (RS232)
MP-Z80 - Microprocesador Z80
RM - Receptáculos para las Memorias
FVM - Fuente de Voltaje Múltiple

FIG. 2.1 ESQUEMA CONCEPTUAL DEL GRABADOR

CARACTERIZACION DEL GRABADOR

El usuario mantiene el control sobre el grabador a través de una relación directa con el programa de la computadora (MIUG), el cual se presenta en forma de menú. De esta forma el usuario le da órdenes de comportamiento al grabador mediante comandos definidos en el menú. Algunos de los comandos que se presentan en el menú no generan órdenes al grabador, debido a que son comandos de despliegue y formateo de información contenida en archivos de disco.

Los comandos que ofrece el grabador son los siguientes:

GRABAR.- Este comando permite grabar memorias EPROM con informacion de archivos que estan bajo un formato Intel. El archivo puede estar en cualquiera de los "drives" definidos en la computadora. Los parámetros que solicita este comando son: tipo de memoria, nombre del archivo a grabar, dirección inicial y final de lo que se desea grabar, y direccion inicial de la memoria EPROM.

LEER.- El comando de lectura permite leer memorias y desplegrar su contenido en la pantalla, o verificar si la memoria esta limpia. Los parametros que solicita son: tipo de memoria a leer, dirección inicial y final de lectura, y opción de lectura o verificacion.

DESPLEGAR.- Este comando no interactúa con el modulo digital. Sólo despliega archivos contenidos en disco, y permite modificarlos. El unico parámetro solicitado es el nombre del archivo.

MODIFICAR.- Con este comando se puede editar una memoria y hacerle modificaciones cuando es posible, o grabar sobre localidades vacias. El parametro solicitado es el tipo de memoria.

COMPARAR.- Este comando permite comparar el contenido de un archivo en disco contra el contenido de una memoria EPROM. Despliega el contenido de las localidades cuando son diferentes a las del archivo. Los parametros solicitados son: tipo de memoria a comparar, nombre del archivo, direccion inicial y final del archivo a

comparar, y dirección inicial en memoria a comparar.

SEPARAR.- Este comando permite separar un archivo hexadecimal, bajo el formato Intel, en dos partes: archivo par, y archivo impar. Dicho comando no interactúa con el módulo digital.

PRINCIPIO DE OPERACION DEL GRABADO

Significado de la Memoria EPROM

Usualmente una memoria **RAM** (random-access read/write memory), en un sistema digital, es utilizada para guardar datos y programas los cuales pueden ser modificados las veces que sea necesario. Desafortunadamente este tipo de memoria es volatil, y cuando se apaga el sistema, los datos se pierden.

En la mayoria de los sitemas digitales existen datos y programas esenciales que deben guardarse en memorias no volátiles llamadas **ROM** (read-only memory). En estos semiconductores (programados por mascarilla), los datos que pueden leerse son determinados durante el proceso de fabricacion. Un dispositivo ROM puede ser accesado aleatoriamente para lectura de la misma forma que la memoria volátil, pero con la ventaja de que los datos permanecen despues de apagar el sistema.

Otro tipo de ROM es la **PROM** (programmable read-only memory), cuyo fabricante la proporciona sin datos para que el usuario decida su contenido, mediante un dispositivo especial de grabación. La memoria PROM presenta las caracteristicas de la memoria grabada por mascarilla (ROM), ademas de que permite una escritura.

La memoria EPROM (erasable programmable read-only memory), **es** un dispositivo que mantiene **un compromiso entre el tipo de memoria PROM**, en la que se escribe una sola vez; **y la memoria volátil o RAM**, en la cual se puede reescribir muchas veces. La memoria EPROM normalmente es encapsulada en un espacio con ventana de cuarzo transparente, y su contenido puede ser borrado mediante exposición a la luz ultravioleta.

Concepto de Grabado de una Memoria EPROM

Grabar una memoria EPROM significa almacenar bits de datos en celdas formadas por transistores FAMOS (floating-gate avalanche-injection metal-oxide semiconductor), de carga almacenada. Estos transistores tienen dos compuertas: la compuerta más baja o flotante esta completamente rodeada por una capa aislante de dioxido de silicio, y la compuerta de arriba, llamada de control o de selección es conectada a la circuiteria externa.

La cantidad de carga electrica almacenada en la compuerta flotante determina cuándo la celda bit contiene un 1 o un 0. Las celdas cargadas son leidas como 0s, y las descargadas se leen como 1s. Las memorias EPROM traen de fábrica, todas las localidades limpias de carga y son leidas como 1s lógicos: cada byte contiene un FF hexadecimal.

Para cambiar una celda bit de 1 a 0, se pasa una corriente a través del canal del transistor, desde fuente hasta drenaje. Al mismo tiempo, se aplica a la compuerta de control una diferencia de potencial relativamente alta, creando un fuerte campo eléctrico dentro de las capas del material semiconductor. Esta es la función de los voltajes de programacion Vpp. En presencia del fuerte campo eléctrico, algunos electrones que pasan por el canal fuente-drenaje adquieren suficiente energia para atravesar la capa aislante que rodea a la compuerta flotante. Conforme los electrones se acumulan en la compuerta flotante, ésta adquiere una carga negativa, que hace que la celda contenga un 0.

Para borrar los datos grabados se expone la memoria a la luz ultravioleta, la cual contiene fotones de relativa alta energía. La incidencia de fotones excita a los electrones que se encuentran en la compuerta flotante, y hace que adquieran estados de energía suficientemente altos para regresar a través de la capa aislante. De esta forma, se remueve la carga de la compuerta , la celda regresa al estado 1.

Procedimiento de Grabación de una Memoria EPROM

El procedimiento para grabar un dato en una memoria EPROM se explica de manera básica mediante las figuras 2.2 y 2.3. Inicialmente, la entrada Vcc de la memoria debe estar conectada a 5 volts. Posteriormente, se aplica el voltaje de programación en la entrada Vpp. Asimismo, se aplica el dato a grabar y la dirección en la que se desea grabar. Cuando las señales del dato y la dirección se han estabilizado en los buses de datos y direcciones, respectivamente, se aplica un pulso de programación en la entrada CE ("Chip Enable"). El pulso de programación debe aplicarse para cada localidad que se desee grabar y su duración es de 50 milisegundos (tiempo mínimo necesario, proporcionado por los fabricantes de memorias, para asegurar que el dato es grabado correctamente).

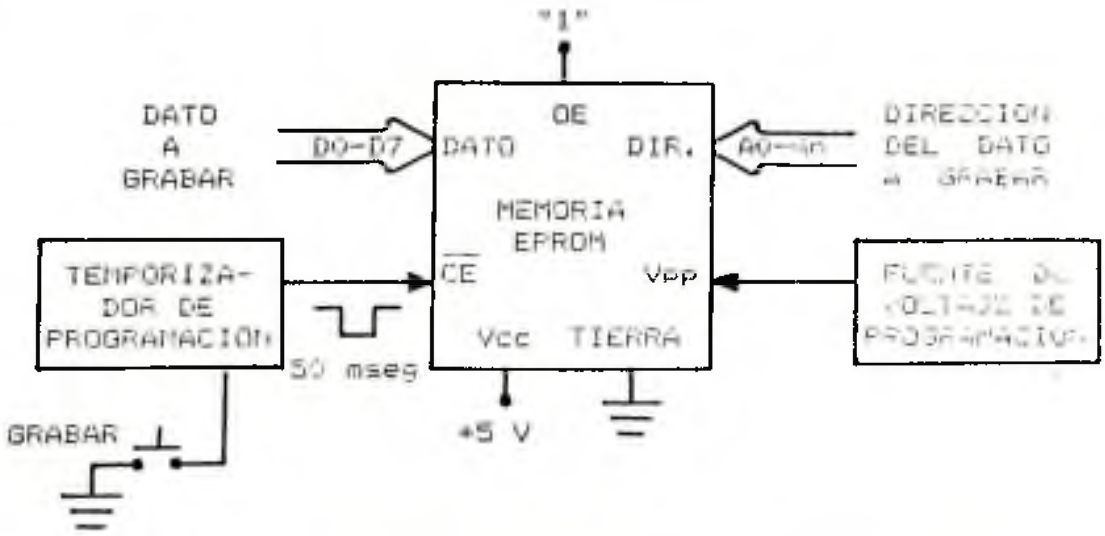


FIG. 2.2 CIRCUITO BASICO PARA GRABAR UNA EPROM

En la figura 2.3 se muestra un diagrama de tiempos basico para grabar una memoria EPROM. Debido a las diferencias que existen entre las memorias tratadas en el presente trabajo, las formas de onda mostradas no corresponden exactamente al total de dichas memorias. Sin embargo, la figura 2.3 ilustra de manera conceptual el procedimiento que se sigue para grabar.

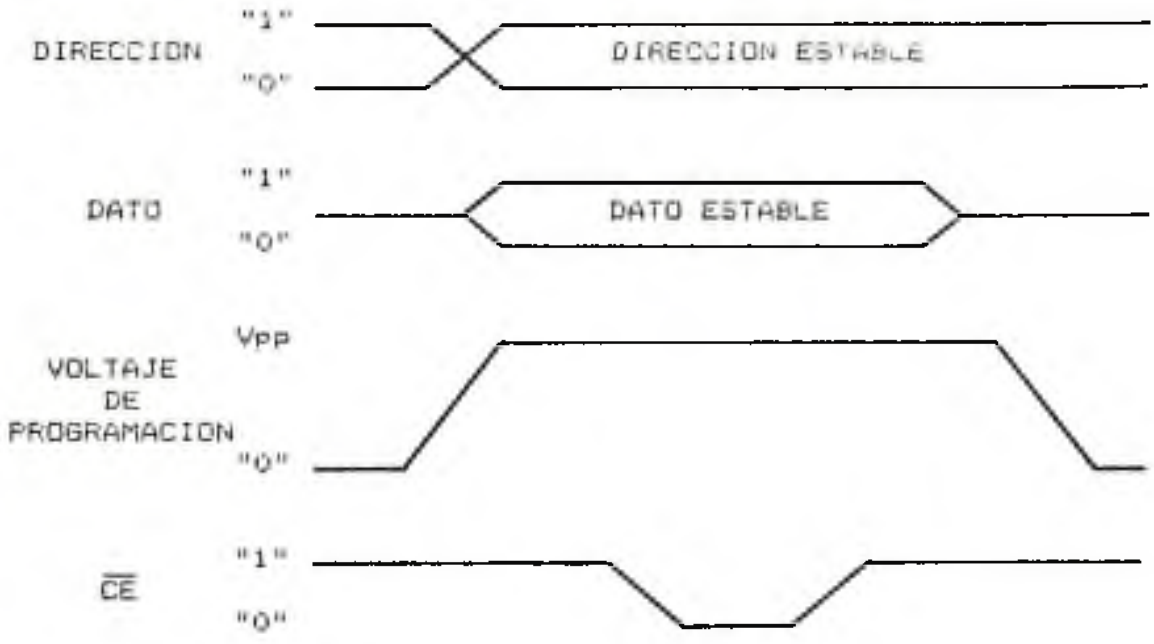


FIG. 2.3 FORMAS DE ONDA DE PROGRAMACION DE MEMORIAS EPROM

La arquitectura interna de las memorias; 2716, 2732, 2732A, 2764, 27128, y 27256, permite accesos de lectura y escritura por octetos (bytes). Esto significa que deben leerse o grabarse 8 bits simultaneamente.

La posibilidad de grabar 8 bits simultaneamente afecta positivamente el tiempo de grabación del total de la memoria, y facilita el uso de un microprocesador de 8 bits en la circuiteria del grabador.

PRINCIPIO DE OPERACION DE LECTURA

La lectura de una memoria EPROM consiste en conocer el estado lógico en que se encuentra cada celda bit. Con la ayuda de las figuras 2.4 y 2.5 se explica el principio basico de lectura.

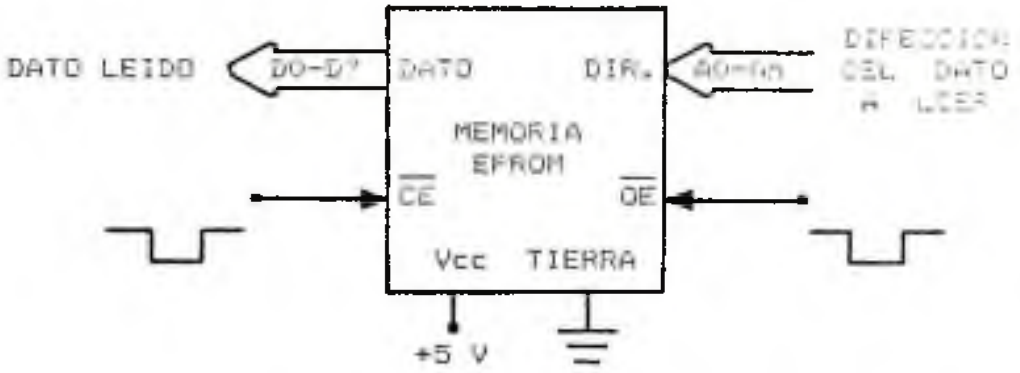


FIG 2.4 CIRCUITO BASICO PARA LEER UNA EPROM

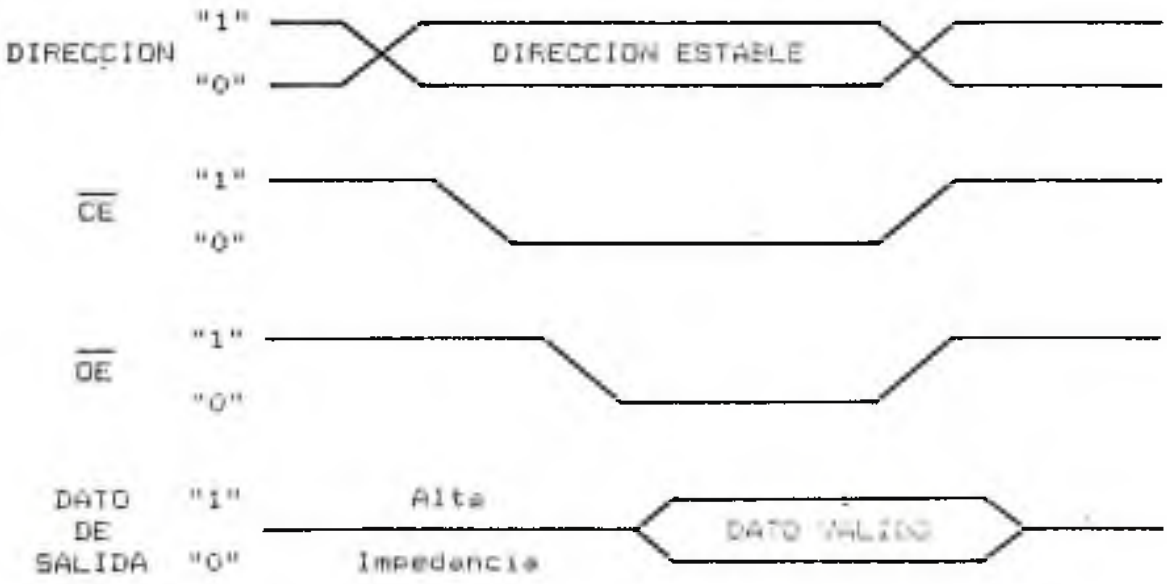
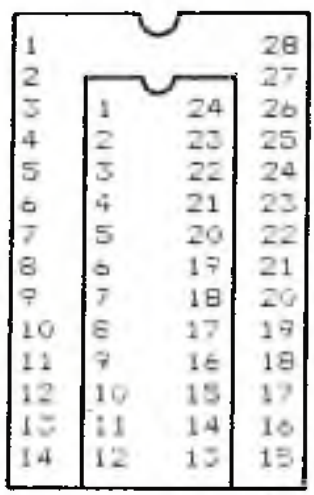


FIG. 2.5 FORMAS DE ONDA DE LECTURA DE EMORIAS EPROM

Para leer una memoria EPROM se debe aplicar la dirección del dato a leer en el bus de direcciones de la memoria. Subsecuentemente debe aplicarse un nivel bajo en la entrada $\overline{CE}$ para habilitar a la memoria. Finalmente, es necesario aplicar un nivel bajo en la entrada $\overline{OE}$ para hacer que el dato a leer este presente en el bus de datos de la memoria. Lo anterior se ilustra mediante el diagrama de tiempos de la figura 2.5.

Las memorias tratadas en el presente trabajo son compatibles en la mayoria de sus terminales de conexión, principalmente en sus buses de datos y direcciones. Sin embargo, existen ciertas diferencias en sus terminales de control. Estas diferencias se deben a varios factores, tales como: capacidad de almacenamiento, voltaje de programación, y número de terminales de conexion. En la figura 2.6 se muestra una tabla comparativa de las memorias mencionadas.

27256	27128	2764	27C32A	2716
Vpp	Vpp	Vpp		
A12	A12	A12		
A7	A7	A7	A7	A7
A6	A6	A6	A6	A6
A5	A5	A5	A5	A5
A4	A4	A4	A4	A4
A3	A3	A3	A3	A3
A2	A2	A2	A2	A2
A1	A1	A1	A1	A1
A0	A0	A0	A0	A0
O0	O0	O0	O0	O0
O1	O1	O1	O1	O1
O2	O2	O2	O2	O2
Gnd	Gnd	Gnd	Gnd	Gnd



2716	27C32A	2764	27128	27256
		Vcc	Vcc	Vcc
		PGM	PGM	A14
Vcc	Vcc	NC	A13	A13
A8	A8	A8	A8	A8
A9	A9	A9	A9	A9
Vpp	A11	A11	A11	A11
OE	OE/Vpp	OE	OE	OE
A10	A10	A10	A10	A10
CE	CE	CE	CE	CE
O7	O7	O7	O7	O7
O6	O6	O6	O6	O6
O5	O5	O5	O5	O5
O4	O4	O4	O4	O4
O3	O3	O3	O3	O3

La configuracion de la memoria 2732 es similar a la 27128.

Vpp(2716) = 25 V Vpp(2732) = 25 V Vpp(27C32A) = 21 V
Vpp(2764) = 21 V Vpp(27128)= 21 V Vpp(27256) = 13 V

FIG 2.6 CONFIGURACION DE LAS TERMINALES DE CONEXION

Las diferencias que existen entre las memorias descritas implican la necesidad de un circuito que genere las señales de control necesarias para cada tipo de memoria. Al conjunto de estas señales se le denomina "Patron de Voltaje", y al circuito que lo genera se le denomina "Conmutador de Patrones de Voltaje". Estas diferencias hacen que el número de tipos de memorias programables por el grabador sea limitado, debido a que con un mismo circuito conmutador de patrones de voltaje no es posible programar o leer cualquier memoria. Es decir, solo se pueden leer o grabar aquellas memorias para las cuales ha sido diseñado el circuito conmutador de patrones.

MODULO DIGITAL DEL GRABADOR

En esta seccion se muestra la arquitectura general del modulo digital del grabador, el cual se basa en un microprocesador Z-80. El significado de los bloques es el siguiente:

CPU - Microprocesador Z-80
MEM - Memoria EPROM con el Programa Monitor y
Memoria de Lectura/Escritura para los Datos
PCS - Puerto de Comunicacion Serie
TEM - Circuito Temporizador de Grabado
IDIR - Interfaz de Direcciones
IDAT - Interfaz de Datos
FVM - Fuente de Voltaje Multiple
CPAT - Conmutador de Patrones de Voltaje
MEMF - Receptáculo para la Memoria Fuente
MEMD - Receptáculo para la Memoria Destino

El microprocesador Z-80 opera a una frecuencia de 1.[illegible] MHz, misma que se aplica al puerto serie 8251. El puerto serie es programado por "hardware" para transmitir y recibir datos de la computadora a razón de 1200, 2400, 4800, ó 9600 baudios.

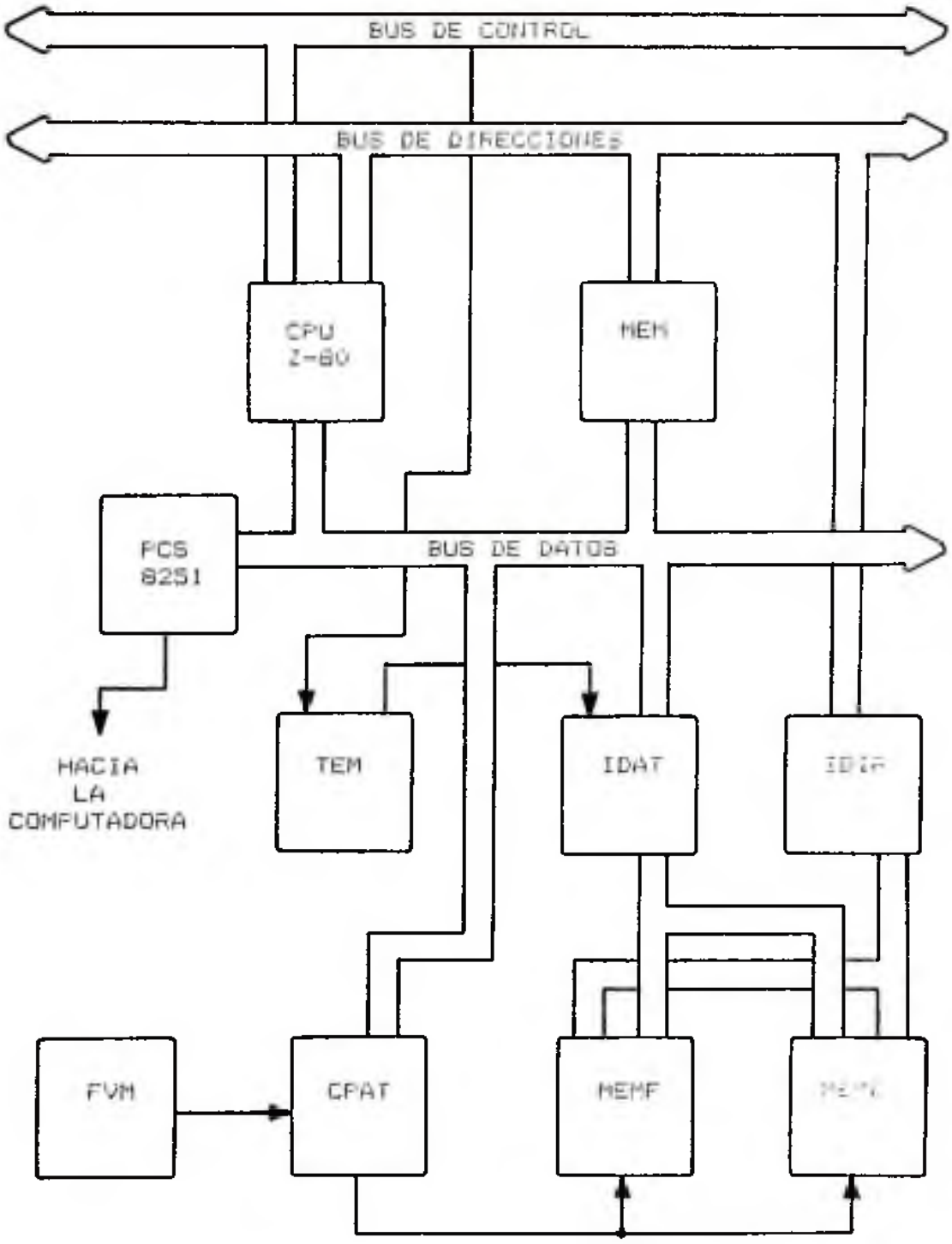


FIG. 2.7 ESTRUCTURA SIMPLIFICADA DEL MODULO DIGITAL DEL GENERADOR

En el bloque MEM se encuentra el programa monitor y la memoria RAM donde se guardan los datos a grabar.

El bloque llamado TEM contiene un circuito temporizador de grabado, compuesto por 2 multivibradores monoestaoles para mantener, durante 45 milisegundos (especificación del fabricante de memorias EPROM), el dato en la entrada de la memoria y la activacion de la misma. También este circuito es utilizado para generar una interrupción del microprocesador despues de cada dato grabado.

Las interfaces de datos y direcciones (IDAT e IDIR), son circuitos excitadores que sirven para proteger al microprocesador, de posibles memorias dañadas.

El bloque llamado CPAT es un circuito compuesto principalmente por un conjunto de multivibradores tipo D que almacenan un patrón de 8 bits, y un conjunto de interruptores electrónicos, hechos con transistores y diodos, para conmutar y aplicar a las memorias EPROM, los diversos voltajes de programacion.

En el bloque FVM se encuentra la fuente de alimentacion del módulo digital, la cual es una fuente múltiple cuyo diseño se basa en la regulacion de los voltajes por medio de diodos zener (para los voltajes de programación), y un circuito regulador integrado de 5 volts (para la alimentacion general). El módulo esta provisto de dos indicadores luminosos: uno para indicar el encendido de la fuente, y otro para indicar cuando una memoria esta siendo grabada o copiada.

Cada uno de los bloques MEMF y MEMD contiene un receptáculo de 28 terminales, en los cuales se insertan las memorias fuente y destino, respectivamente. El receptáculo de memoria fuente solo se utiliza cuando se realiza un copiado directo. En este caso, la seleccion del tipo de memoria se realiza manualmente.

MAPA DE MEMORIAS Y PUERTOS

Los puertos que contiene el modulo digital del grabador se encuentran mapeados en memoria. Por consiguiente, dentro de los 64 Koctetos direccionables por el microprocesador Z-80, se tienen 4 bloques de memoria y 6 puertos. Unicamente 4 de los 6 puertos son utilizados por el modulo digital, y los restantes quedan disponibles para una posible expansión del grabador con el objeto de incrementar los tipos de memorias EPROM a grabar.

La decodificación se realiza por medio de un circuito 74LS138, el cual ofrece 8 lineas de salida, y algunas compuertas conectadas directamente a las lineas de direccion A14 y A15 del microprocesador.

En la figura 2.8, la cual muestra el mapa de memoria, puede observarse que: los primeros 2 Koctetos (0000H-07FFH), contienen el programa monitor del grabador; y los 2 Koctetos siguientes (0800H-0FFFH), estan dedicados a una memoria tipo RAM que opera como memoria temporal para recibir de la computadora los datos a grabar. Esta memoria contiene tambien las constantes y variables propias del modulo digital.

El puerto serie 8251 maneja un registro de datos y uno de control, los cuales se encuentran mapeados en las direcciones 1000H y 1001H, respectivamente. El puerto mapeado en la direccion 1800H es para el circuito generador del patron de señales de voltaje. La lectura de los interruptores de seleccion de memoria, para modo copiado, se realiza mediante el puerto mapeado en la direccion 2000H. El encendido del bit A14 de la memoria fuente se reliza mediante el puerto localizado en la direccion 2800H.

La memoria EPROM fuente, utilizada unicamente para copiado directo, ocupa de la direccion 4000H a la 7FFFH. Mientras que, la memoria EPROM destino ocupa de la direccion 8000H a la FFFFH.

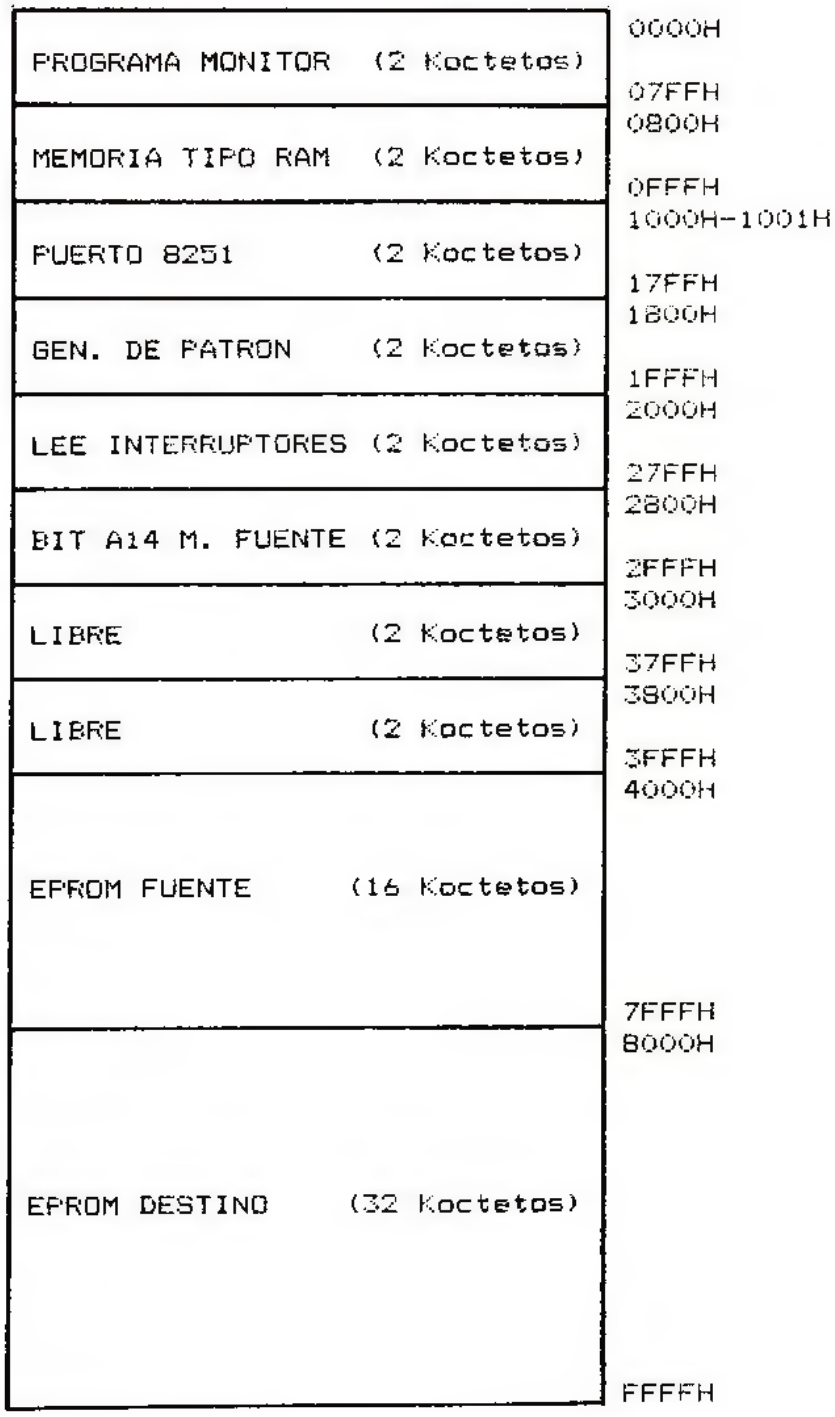


FIG. 2.8 MAPA DE MEMORIAS Y PUERTOS DEL GRABADOR

El espacio de 16 Koctetos que ocupa la memoria EPROM fuente, limita el proceso de copiado directo a memorias no mayores de la 27128 (16K x 8). Para resolver este problema con las memorias 27256, se copia primeramente la mitad más baja de la memoria y posteriormente se repite el procedimiento, pero con el bit más de la memoria fuente, encendido.

EL PROGRAMA MONITOR DEL MODULO DIGITAL DEL GRABADOR

El programa monitor, cuyo código se muestra en el apendice C, esta escrito en lenguaje ensamblador Z-80. Ha sido diseñado de manera que su estructura la conforman 4 partes fundamentales: el programa principal, la rutina de escritura, la rutina de lectura, y la rutina de copiado. Contiene además, una parte importante que es la rutina de interrupción, la cual puede considerarse como parte de las rutinas de copiado y escritura.

Como se observa en la figura 2.9, el programa principal es la parte que se encarga de determinar el modo de operación solicitado para saltar a la rutina correspondiente. Si el modo solicitado es local, se salta a la dirección del procedimiento COPIAR. Si el modo solicitado es remoto, se recibe una señal de requerimiento, la cual es reconocida mediante el envio de otra señal hacia la computadora. En seguida se recibe el tipo de comando (Leer o Grabar), y los parámetros correspondientes (Direc. Inicial, Num de Bytes, y Tipo de Memoria). Dependiendo del comando, se salta a la rutina de ESCRIBIR o de LEER. Despues de leer o escribir, se retorna al inicio del programa para esperar un nuevo comando.

Debido a que la velocidad de recepcion de los datos a grabar es mucho mayor que la de grabación, el programa monitor está provisto para controlar el flujo de datos e impedir que el "buffer" del modulo digital se desborde. El Modulo de interaccion con el Usuario del Grabador no maneja el control de flujo por lo que, actualmente, se envian al grabador bloques fijos, de menor tamaño que el del "buffer".

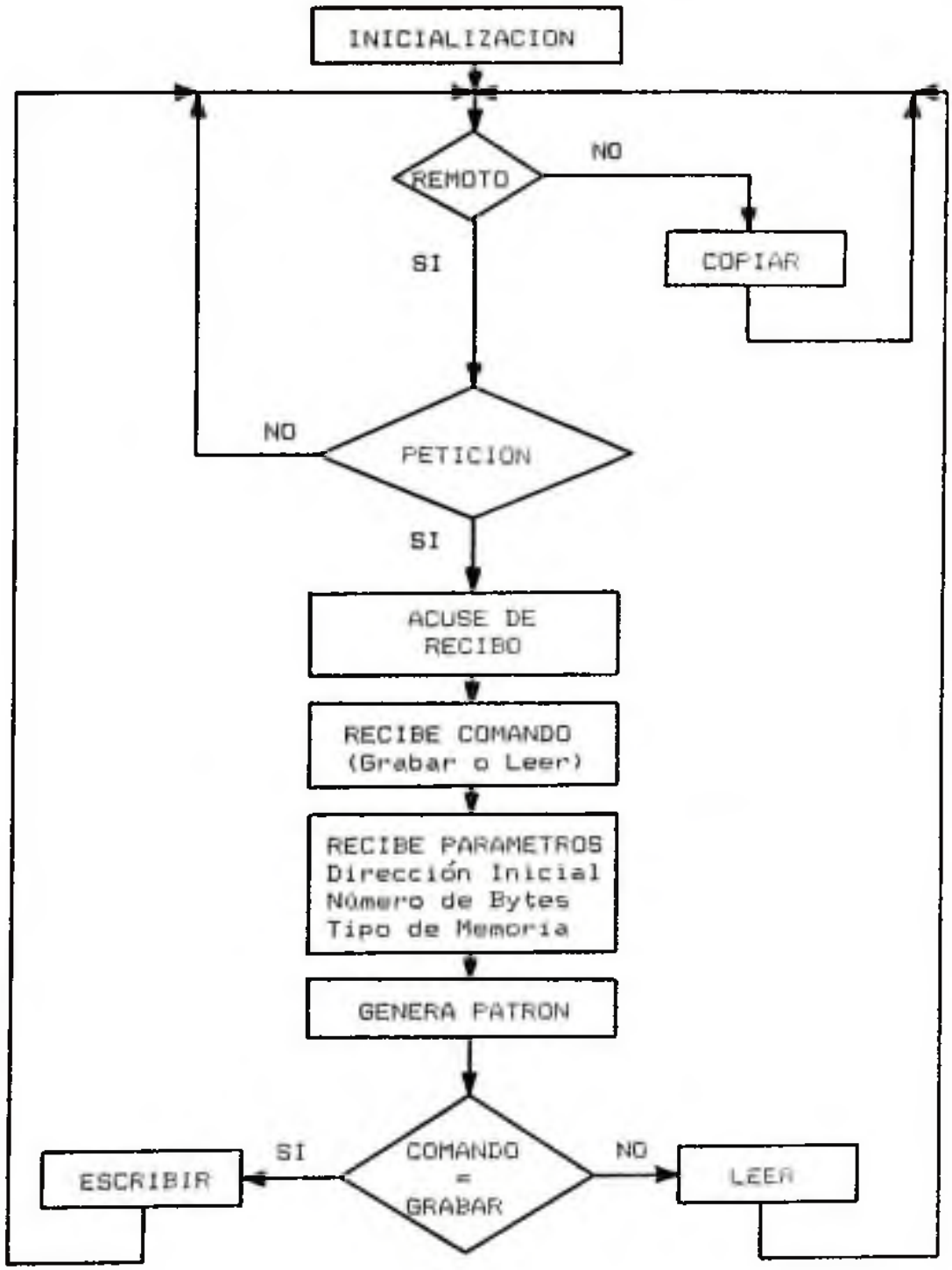


FIG. 2.9 ESTRUCTURA GENERAL DEL PROGRAMA MONITOR DEL GRABADOR

C A P I T U L O I I I

DISEÑO DE UN EMULADOR PARA LOS MICROPROCESADORES 8086 Y 8088

PRINCIPIO DE FUNCIONAMIENTO

El Emulador de Sistemas Digitales es un aparato que se conecta al prototipo del diseñador y a una computadora personal. La conexión con el prototipo se realiza a través de un cable de 40 hilos y un conector que se inserta en lugar del microprocesador en el prototipo del diseñador. De esta forma, el prototipo realiza todas sus operaciones con el microprocesador del emulador. La conexión con la computadora se hace a través de un cable de 3 hilos para comunicación en modo serie.

Cuando el prototipo del diseñador trabaja conjuntamente con el Emulador, la velocidad de operación puede ser la misma del prototipo.

La memoria existente en el Emulador de Sistemas Digitales se puede usar para extender o substituir, durante la etapa de pruebas, la del prototipo del diseñador.

Este sistema de desarrollo ofrece la posibilidad de seleccionar el reloj del prototipo o el del propio Emulador, durante las puebas. Con esta facilidad el usuario puede realizar sus pruebas antes de que su diseño disponga de un reloj propio. Asimismo, el usuario puede comprobar el buen funcionamiento de su circuito de reloj, reemplazando el del Emulador.

El modulo digital del Emulador esta provisto de un circuito llamado Interfaz Prototipo Diseñador. Dicho circuito aisla el Emulador de Sistemas Digitales del prototipo del diseñador, cuando no se esta emulando. Los circuitos que componen la Interfaz Prototipo Diseñador no introducen retardos que afecten la velocidad de operación del prototipo, ni las caracteristicas de bidireccionalidad y alta impedancia de las entradas y salidas del microprocesador. La circuiteria del Emulador de Sistemas Digitales es transparente para el prototipo del usuario.

PRINCIPIO DE FUNCIONAMIENTO

El esquema conceptual del Emulador de Sistemas Digitales se muestra en la figura 3.1, donde:

PC - Computadora Personal
MIU - Módulo de Interacción con el Usuario
MD - Módulo Digital
PS - Puerto Serie RS-232
IPD - Interfaz Prototipo Diseñador
CPU - Microprocesador
PD - Prototipo Diseñador

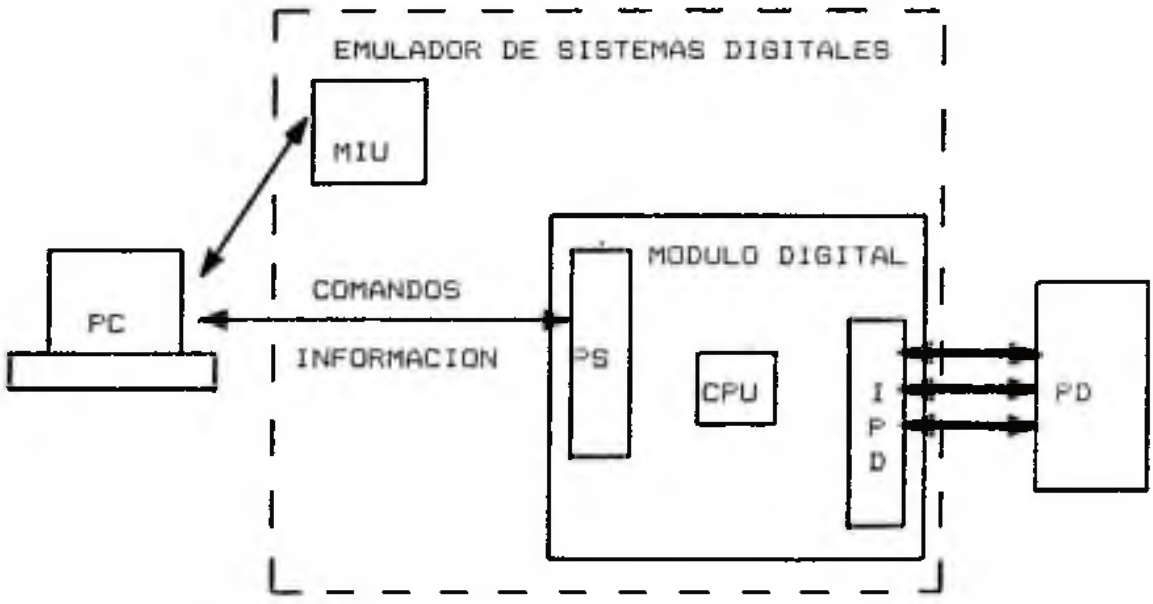


FIG. 3.1 ESQUEMA CONCEPTUAL DEL EMULADOR

El emulador de sistemas digitales esta constituido por:

1.- Un Módulo de Interacción con el Usuario
2.- Un Módulo Digital

El módulo de interacción con el usuario es un programa que se comunica con el módulo digital, a través de la computadora, bajo un esquema amo-esclavo. El módulo de interaccion con el usuario genera ordenes de comportamiento al prototipo de acuerdo con la solicitud del usuario, mientras que el modulo digital sigue las órdenes e informa a la computadora sobre el estado de comportamiento dinámico del prototipo.

CARACTERIZACION DEL EMULADOR

Recientemente algunas compañias han comercializado equipos acoplados a microcomputadoras personales como la IBM-PC y otras, para usarse como emuladores. A pesar de que se carece de literatura sobre el tema, en seguida se presenta un analisis del comportamiento, y de las funciones y elementos que se considera deben tener los emuladores de sistemas digitales.

Haciendo un análisis conceptual, de acuerdo con el funcionamiento descrito en la seccion anterior, se dice que el usuario mantiene el control sobre el emulador a traves de una relación directa con el programa de la computadora (MIU), el cual se presenta en forma de menú. De esta forma el usuario le da órdenes de comportamiento al emulador, a través de comandos o funciones completamente definidos. Dichos comandos se han agrupado en tres tipos:

Comandos de Servicio.- Permiten la creación de archivos objeto, definen los recursos que puede proporcionar el modulo digital al prototipo (reloj, memoria...).

Comandos de Transferencia.- Permiten cargar archivos en la memoria del módulo digital o del prototipo, asi como leer y/o escribir sobre memoria o puertos del prototipo.

Comandos de Ejecucion.- establecen el modo de ejecucion deseado (simple, automatico, tiempo real, por bloque).

Los comandos de ejecucion constituyen la parte mas importante del emulador ya que el objetivo de este es ejecutar programas con diferentes modalidades que ayuden al usuario a rastrear el comportamiento del hardware y software del prototipo, agilizando asi la depuración del mismo.

Durante la fase de pruebas del prototipo, el modulo digital proporciona el microprocesador al usuario con todas sus lineas de datos, direcciones y control. Cuando el usuario lo desea puede hacer uso de otros recursos provistos por el módulo digital, tales como el reloj y memoria de lectura/escritura.

El módulo digital observa el comportamiento dinámico del prototipo e informa sobre su estado al módulo de interaccion con el usuario, el cual muestra tal información en la pantalla de la microcomputadora. Apoyandose en la información desplegada en la pantalla, el usuario puede hacer correcciones en forma iterativa, tanto en el hardware como en el software, hasta alcanzar el objetivo deseado. El módulo digital esta constituido por un microprocesador 8086 u 8088, según sea el que se haya insertado en la base correspondiente al microprocesador, un puerto serie RS-232, a través del cual se realiza la transferencia de información entre el módulo digital y la microcomputadora, la memoria de programa, la memoria de datos, la circuiteria para decodificacion y control, la circuiteria para seleccion de tipo de microprocesador (8086 u 8088) y el modo de operacion (minimo o maximo), e interfaces con el prototipo. El tipo de microprocesador que el usuario elige para su prototipo determina el microprocesador que debe instalarse en el modulo digital. El programa monitor correspondiente al módulo digital incluye la comunicación con el módulo de interaccion con el usuario, asi como todos los comandos definidos en la siguiente seccion.

DISEÑO DEL EMULADOR

En esta sección se describen las bases de diseño para el Modulo de Interacción con el Usuario, y finalmente se especifica el diseño del Módulo Digital.

EL MODULO DE INTERACCION CON EL USUARIO DEL EMULADOR

Como ya se ha mencionado, el módulo de interaccion con el usuario es la interfaz entre el usuario y el módulo digital, por lo que ha sido diseñado en forma de menú. Dicho menu presenta los comandos cuya solicitud puede hacer el usuario.

Dado que algunos comandos tienen funciones semejantes entre si, estos son agrupados para formar submenues donde los parametros son requeridos uno a uno. Los comandos estan diseñados como subprogramas cuya sintáxis y semántica son verificadas para evitar incongruencias.

Los comandos definidos son los siguientes:

1.- Cargar archivo
2.- Salvaguardar archivo
3.- Leer y/o modificar registros
4.- Leer y/o modificar puertos de E/S
5.- Leer y/o modificar localicades de memoria
6.- Ejecutar paso a paso manualmente
7.- Ejecutar paso a paso automaticamente
8.- Ejecutar por bloque
9.- Ejecutar en tiempo real
10.- Facilitar reloj emulador
11.- Inicialización

Tales comandos son agrupados en 5 menúes como sigue:

1.- Cargar y/o salvaguardar un archivo
2.- Leer y/o modificar registros, memoria, puertos
3.- Ejecutar
4.- Condiciones de control
5.- Inicializacion

Los comandos de cargar y salvaguardar archivo realizan tales funciones hacia o desde la memoria del modulo digital o del prototipo, respectivamente. El comando cargar archivo requiere de dos parámetros: el nombre del archivo objeto, y la direccion inicial de carga, la cual es opcional ya que, el MIU introduce una direccion de carga predeterminada cuando el usuario no la especifica. Para salvaguardar archivo se requieren las direcciones inicial y final, además del nombre del archivo en donde se va a guardar la información leida.

Los comandos para leer y/o modificar registros, puertos, y localidades de memoria, son equivalentes en su semántica pero no en su sintáxis. En ellos, el usuario especifica el nombre del registro o la dirección del puerto o la localidad de memoria, y el emulador muestra su contenido; en seguida el usuario tiene la opción de modificarlo o no, antes de pasar al próximo registro, puerto o localidad, o bien, regresar al menú principal.

Los comandos de ejecucion operan con cuatro modos distintos: La ejecución simple o paso a paso significa que despues de cada instruccion ejecutada se despliega el contenido de todos los registros del procesador, y el usuario tiene la opción de modificarlos o no. La ejecucion automática consiste en desplegar, durante 5 segundos y en forma automatica, el contenido de los registros después de cada instrucción ejecutada, sin que el usuario pueda modificarlos. La ejecucion por bloque consiste en ejecutar un conjunto de intrucciones, especificado por el usuario, y solo al término de éste se desplegará el contenido de los registros del procesador, dando al usuario la opción de

modificarlos. El comando de ejecución en tiempo real permite al usuario mandar un programa a ejecutarse en tiempo real. Este comando, en contraste con los anteriores, cede el control del sistema al programa del usuario. Al inicio de cada modo de ejecución (antes de ejecutar la primer instrucción), se despliega el contenido de los registros con el objeto de que el usuario pueda dar valores iniciales, si así lo desea. En caso contrario, el emulador introduce valores implícitos. En todos los modos de ejecución, excepto el de por bloque, solo se requiere la dirección inicial como parametro. En este último se pide además la dirección final del bloque.

El comando iniciar se encarga de activar las señales (RESET) que establecen las condiciones iniciales del módulo digital y del prototipo.

El comando facilitar reloj emulador permite hacer uso del reloj del módulo digital en el prototipo. Cuando el prototipo tiene reloj propio el usuario no requiere de este comando.

PRINCIPIO BASICO DE UN SISTEMA DE DESARROLLO

Un sistema de desarrollo es una herramienta de diseño que, fundamentalmente, permite al diseñador de sistemas digitales ejecutar programas desde el propio ambiente del prototipo, sin restringir la arquitectura del mismo.

El diseñador puede hacer uso de recursos del sistema, y hacer ejecutar programas en modo continuo o en modo interpretado (paso a paso). La capacidad de rastreo ofrece una historia detallada de la ejecución del programa, antes de la interrupción. Un sistema de desarrollo ofrece capacidades auxiliares como: leer y/o modificar memoria y puertos, cargar archivos en memoria, y salvaguardar archivos en disco.

Conceptualmente, un sistema de desarrollo es utilizado en tres etapas del diseño:

1.- Antes de alambrar el prototipo. El emulador puede ser operado sin estar conectado al prototipo del diseñador, ya que, las capacidades de ejecución interrumpida y la disponibilidad de memoria pueden utilizarse para facilitar el desarrollo del programa, antes de que este disponible la circuiteria de interfaz.

En el presente trabajo, el emulador esta provisto de un espacio de memoria disponible para el programa prototipo. (vease el mapa de memorias mostrado en la figura 3.5). Durante la primer etapa del desarrollo, el tamaño del programa prototipo no debe exceder este espacio.

2.- Durante el alambrado del prototipo. La integracion de la programación y la circuiteria puede irse logrando conforme los elementos funcionales de prototipo sean conectados al sistema de desarrollo. El programa prototipo puede ser probado en la memoria del sistema de desarrollo o en la memoria del prototipo. Tambien puede usarse, durante las pruebas, una combinación de ambas memorias. Además de estas capacidades, se puede agregar la facilidad de utilizar, en las pruebas, el reloj del emulador y el circuito de borrado (reset), cuando estos elementos no han sido alambrados en el prototipo.

En la figura 3.2 se muestra un ciclo tipico de desarrollo de un prototipo, mediante la ayuda de un sistema de desarrollo. En dicha figura se ilustra la forma interactiva en que se verifican la circuiteria y la programación. Esta es una caracteristica fundamental de los sistemas de desarrollo capaces de emular.

3.- Después de alambrar el prototipo. Cuando el diseño del usuario esta completo, éste es probado con la version final del programa prototipo. En esta parte se utiliza al sistema de desarrollo en el modo de emulacion en tiempo real para probar al prototipo como una unidad completa.

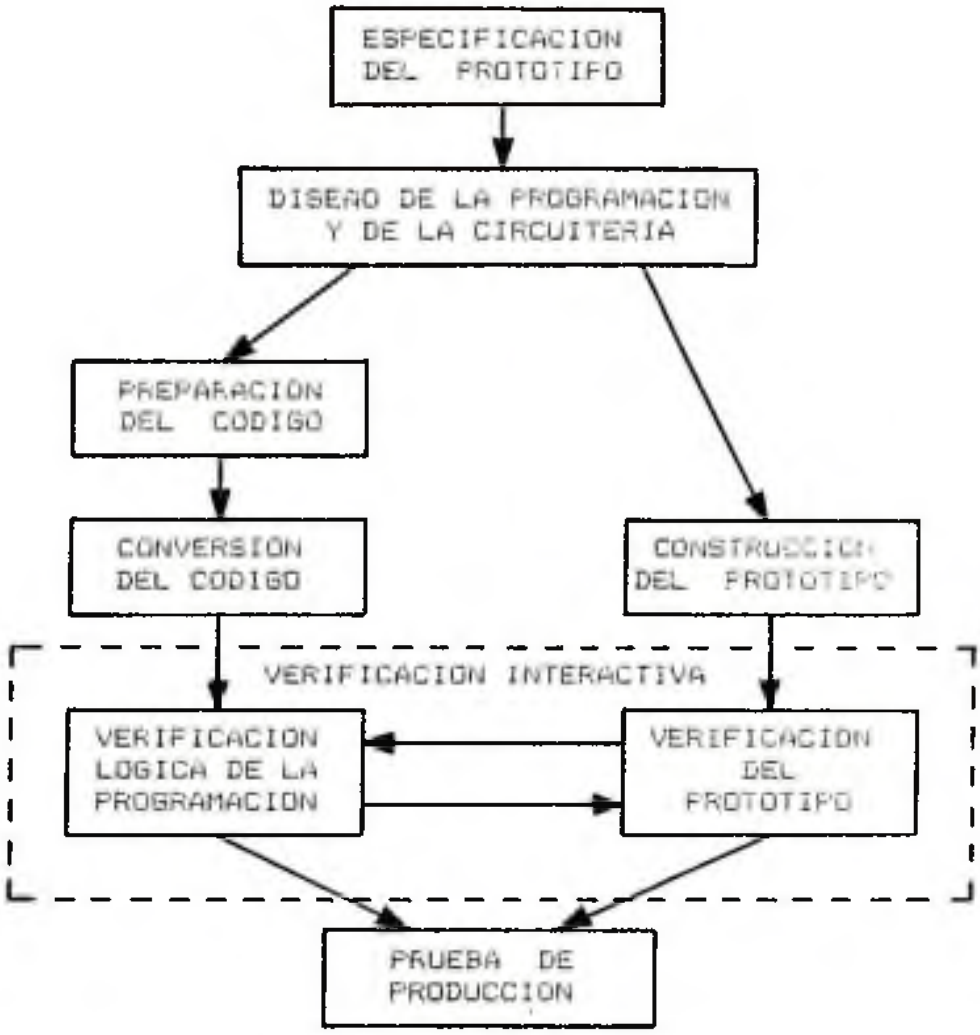


FIG. 3.2 CICLO DE DESARROLLO DE PROTOTIPOS MEDIANTE UN EMULADOR

La facilidad que ofrece el sistema de desarrollo, presentado en este trabajo, para seleccionar entre el reloj del emulador y el del prototipo, se obtiene a partir de un circuito basado en compuertas excitadoras de tres estados, como se muestra en la figura 3.3. Las condiciones posibles son:

Implícita.— Esta condicion aparece al encender el sistema o al inicializarlo, y habilita solamente la compuerta C1.

Reloj emulador.- Esta condición habilita, mediante un comando, las compuertas C1 y C3. De esta forma, el reloj del emulador alimenta al microprocesador del emulador y a los circuitos del prototipo que requieren esta señal. Si el reloj del usuario existe, debe ser desconectado para evitar conflictos con el reloj del emulador (abrir en el punto A referido en la figura).

Reloj usuario.- Cuando se utiliza el reloj del usuario, se habilita unicamente la compuerta C2. Así, el microprocesador del emulador es alimentado con la señal de reloj del usuario.

La seleccion de la señal de borrado (reset), se realiza mediante un circuito similar al selector de reloj.

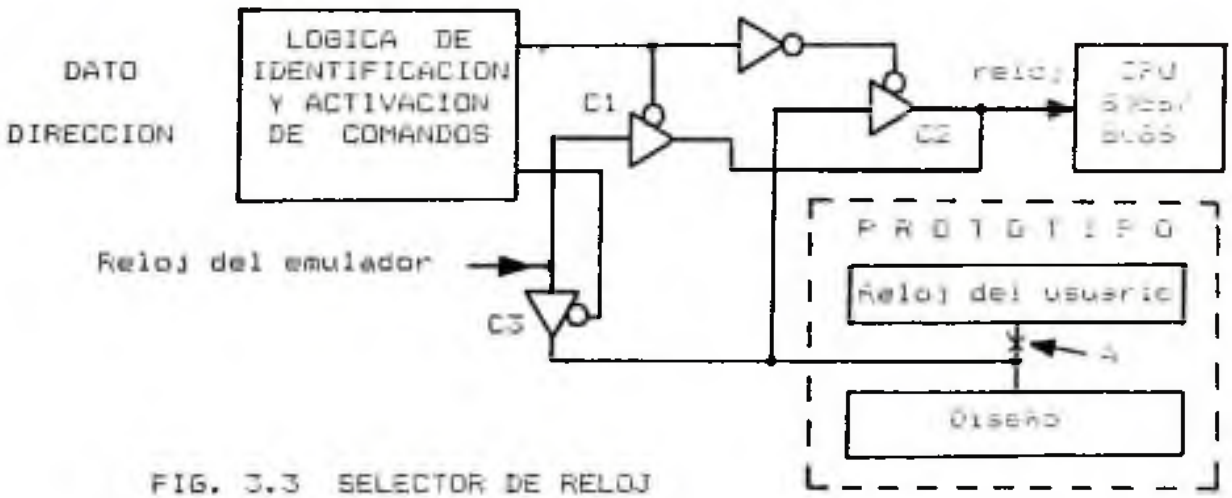


FIG. 3.3 SELECTOR DE RELOJ

EL MODULO DIGITAL DEL EMULADOR

En esta seccion se define la arquitectura general del modulo digital, cuyo diseño está orientado hacia la emulación de sistemas digitales basados en microprocesadores de 8 bits. La figura 3.4 muestra, en modo simplificado, la estructura del modulo digital. En el apéndice B se muestra un diagrama mas detallado.

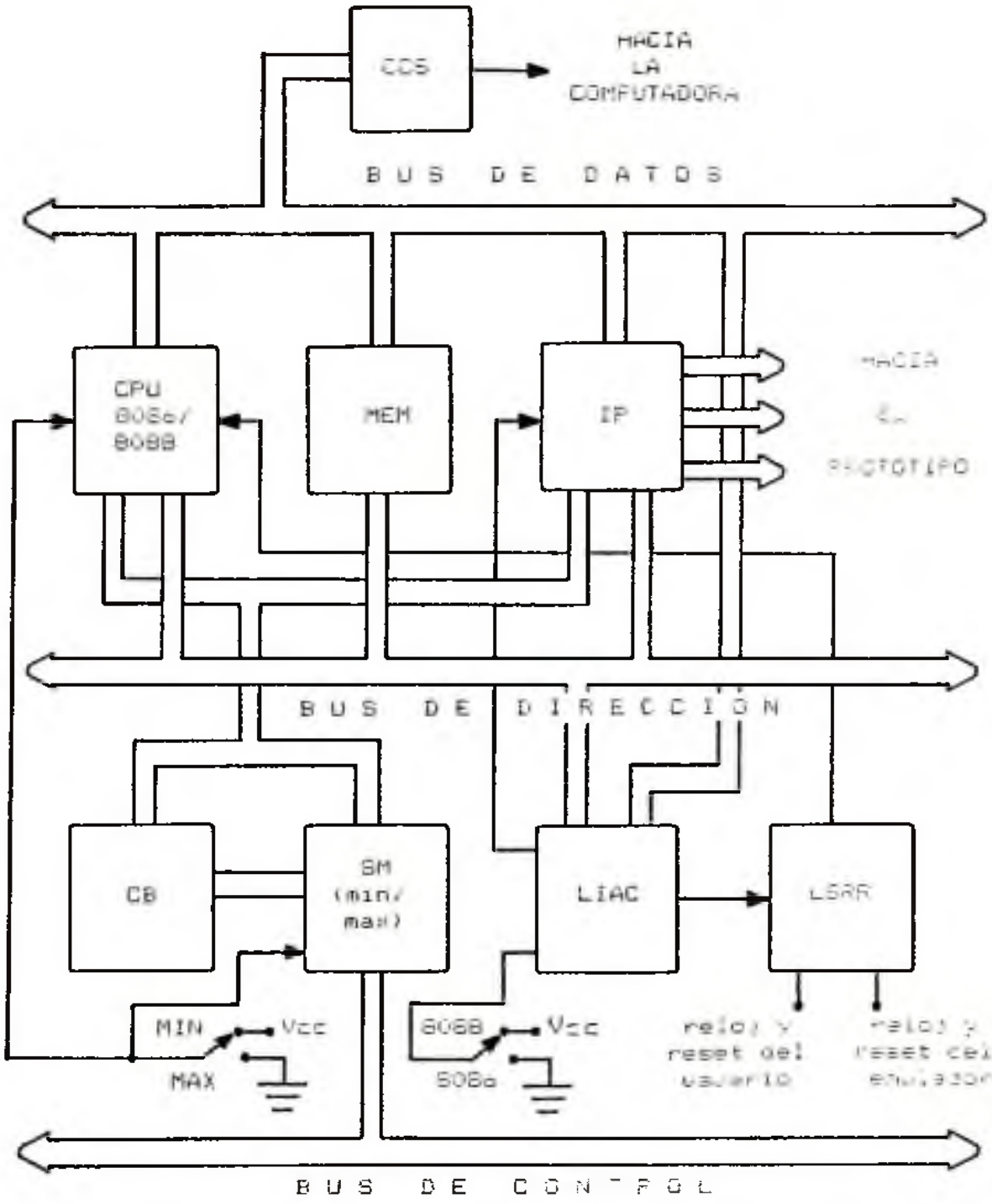


FIG 3.4 ESTRUCTURA SIMPLIFICADA DEL MODULO DIGITAL DEL EMULADOR

Como puede observarse en la figura 3.4, el modulo digital tiene dos interruptores: El primero de ellos es de dos posiciones y permite al usuario la seleccion del tipo de microprocesador a emular (8086 u 8088), previa colocación del microprocesador respectivo en el módulo digital. El siguiente interruptor sirve para seleccionar el modo de operacion del microprocesador (mínimo o maximo). El significado de los bloques de la figura 3.4 es el siguiente:

CCS - Circuito de comunicación serie (basado en un C.I. 8250).
EDA - Excitador de datos.
CPU - Microprocesdor 8086 u 8088.
MEM - Memoria EPROM para el monitor del módulo digital, y memoria RAM para los datos del módulo digital y para los programas del prototipo).
LIAC- Lógica de interpretación y de activación de comandos.
CB - Controlador de bus.
SM - Selector de modo de operación (modo mínimo o máximo).
LSRR- Lógica de selección de reloj y reset.

MAPA DE MEMORIAS Y PUERTOS

Los microprocesadores 8086 y 8088 tienen la capacidad de direccionar hasta 16 segmentos de 64 Koctetos cada uno, es decir, un millón de bytes (1 Mega-octeto) . El diseño del módulo digital solo maneja un reducido espacio de memoria del Mega-octeto, incluyendo en este un área de memoria disponible para el programa de usuario. Sin embargo, esto no representa una limitación en memoria para el diseñador del prototipo, ya que puede disponer del resto del espacio de memoria hasta alcanzar el Mega-octeto. Ciertas localizaciones de memoria son reservadas para operaciones especificas de estos microprocesadores. Los últimos 16 octetos de todo el espacio de memoria son usados para incluir un salto a la rutina de inicialización del sistema. Los 1024 octetos

más bajos del espacio de memoria, mantienen las direcciones de programas que deben ejecutarse cuando un dispositivo externo interrumpe. Estas direcciones son conocidas como vectores de interrupcion. Por lo anterior, el mapa de memoria del módulo digital es tal que utiliza parte del último segmento y parte del primero de los 16 que conforman el total.

La memoria de lectura/escritura (RAM), formada por dos circuitos integrados (6164) de 8 Koctetos cada uno, ocupa los primeros 16 Koctetos del mapa. Esta memoria es usada para los vectores de interrupción, tablas, constantes, variables..., propias del módulo digital y de la emulación particular. En esta memoria se reserva un espacio de la direccion 00600H a la 03FFFH para programas del usuario.

La memoria de lectura (EPROM), formada por dos circuitos integrados (2732), de 4 Koctetos cada uno, ocupa los últimos 8 Koctecos del mapa. En esta memoria reside el programa monitor del módulo digital y la dirección de salto a la rutina de inicialización.

Además del espacio de memoria de 1 Mega-octeto, los microprocesadores 8086 y 8088 pueden hacer referencia a dispositivos externos por medio de puertos de entrada/salida. Existen instrucciones para accesar directamente los primeros 256 puertos (0 a 255). Otras instrucciones permiten accesar indirectamente, mediante un registro de datos, hasta 64 K puertos de entrada/salida; es decir, 65536 puertos. El módulo digital ocupa solamente 16 puertos (F0 a FF). Por consiguiente, el usuario dispone de 240 puertos directamente, o de 65520 puertos en forma indirecta.

El prototipo debe carecer de memorias en el espacio utilizado por el módulo digital, al menos, durante el periodo de pruebas. Al terminar el proceso iterativo, el usuario puede colocar sus memorias. Esto significa que, durante el proceso iterativo, el

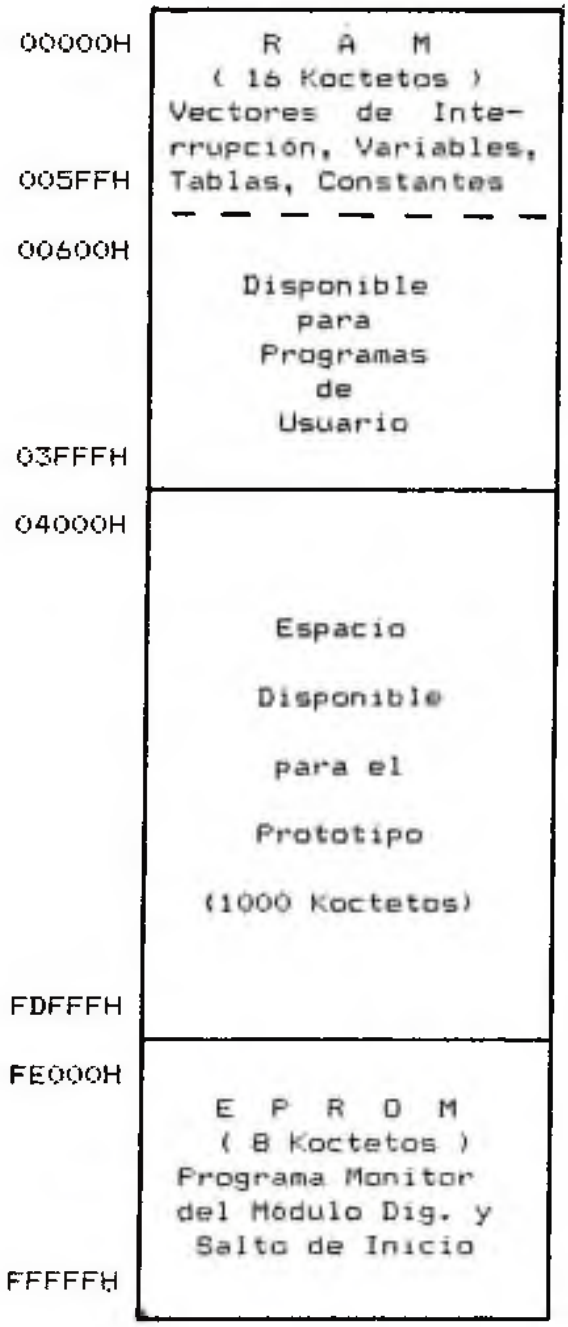


FIG. 3.5 MAPA DE MEMORIAS

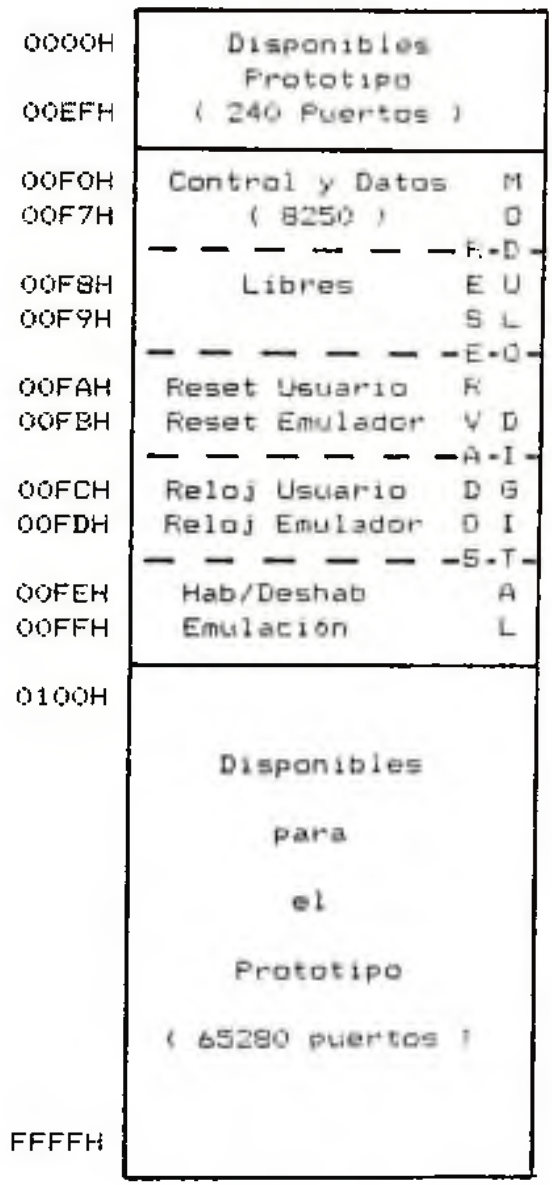


FIG. 3.6 MAPA DE PUERTOS

prototipo si puede tener la decodificacion respectiva, pero no las memorias. Para lograr esta forma de trabajo, durante el proceso iterativo, el programa del prototipo debera estar localizado en un espacio distinto de la memoria EPROM del módulo digital, y al finalizar dicho proceso, debera ser relocalizado el programa para ocupar tal espacio.

En las figuras 3.5 y 3.6 se muestra esquematicamente la disposición de las memorias y puertos respectivamente.

COMANDOS DE RELOJ E INICIO (RESET)

El diseño de estos comandos implica cierta lógica que es activada mediante el circuito de control de comandos, haciendo referencia a los puertos, FAH-FBH para el comando de inicio, y F0H-F0H para el comando de reloj. El inicio o reset puede ser ordenado por: el prototipo, el reset maestro del modulo digital, o el comando de inicio. La lógica de control de la inicializacion proporciona la señal correcta según sea el caso.

El módulo digital opera con su propio reloj, de manera implicita, e ignora la posible existencia de señal de reloj en el prototipo, excepto cuando se usa el comando de reloj. El comando de reloj sirve para emular con el reloj del prototipo. La señal de reloj, asi como la de reset, son suministradas directamente al microprocesador 8088 u 8086 por un circuito (8284), encargado de sincronizar ambas señales.

La interfaz con el prototipo tiene la funcion de evitar que la circuiteria del prototipo interfiera con las lineas de direcciones, datos, y control, del módulo digital. Dicha interfaz actúa como una compuerta bidireccional que solo se abre cuando se requiere una transferencia de datos prototipo-modulo digital, o viceversa. También actúa como protección del módulo digital,

cuando el prototipo tiene fallas. En la interfaz se utilizan compuertas excitadoras de tres estados; bidireccionales para las lineas de dirección y datos (74LS245), y unidireccionales para las lineas de control del microprocesador (74LS244).

EL PROGRAMA MONITOR DEL MODULO DIGITAL DEL EMULADOR

El programa monitor esta escrito en lenguaje de alto nivel F[illegible], y se utilizan ocasionalmente algunas partes de codigo escritas directamente en lenguaje ensamblador. El estilo utilizado en la programación es modular, con un modelo como el que se muestra en la Fig. 3.7.

El programa es diseñado modularmente, de tal forma que, está constituido por un conjunto de procedimientos independientes entre si, los cuales pueden ser llamados por cualquiera de los comandos para realizar su función. Por consiguiente, un comando es considerado, dentro del modulo digital, como uno o mas llamados a procedimientos. Cada comando puede ser ejecutado en forma iterativa hasta que el usuario solicite dejarlo (regreso a menú principal). Unicamente con el comando de inicialización o RESET, el procesador ejecuta el programa nuevamente desde el inicio. La comunicación con el modulo de interaccion con el usuario (MIU), se realiza mendiante un protocolo sencillo, basado en una señal de requerimiento (ENQ), que recibe el módulo digital, y otra de reconocimiento (ACK), que envia el módulo digital al MIU.

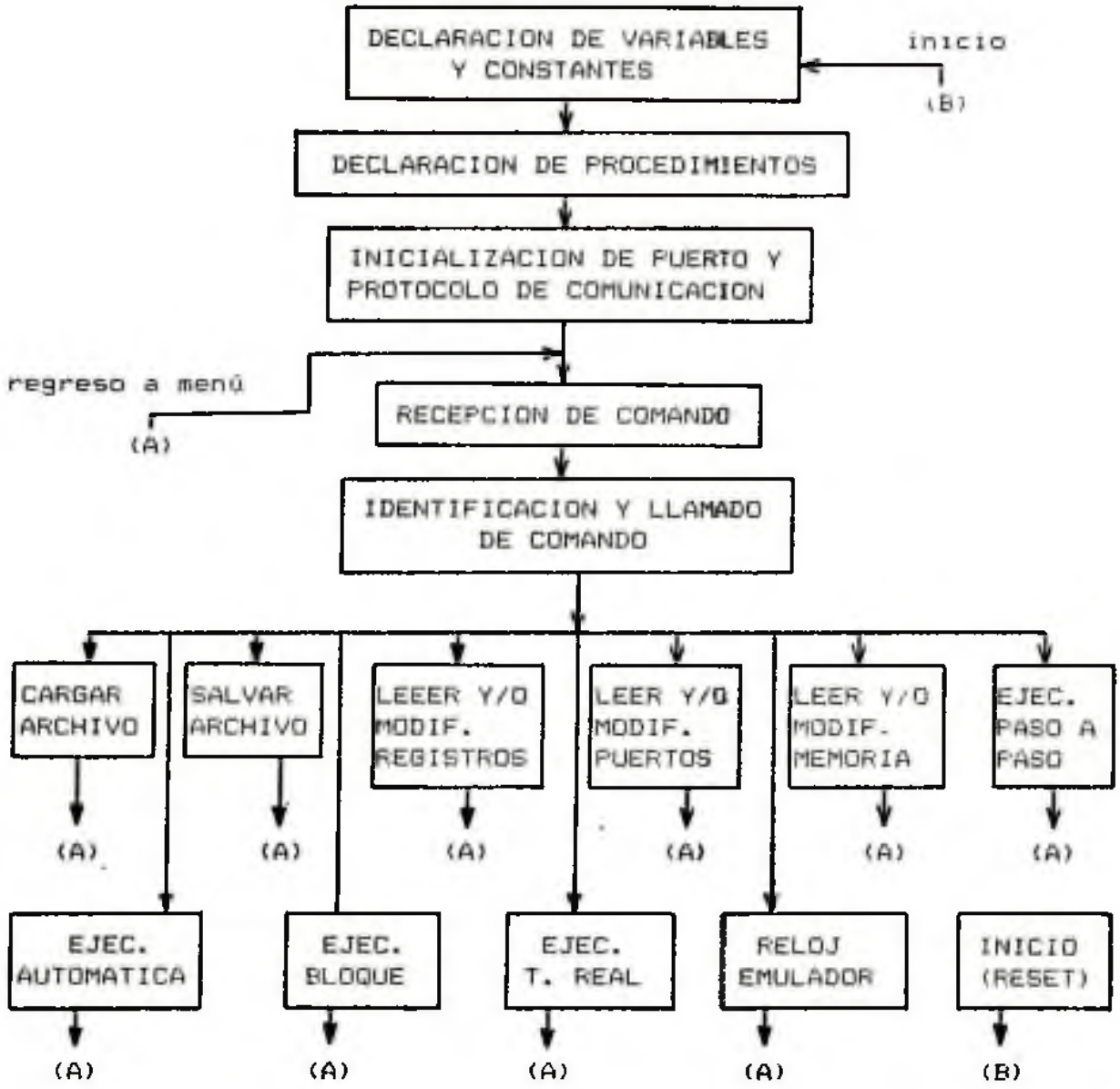


FIG. 3.7 MODELO GENERAL DE PROGRAMACION DEL MODULO DIGITAL DEL EMULADOR

C A P I T U L O I V

CONSTRUCCION, PRUEBAS Y RESULTADOS

CONSTRUCCION DEL GRABADAOR

El grabador de memorias fue construido bajo la técnica de alambre enrollado" (wire-wrapping) sobre dos tarjetas de proposito general de 20 x 11 centimetros. En una de ellas se localizan: el microprocesador Z-80, las memorias EPROM y RAM, el puerto de comunicación serie 8251, el circuito de decodificacion, las interfaces de datos y memorias, los selectores de control directo, el temporizador de grabado, un conector hacia el puerto serie de la computadora, y dos receptaculos de 28 terminales para las memorias fuente y destino. En la otra tarjeta se localizan: la fuente de voltaje múltiple que alimenta a las memorias y el grabador mismo, y el circuito generador de patrones de voltaje para las memorias. Ambas tarjetas estan conectadas mediante un cable plano de 40 hilos. Se tiene construido, actualmente, un gabinete en el cual serán instaladas las dos tarjetas del grabador para hacerlo más portátil. En la figura 4.1 se muestra una fotografia de la parte superior del grabador.

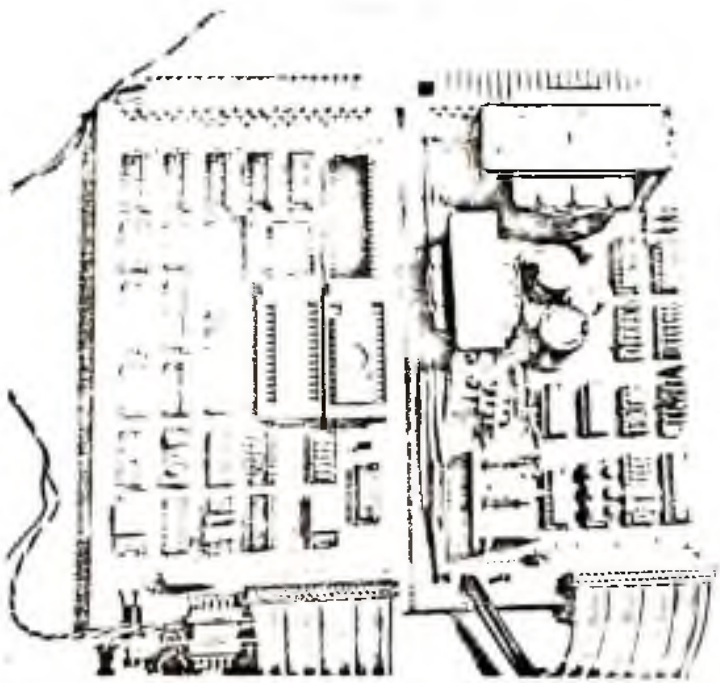


FIG. 4.1 GRABADOR UNIVERSAL DE MEMORIAS EPROM.

PRUEBAS EN EL GRABADOR

Durante la etapa de armado del grabador, y mientras se realizó la programación necesaria para el modulo de interacción con el usuario del grabador, las pruebas se hicieron con la ayuda de una terminal no inteligente, simulando a la computadora personal. Asimismo, el equipo utilizado durante las pruebas fue una microcomputadora, marca Cromemco, para ensamblar los programas de prueba; un osciloscopio de doble trazo; un analizador de estados logicos con capacidad para retener 16 palabras de 16 bits a partir de la llave de disparo; y dos multimetros (uno analogico y otro digital). Para grabar las memorias EPROM con los programas de prueba de la circuiteria y el programa monitor del grabador, se utilizó unicamente un grabador para memorias 2716 acoplado a un "kit MKE Z80", mediante el cual se introdujo manualmente el codigo de cada prueba, por lo que resultó laborioso este procedimiento.

Las principales pruebas que se le efectuaron al grabador son las siguientes:

- Medición y ajuste de valores de salida en la fuente de voltaje múltiple (voltajes de alimentacion y de grabado).
- Prueba de "eco". Comprobar la correcta transmisión y recepción a través del puerto serie.
- Generación correcta de los patrones de voltaje para la lectura y escritura de cada una de las memorias.
- Medición y ajuste del tiempo de grabado para cada localidad, en el circuito temporizador de grabado.
- Pruebas de copiado directo con cada uno de los tipos de memoria que maneja el grabador.
- Finalmente, se probaron los comandos de lectura y escritura, haciendo uso del modulo de interaccion con el usuario del grabador. Esta prueba se realizó conectando el grabador de memorias a la computadora personal.

RESULTADOS DEL GRABADOR

La capacidad de almacenamiento de las memorias influye directamente sobre el tiempo de programacion total de estas. Estadisticamente, se sabe que la mayoria de las localidades de las memorias EPROM pueden ser grabadas en tiempos menores a los 8 milisegundos. Sin embargo, el tiempo minimo necesario para asegurar el grabado correcto de una localidad, es de 45 milisegundos. Este tiempo, que es el peor de los casos, es el que utiliza el temporizador del Grabador Universal. Por consiguiente, los resultados obtenidos de las pruebas realizadas son los siguientes:

MEMORIA	CAPACIDAD DE ALMACENAMIENTO	TIEMPO TOTAL DE PROGRAMACION
2716	2 Koctetos X 8	1 min. 40 seg.
2732	4 Koctetos X 8	3 min. 20 seg.
2732A	4 Koctetos X 8	3 min. 20 seg.
2764	8 Koctetos X 8	6 min. 40 seg.
27128	16 Koctetos X 8	13 min. 20 seg.
27256	32 Koctetos X 8	26 min. 40 seg.

Estos tiempos pueden reducirse si se utiliza la tecnica de programación inteligente. Dicha técnica es permitida en las memorias de mayor densidad (2764, 27128, 27256).El algoritmo de programación inteligente consiste principalmente en aplicar, a la memoria, pulsos de corta duracion (1 o 4 milisegundos), en forma iterativa. Despues de cada pulso se verifica la programacion. Si el dato ha sido grabado se pasa a la siguiente localidad de la memoria. Cuando el dato no se graba después de cierto numero de pulsos (tipicamente se utilizan 15 pulsos), se determina que la memoria esta dañada o que existe una falla. Con la programacion inteligente se pueden reducir los tiempos mostrados anteriormente a una quinta parte, aproximadamente.

Confiablidad de Programación

La confiabilidad de programación del Grabador Universal es definida como; el grado de certeza de que los datos que se hacen grabar en una memoria EPROM quedan completa y correctamente grabados.

Considerando que el tiempo de uso del grabador es, actualmente, mayor a los 18 meses, y que los problemas ocurridos en ese periodo han sido exclusivamente por fallas en los componentes o por falsos contactos, pero nunca por mal diseño, se puede afirmar que la confiabilidad es de 100%.

Seguridad

La seguridad del Grabador Universal se define como; la confianza que se tiene de que las memorias que se insertan en éste, para ser grabadas, no sufren daños ocacionados por el grabador.

Durante las primeras pruebas, se observó que el grabador dañaba con cierta frecuencia las memorias que se programaban en él. Esto se debió a que el encendido de los voltajes de programación y los de alimentación del grabador se realizaba simultáneamente, lo cual provocaba señales transitorias que resutaban peligrosas para las memorias (según el manual del fabricante). El problema fué corregido, separando los interruptores de encendido de la fuente de alimentacion del grabador (Vcc), y los voltajes de programacion (Vpp). El procedimiento, ahora, consiste en encender primero Vcc y despues Vpp. Para apagar, primero es Vpp y despues Vcc. Con esta corrección, la seguridad que se ha alcanzado es de 100%.

CONSTRUCCION DEL EMULADOR

El diseño del emulador esta contemplado para formar parte de un equipo modular de desarrollo, para varios tipos de microprocesadores, basado en una tarjeta común y varias opcionales (una para cada tipo de microprocesador). Por consiguiente, la construcción ha sido separada en dos tarjetas: la primera es justamente el emulador, y la segunda es la tarjeta común. En la primer tarjeta se localiza: el microprocesador 8086 u 8088, las memorias EPROM y RAM, la interfaz con el prototipo, la logica de decodificación y control de comandos, el controlador del bus, los selectores de configuración y tipo de microprocesador. En la misma tarjeta se localiza la interfaz con el prototipo y un cable plano de 40 hilos terminado con un conector tipo "dual inline plug", que se inserta en la base del microprocesador del prototipo. La segunda tarjeta constituye la parte común del equipo modular mencionado. En ella se encuentra basicamente la parte de comunicación con la computadora personal, basada en un elemento de comunicación asincrona (8250). El montaje fué hecho con la técnica de "alambre enrollado" (wire-wrapping).

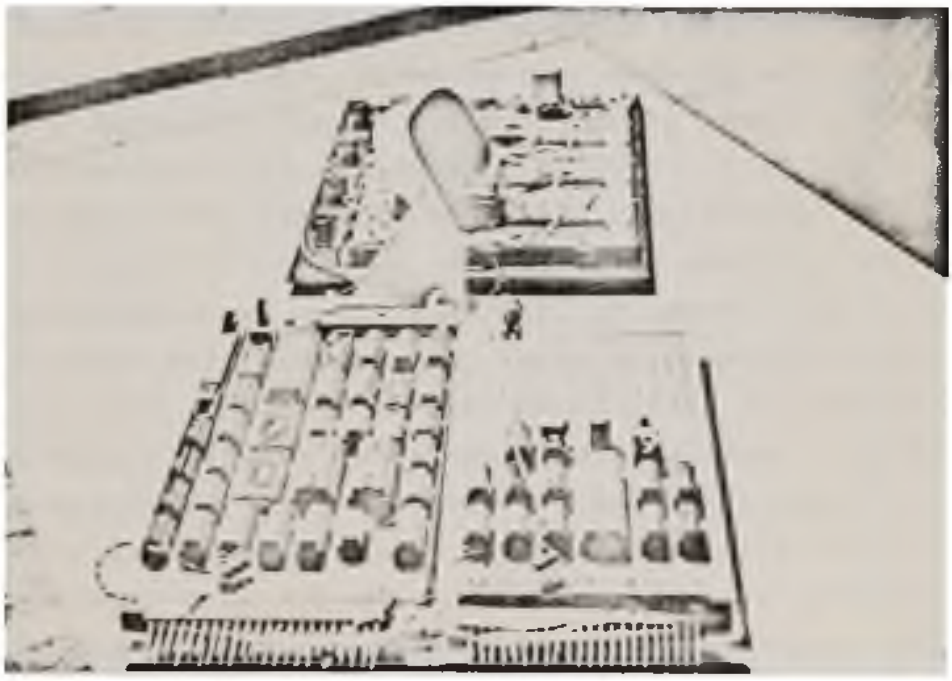


FIG. 4.2 EMULADOR PARA LOS MICROPROCESADORES 8086 Y 8088.

PRUEBAS EN EL EMULADOR

Las pruebas al emulador se realizaron con el microprocesador 8088 en configuración mínima. Se usó inicialmente, para probar la circuitería, el siguiente equipo: dos terminales no inteligentes; un analizador de estados lógicos, capaz de retener 256 palabras de 16 bits a partir de la llave de disparo; un sistema mínimo como prototipo de usuario (el sistema mínimo cuenta con un puerto serie basado en en circuito 8250); un osciloscopio de doble trazo, y multímetros. Posteriormente, se utilizó una computadora personal (compatible con IBM-PC) para probar la programación. Se hace notar que, en esta fase del trabajo, ya se pudo disponer del Grabador Universal de Memorias EPROM (parte del presente trabajo) para grabar los programas de prueba y el programa monitor del emulador.

Las principales pruebas efectuadas son:

- Prueba de "eco". Comprobar la correcta transmisión y recepción a través del puerto serie (8250).
- Prueba de la interfaz con el prototipo. Con una terminal conectada al puerto serie del prototipo y otra al emulador, se enviaron mensajes entre ellas. Los mensajes fueron leídos de una memoria EPROM, instalada en el prototipo.
- Prueba del comando de carga. La prueba se hizo cargando un archivo de disco a memoria RAM del prototipo, y a memoria RAM del emulador.
- Prueba del comando de lectura y modificación de memoria. En esta prueba se utilizó el comando de carga, y de esta forma se comprobó el funcionamiento correcto de ambos comandos (carga y lectura).
- Prueba del comando de lectura y modificación de puertos. Esta fué una prueba similar a la anterior.
- Prueba de lectura y modificación de registros. Esta parte consistió en modificar el contenido de los

registros del microprocesador, antes de ejecutar un programa, y durante la ejecución del mismo.

- Prueba de los comandos de ejecución. Inicialmente, se probó la ejecución de programas en modo interrumpido. Se comenzó con el comando de ejecución simple (paso a paso no automático). Posteriormente, se probaron los comandos de ejecución automática y por bloque. Por último, se probo el comando de ejecución en tiempo real (modo ininterrumpido). La prueba de estos cuatro comandos se realizo con programas residentes en memoria RAM y EPROM, tanto del emulador como del prototipo.
- Prueba del comando para facilitar el reloj del emulador. Esta prueba se realizó con la ayuda de un osciloscopio para verificar la conmutación de reloj prototipo a reloj emulador. Para asegurar la prueba se desconectó el reloj del prototipo.
- Finalmente, se probó el comando de inicialización (reset). La verificacion se realizo leyendo el contenido de los registros del microprocesador, antes y después de ejecutar este comando.

RESULTADOS DEL EMULADOR

Se considera que los resultados obtenidos hasta ahora, bajo una configuración mínima con el microprocesador 8088, son satisfactorios en cuanto a la velocidad de emulación, la cual esta determinada basicamente por el tiempo de despliegue de información en la pantalla. Ya que, entre cada instruccion ejecutada, del programa prototipo , existe un tiempo necesario para desplegar el contenido de los registros internos del microprocesador. Este tiempo es significantemente mayor que el requerido por cada instrucción del prototipo. Por lo que respecta a los comandos: cargar archivo; leer y modificar memoria, puertos, y registros, se puede afirmar que son de alta confiabilidad. La proteccion del emulador contra daños por fallas o malas conexiones en el prototipo, esta asegurada por medio de la interfaz del prototipo.

C A P I T U L O V

CONCLUSIONES

CONCLUSIONES

Durante varios años, principalmente con el advenimiento de las computadoras personales, ha existido en nuestro medio una mayor tendencia a desarrollar aplicaciones en el área de programación, con respecto al área de circuitería. Esto se explica por los altos costos de inversión que implica un buen laboratorio de sistemas digitales, y por las carencias de tecnología propia en este campo. Por ello, es importante desarrollar aplicaciones en el área de sistemas digitales, que no solo sean útiles por sí mismas, sino que además, nos sirvan como herramientas para desarrollar nuevas y más complejas aplicaciones.

El presente trabajo pretende coadyuvar al desarrollo de tecnología propia, en el área de sistemas digitales. El logro obtenido hasta ahora no solo se circunscribe al hecho de haber construido una valiosa herramienta de apoyo para los diseñadores de sistemas digitales; también se ha logrado adquirir experiencia, la cual es trasladable al diseño de sistemas de desarrollo orientados a otros microprocesadores.

Dadas las condiciones en las que este trabajo está ligado con otros del lugar en que fué desarrollado, se ha obtenido una formación profesional por parte del autor, así como de las personas que de alguna forma tuvieron relación con el proyecto. Cabe mencionar que las actividades realizadas durante el desarrollo del presente trabajo fueron muy diversas. Se estudiaron los microprocesadores Z-80, 8088 y 8085, con sus respectivos ensambladores. Se estudiaron los lenguajes C y PLM-86. Se estudiaron técnicas de programación de memorias EPROM, y aún queda por buscar nuevos métodos de emulación mas flexibles que el mostrado en este trabajo.

Las principales ventajas de este producto son: alta competitividad debido a sus caracteristicas técnicas y su bajo costo, y la filosofia de modularidad bajo la cual fue concebido. Su diseño esta contemplado para formar parte de un equipo modular de desarrollo, para varios tipos de microprocesadores, basado en una tarjeta común y tarjetas opcionales (una tarjeta para cada tipo de microprocesador).

Su principal justificación radica no solo en su bajo costo y la posibilidad de ser reproducido tantas veces como sea necesario en nuestras instituciones de docencia e investigacion. El trabajo tambien se justifica por la posibilidad que tiene de ser comercializado, dando asi la alternativa de evitar la importación de este tipo de productos, y ayudar a mejorar el nivel de vida de quienes lo elaboren en nuestro pais.

B I B L I O G R A F I A

Intel Corporation.
Memory Components Handbook.
1985.

Steve Ciarcia.
Build a Serial Eprom Programmer.
Byte, february 1985.

B & C Microsystems.
1409 Eprom Programmer Technical Manual.
Version 3.3.
September 1984.

Octavio F. Garcia Narcia.
Microprocesadores Z-80 e Interfaces.
Marzo 1982.

Kline B., Maerz M., Rosendfeld F.
The In-Circuit Approach to the Development of
Microcomputer-Based Products.
IEEE, June 1976.

Maldonado A., Vega J. I., Yañez V.
Emulador de Sistemas Digitales.
IV Coloquio de Control Automático. CINVESTAV-IPN.
Septiembre 1985.

Intel Corporation.
Microsystem Components Handbook.
Microprocessors and Peripherals.
Volume I and II, 1985.

Intel Corporation.
The 8086 Family User's Manual.
1980.

Intel Corporation.
PL/M-86 User's Guide.
1981.

Western Digital Corporation.
Communication Products Handbook.
June, 1984.

A P E N D I C E A

DIAGRAMA ELECTRICO DEL GRABADOR

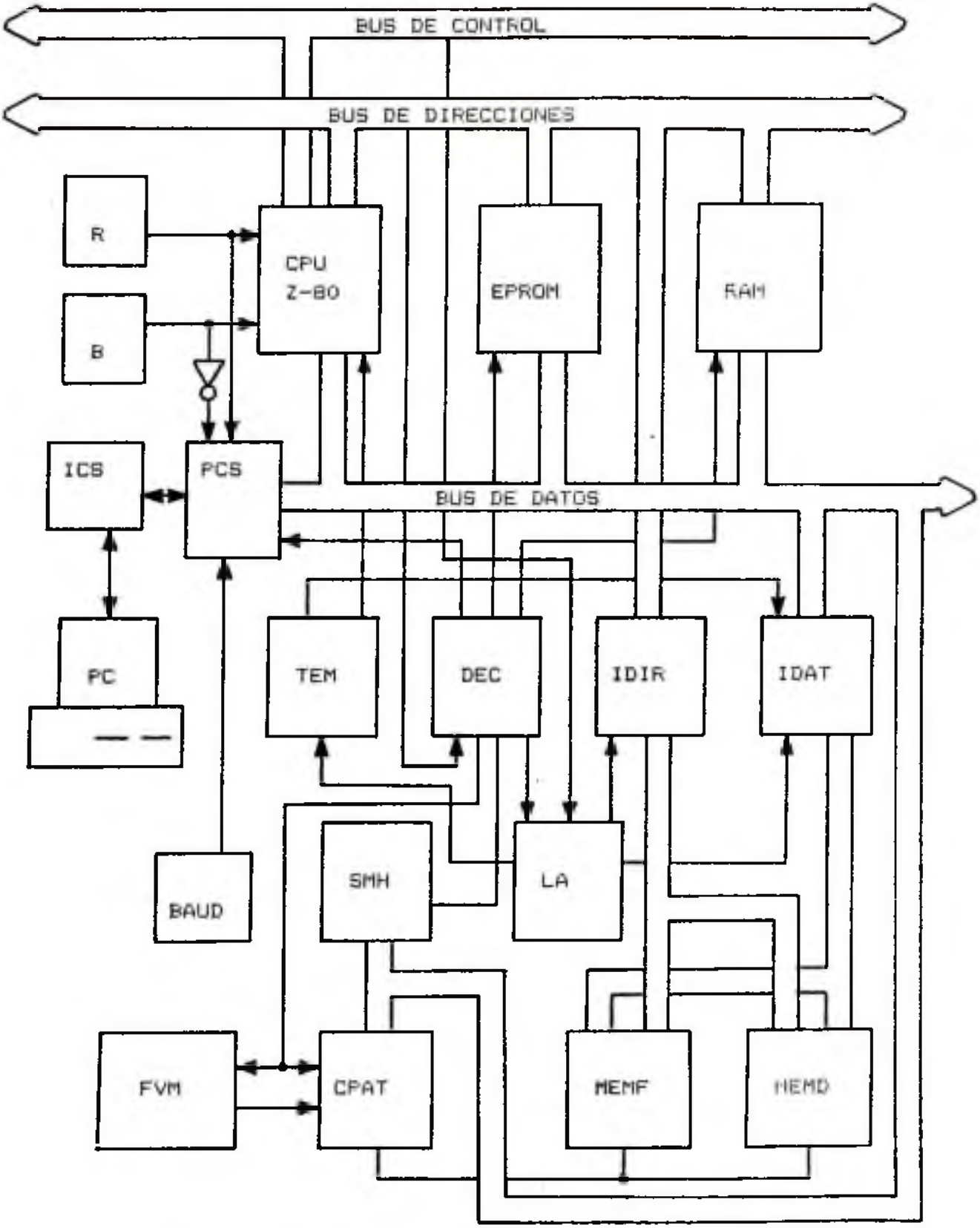
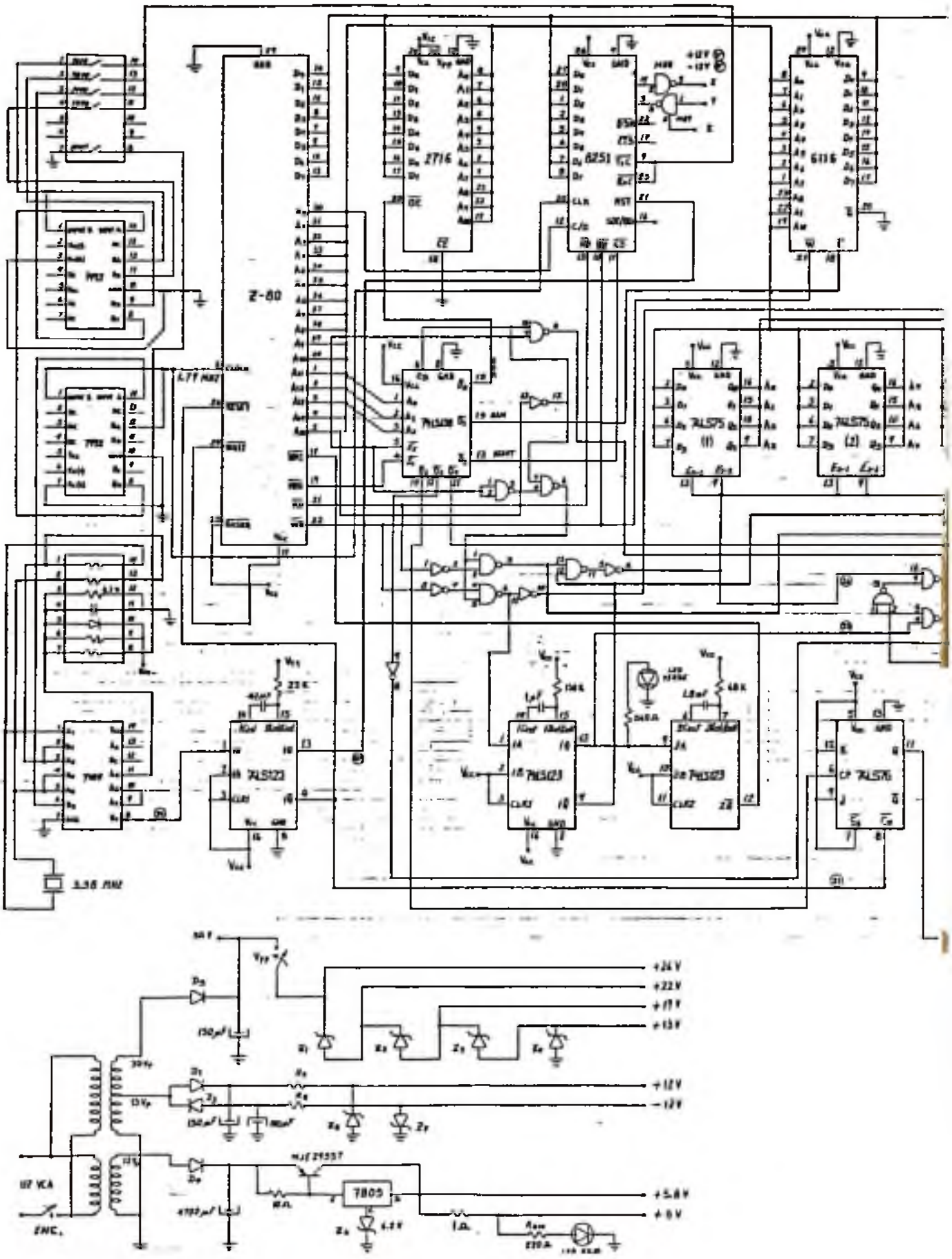


FIG. A.1 ESTRUCTURA GENERAL DEL MODULO DIGITAL DEL GRABADOR

IDENTIFICACION DE BLOQUES DEL GRABADOR

R - Circuito de Reloj
B - Circuito de Borrado (Reset)
CPU - Microprocesador Z-80
EPROM- Memoria de Lectura con el Programa Monitor
RAM - Memoria de Lectura/Escritura para los Datos
ICS - Interfaz de Comunicacion Serie
PCS - Puerto de Comunicación Serie
PC - Computadora Personal
TEM - Circuito Temporizador de Grabado
DEC - Circuito de Decodificacion de Memorias , Puertos
IDIR - Interfaz de Direcciones
IDAT - Interfaz de Datos
BAUD - Adaptador de Velocidad
SMH - Circuito Selector de Memorias por "Hardware"
LA - Lógica de Activacion
FVM - Fuente de Voltaje Múltiple
CPAT - Conmutador de Patrones de Voltaje
MEMF - Receptáculo para la Memoria Fuente
MEMD - Receptaculo para la Memoria Destino


Z-80
2716
8251
6116
74LS138
74LS75
74LS123
74LS76
7805
+24V
+22V
+17V
+13V
+12V
-12V
+5.8V
+5V

A P E N D I C E B

DIAGRAMA ELECTRICO DEL EMULADOR

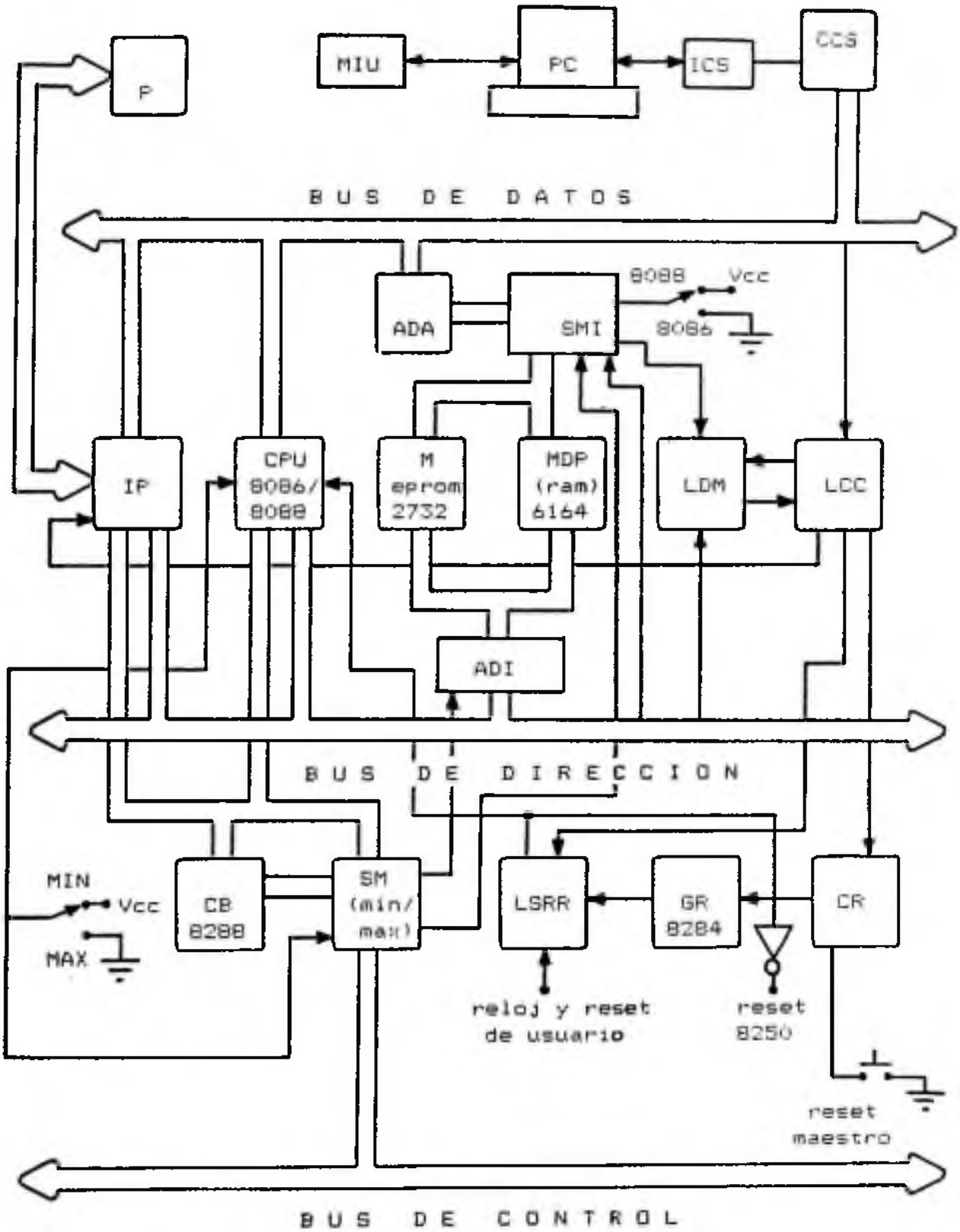
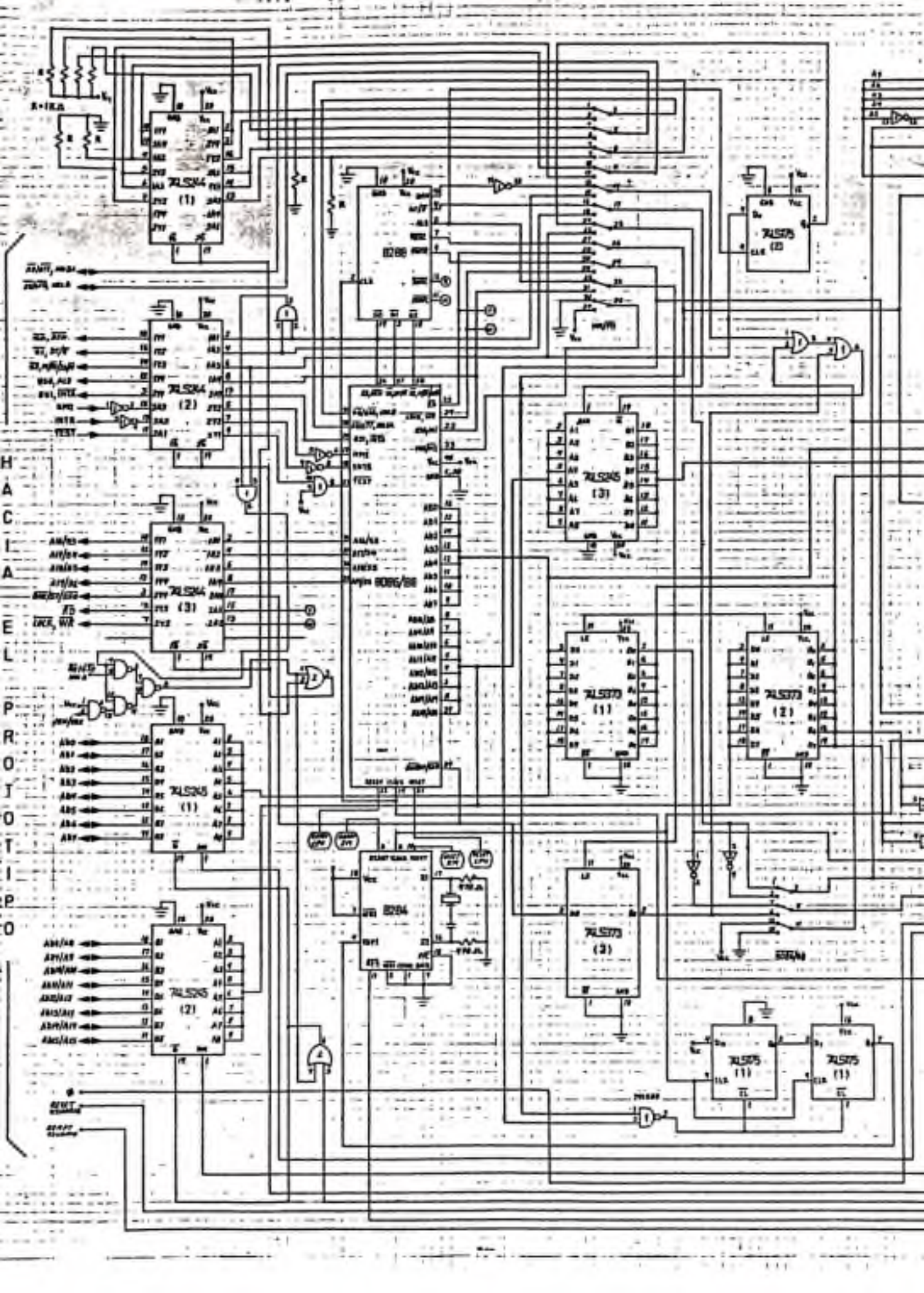
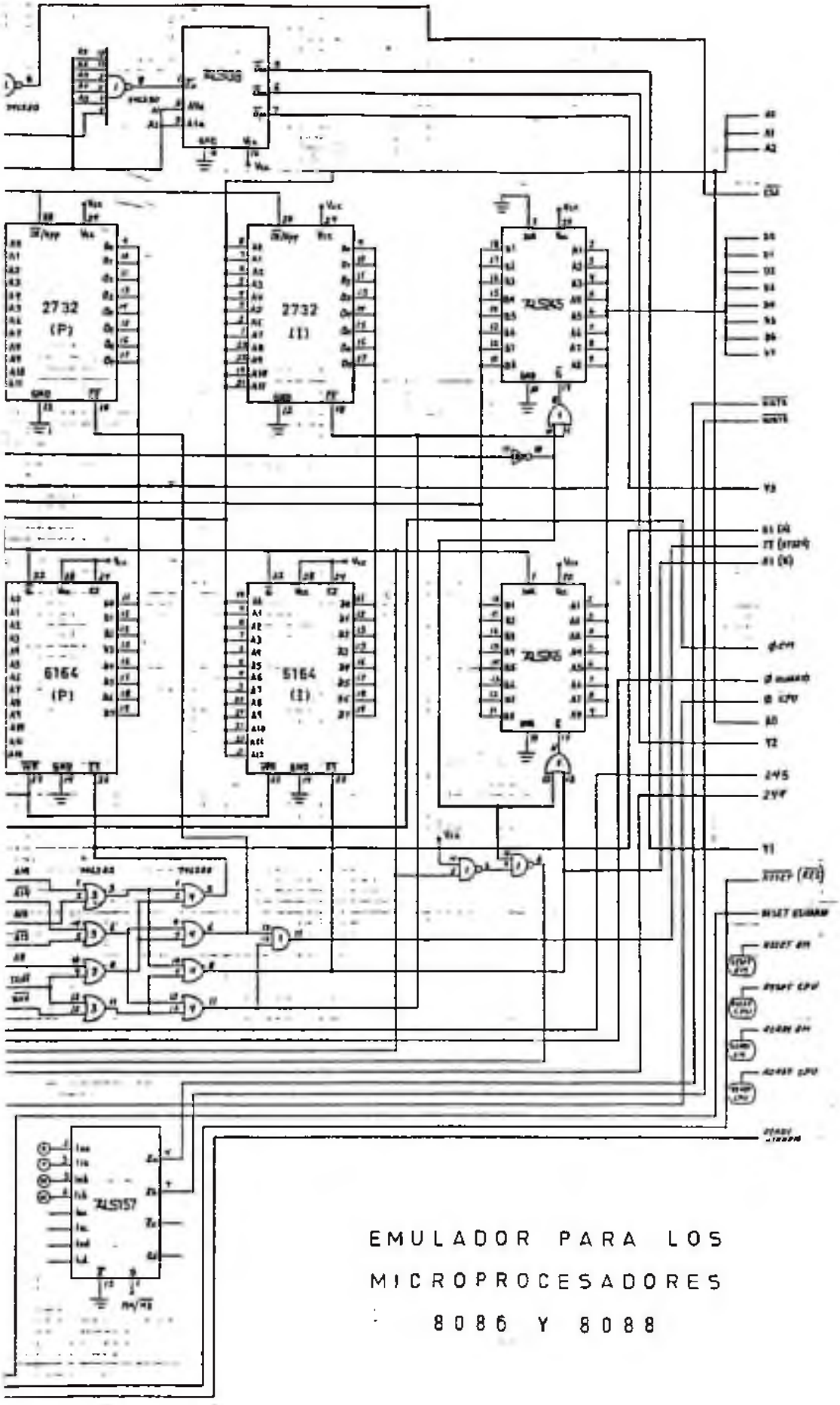


FIG. B.1 ESTRUCTURA GENERAL DEL MODULO DIGITAL DEL EMULADOR

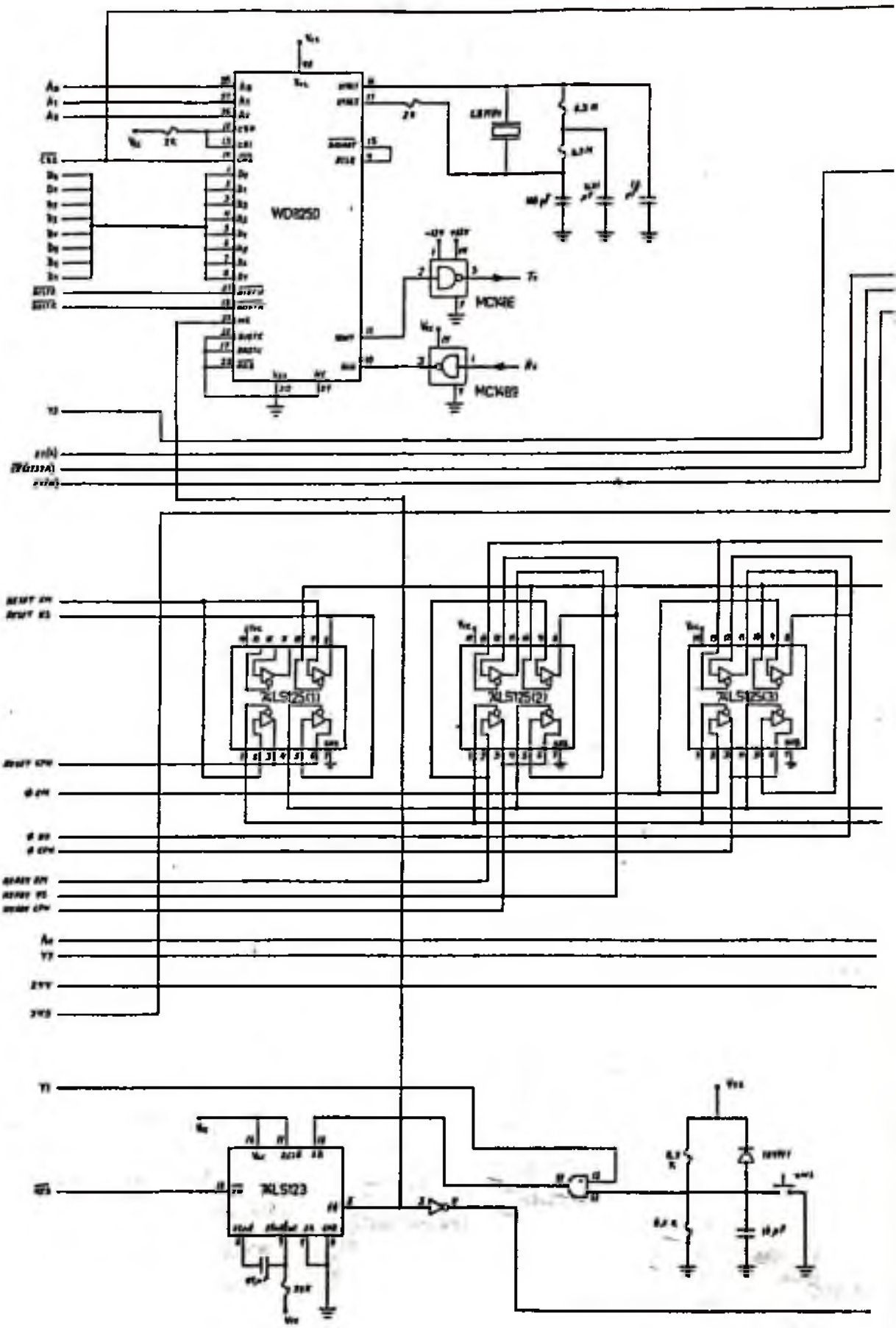
IDENTIFICACION DE BLOQUES DEL EMULADOR

P - Prototipo del Usuario.
MIU - Módulo de Interacción con el Usuario.
PC - Microcomputadora Personal.
ICS - Interfaz de Comunicación Serie.
CCS - Controlador de Comunicación Serie (C.I. 8250).
ADA - Amortiguador (buffer) de Datos.
SMI - Selector de Microprocesador e Interpretador de Datos.
IP - Interfaz con el Prototipo.
CPU - Microprocesador 8086 u 8088.
M - Memoria EPROM para el Monitor del Modulo Digital.
MDP - Memoria RAM para los Datos del Módulo Digital y el Programa Prototipo.
LDM - Lógica para Decodificación de Memoria.
LCC - Lógica de Control de Comandos.
ADI - Amortiguador (buffer) de Direcciones.
CB - Controlador de Bus.
SM - Selector de Modo (modo mínimo o máximo).
LSRR- Lógica de Selección de Reloj y Reset.
GR - Generador de Reloj.
CR - Circuito de Inicialización (RESET)


HACIA EL PROTIPO
74LS244 (1)
74LS244 (2)
74LS244 (3)
74LS245 (1)
74LS245 (2)
74LS245 (3)
74LS373 (1)
74LS373 (2)
74LS373 (3)
74LS375 (1)
74LS375 (2)
74LS375 (3)
8288
8284



EMULADOR PARA LOS
MICROPROCESADORES
8086 Y 8088


WD8250
MC1488
MC1489
74LS125(1)
74LS125(2)
74LS125(3)
74LS123

A P E N D I C E C

CODIFICACION DEL PROGRAMA MONITOR DEL GRABADOR

```
0000:          ; ****************************************
0000:          ; *  PROGRAMA MONITOR DEL GRABADOR UNIVERSAL  *
0000:          ; *          DE MEMORIAS EPROM                *
0000:          ; ****************************************

0000:                        .ORG $0000

0000:          PMODO     EQU 0CAh        ;Palabra de modo del 8251.
0000:          PCOM      EQU 015h        ;Palabra comando del 8251.
0000:          PRES      EQU 055h        ;Palabra de reset por SW.
0000:          PCONT     EQU 1001h       ;Reg. control (map. en m.)
0000:          PDAT      EQU 1000h       ;Reg. datos (map. en m.).
0000:          COMA      EQU 0FFEh       ;Dir. de comando G o L.
0000:          DIR       EQU 0FD0h       ;Dir. dato (DIR. INIC.).
0000:          NUBYTE    EQU 0FE0h       ;Dir. dato (NUM. BYTES).
0000:          TIPO      EQU 0FC0h       ;Dir. dato (TIPO EPROM).
0000:          LIMIB     EQU 0901h       ;Limite inferior buffer.
0000:          LIMSB     EQU 0FB0h       ;Limite superior buffer.
0000:          FLOF      EQU 0FF0h       ;Direccion de bandera XOF.
0000:          BYTES     EQU 0FF2h       ;Num. de bytes por grabar.
0000:          FUNO      EQU 0FF4h       ;Dir. band. primer grab.
0000:          BUFTOT    EQU 06AFh       ;Tamano total del buffer.
0000:          TBUF      EQU 000Fh       ;Tamano minimo del buffer.
0000:          DIRDES    EQU 0FD2h       ;Dir. dato (DIR. DESTINO).
0000:          FLR       EQU 0FF6h       ;Bandera Local/Remoto.
0000:          INT       EQU 0FF8h       ;Bandera de interrupcion.
0000:          ;
0000: 00                 NOP             ;Retardo mientras se
0001: 00                 NOP             ;estabiliza el ptc. 8251.
0002: 00                 NOP
0003: 3E CA              LD A,PMODO      ;Carga palabra de modo.
0005: 32 01 10           LD (PCONT),A    ;Envia por reg. de control.
0008: 3E 55              LD A,PRES       ;Carga y
000A: 32 01 10           LD (PCONT),A    ;envia palabra de reset.
000D: 3E CA              LD A,PMODO      ;Carga y
000F: 32 01 10           LD (PCONT),A    ;envia palabra de modo.
0012: 3E 15              LD A,PCOM       ;Carga y
0014: 32 01 10           LD (PCONT),A    ;envia palabra de comando.
0017: 31 FF 08           LD SP,08FFh     ;Inic. apunt. de stack.
001A: 3E 00              LD A,00h        ;Pone en ceros
001C: 32 00 18           LD (1800h),A    ;el patron.
001F: 32 00 18           LD (1800h),A
0022: 3A 00 20 REMOTO    LD A,(2000h)    ;Lee interruptores.
0025: E6 1F              AND 1Fh         ;Aisla los 5 mas bajos.
0027: 4F                 LD C,A          ;Salva el resultado.
0028: FE 00              CP 00h          ;Pregunta si son ceros.
002A: C2 6D 02           JP NZ,COPIAR    ;Va a COPIAR si no son.
002D: 3E FF              LD A,0FFh       ;Pone band. Local/Remoto
002F: 32 F6 0F           LD (FLR),A      ;en modo remoto (Grab/lee)
0032: CD EA 02           CALL RX
0035: 3A 00 10           LD A,(PDAT)     ;Espera caracter "Y" como
0038: 1E 59              LD E,59h        ;senal de requerimiento.
003A: BB                 CP E            ;Si no hay requerimiento
003B: C2 22 00           JP NZ,REMOTO    ;regresa a leer interr.
003E: CD F2 02           CALL TX         ;Transmite un caracter
0041: 3E 58              LD A,58h        ;"X" como senal de
0043: 32 00 10           LD (PDAT),A     ;reconocimiento.
```

```
 58. 0046: CD EA 02              CALL RX          ;Recibe un comando
 59. 0049: 3A 00 10              LD A,(PDAT)      ;G o L para grabar
 60. 004C: 32 FE 0F              LD (COMA),A      ;o leer, respectivamente,
 61. 004F: CD F2 02              CALL TX          ;y tranmite un eco.
 62. 0052: 3A FE 0F              LD A,(COMA)
 63. 0055: 32 00 10              LD (PDAT),A
 64. 0058: C3 FD 00              JP 00FDh         ;Continua en la dir. 00FD.
 65. 005B:                  ;
 66. 00FD:                  .ORG $00FD
 67. 00FD:                  ;
 68. 00FD: 21 D0 0F              LD HL,DIR        ;Carga HL con DIR.
 69. 0100: 06 04                 LD B,04h         ;Inic. un contador de 4
 70. 0102: CD EA 02    LEE1      CALL RX          ;Recibe 4 digitos
 71. 0105: CD 03 03              CALL METE        ;correspondientes
 72. 0108: 23                    INC HL           ;a la DIRECCION
 73. 0109: 10 F7                 DJNZ LEE1        ;INICIAL.
 74. 010B: 21 D0 0F              LD HL,DIR        ;Carga HL con Dir
 75. 010E: CD 21 03              CALL AGRUPA      ;Agru. 4 dig. en 2 bytes.
 76. 0111: 21 E0 0F              LD HL,NUBYTE     ;Carga HL con NUBYTE.
 77. 0114: 06 04                 LD B,04h         ;Inic. contador de 4.
 78. 0116: CD EA 02    LEE2      CALL RX          ;Recibe 4 digitos
 79. 0119: CD 03 03              CALL METE        ;correspondientes
 80. 011C: 23                    INC HL           ;al NUMERO DE
 81. 011D: 10 F7                 DJNZ LEE2        ;BYTES.
 82. 011F: 21 E0 0F              LD HL,NUBYTE     ;Carga HL con NUBYTE.
 83. 0122: CD 21 03              CALL AGRUPA      ;Agru. 4 dig. en 2 bytes.
 84. 0125: CD EA 02              CALL RX          ;Recibe TIPO
 85. 0128: 3A 00 10              LD A,(PDAT)      ;DE EPROM
 86. 012B: 32 C0 0F              LD (TIPO),A      ;y transmite
 87. 012E: CD F2 02              CALL TX          ;un eco.
 88. 0131: 3A C0 0F              LD A,(TIPO)
 89. 0134: 32 00 10              LD (PDAT),A
 90. 0137: CD EA 02              CALL RX          ;Por seguridad,
 91. 013A: 3A 00 10              LD A,(PDAT)      ;recibe orden p/continuar
 92. 013D: FE 53                 CP 53h           ;si recibe "S" continua
 93. 013F: C2 22 00              JP NZ,REMOTO     ;si no, regresa a REMOTO.
 94. 0142: 3A FE 0F              LD A,(COMA)      ;Si el comando recibido
 95. 0145: FE 4C                 CP 4Ch           ;no es "L", salta a
 96. 0147: C2 4F 01              JP NZ,GENPG      ;generar patron de grab.
 97. 014A: 26 07                 LD H,07h         ;si es "L", salta a
 98. 014C: C3 51 01              JP GENPL         ;generar patron de lect.
 99. 014F: 26 06         GENPG   LD H,06h         ;Genera patron de grab.
100. 0151: 3A C0 0F      GENPL   LD A,(TIPO)      ;Genera patron de lect.
101. 0154: 6F                    LD L,A
102. 0155: 7E                    LD A,(HL)
103. 0156: 32 00 18              LD (1800h),A
104. 0159: 3A FE 0F              LD A,(COMA)      ;Pregunta si el comando
105. 015C: 16 47                 LD D,47h         ;es "G" para continuar
106. 015E: BA                    CP D             ;o "L" para ir a LEER.
107. 015F: C2 28 02              JP NZ,LEER
108. 0162: 21 E0 0F              LD HL,NUBYTE     ;Carga el
109. 0165: 46                    LD B,(HL)        ;numero de
110. 0166: 23                    INC HL           ;datos a grabar.
111. 0167: 4E                    LD C,(HL)
112. 0168: ED 43 F2 0F           LD (BYTES),BC
113. 016C: FD 21 01 09           LD IY,LIMIB      ;Inicializa apunt. AP1 y
114. 0170: DD 21 01 09           LD IX,LIMIB      ;AP2 (grabar y guardar).
```

```
115. 0174: 3E 00                    LD A,00h
116. 0176: 32 F4 0F                 LD (FUNO),A     ;Inicializa banderas.
117. 0179: 32 F0 0F                 LD (FLOF),A
118. 017C: 3A D0 0F                 LD A,(0FD0h)    ;Suma un 8000h como
119. 017F: 67                       LD H,A          ;cero relativo a
120. 0180: 3A D1 0F                 LD A,(0FD1h)    ;la direccion inicial.
121. 0183: 6F                       LD L,A
122. 0184: 01 00 80                 LD BC,8000h
123. 0187: 09                       ADD HL,BC
124. 0188: 22 D2 0F                 LD (DIRDES),HL ;Salva direccion inicial.
125. 018B: CD 39 03     GRABAR      CALL DIGITO    ;Recibe 2 caracteres
126. 018E: 07                       RLCA           ;ascii y los integra en
127. 018F: 07                       RLCA           ;un solo byte, el cual
128. 0190: 07                       RLCA           ;corresponde al dato a
129. 0191: 07                       RLCA           ;grabar.
130. 0192: 47                       LD B,A
131. 0193: CD 39 03                 CALL DIGITO
132. 0196: 80                       ADD A,B
133. 0197: FD 77 00                 LD (IY+00h),A  ;Carga el dato en AP2.
134. 019A: ED 4B F2 0F              LD BC,(BYTES)  ;Pregunta si num. de
135. 019E: 79                       LD A,C         ;bytes es cero para
136. 019F: B0                       OR B           ;determinar fin
137. 01A0: CA 22 00                 JP Z,REMOTO    ;de archivo.
138. 01A3: 0B                       DEC BC         ;Decrementa num. de bytes
139. 01A4: ED 43 F2 0F              LD (BYTES),BC  ;y actualiza apunt. AP2
140. 01A8: FD 23                    INC IY
141. 01AA: 3A F4 0F                 LD A,(FUNO)    ;Pregunta si ya se grabo
142. 01AD: B7                       OR A           ;el primer dato para
143. 01AE: C2 C1 01                 JP NZ,ETFLOF   ;no grabar.
144. 01B1: 2A D2 0F                 LD HL,(DIRDES)
145. 01B4: DD 7E 00                 LD A,(IX+00h)  ;Graba el primer dato,
146. 01B7: 77                       LD (HL),A      ;actualiza la direccion
147. 01B8: 23                       INC HL         ;destino, e indica con la
148. 01B9: 22 D2 0F                 LD (DIRDES),HL ;bandera FUNO que ya se
149. 01BC: 3E FF                    LD A,0FFh      ;grabo el primer dato.
150. 01BE: 32 F4 0F                 LD (FUNO),A
151. 01C1: 3A F0 0F     ETFLOF      LD A,(FLOF)    ;Pregunta si ha recibido
152. 01C4: B7                       OR A           ;XOF para ir a CHAP2L.
153. 01C5: C2 FB 01                 JP NZ,CHAP2L
154. 01C8: DD E5                    PUSH IX        ;Pregunta si AP1 es
155. 01CA: E1                       POP HL         ;mayor que AP2 para
156. 01CB: FD E5                    PUSH IY        ;saltar a ET6.
157. 01CD: D1                       POP DE
158. 01CE: 37                       SCF
159. 01CF: 3F                       CCF
160. 01D0: ED 52                    SBC HL,DE
161. 01D2: D2 11 02                 JP NC,ET6
162. 01D5: 21 AF 06                 LD HL,BUFTOT   ;Pregunta si hay area
163. 01D8: 37                       SCF            ;disponible en el buffer
164. 01D9: 3F                       CCF            ;para guardar datos,
165. 01DA: ED 52                    SBC HL,DE      ;y salta a CHAP2L cuando
166. 01DC: DD E5                    PUSH IX        ;si la hay o de lo
167. 01DE: C1                       POP BC         ;contrario va a XOF
168. 01DF: 09                       ADD HL,BC      ;para detener el flujo
169. 01E0: E5                       PUSH HL        ;de datos.
170. 01E1: 21 0F 00                 LD HL,TBUF
171. 01E4: E5                       PUSH HL
```

```
172. 01E5: D1                        POP DE
173. 01E6: E1                        POP HL
174. 01E7: 37                        SCF
175. 01E8: 3F                        CCF
176. 01E9: ED 52                     SBC HL,DE
177. 01EB: D2 FB 01                  JP NC,CHAP2L
178. 01EE: CD F2 02     XOF          CALL TX          ;Transmite un XOF y
179. 01F1: 3E 13                     LD A,13h         ;pone bandera de XOF
180. 01F3: 32 00 10                  LD (FDAT),A      ;en verdadero.
181. 01F6: 3E FF                     LD A,0FFh
182. 01F8: 32 F0 0F                  LD (FLOF),A
183. 01FB: 21 B0 0F     CHAP2L       LD HL,LIMSB      ;Checa si el apuntador
184. 01FE: FD E5                     PUSH IY          ;AP2 (IY) ya llego al
185. 0200: C1                        POP BC           ;limite para regresarlo
186. 0201: 37                        SCF              ;al origen. De lo
187. 0202: 3F                        CCF              ;contrario regresa
188. 0203: ED 42                     SBC HL,BC        ;a GRABAR.
189. 0205: 7D                        LD A,L
190. 0206: B4                        OR H
191. 0207: C2 8B 01                  JP NZ,GRABAR
192. 020A: FD 21 01 09               LD IY,LIMIB
193. 020E: C3 8B 01                  JP GRABAR
194. 0211: DD E5        ET6          PUSH IX          ;Pregunta si hay area
195. 0213: E1                        POP HL           ;disponible en el buffer
196. 0214: FD E5                     PUSH IY          ;para guardar datos,
197. 0216: D1                        POP DE           ;y salta a CHAP2L cuando
198. 0217: 37                        SCF              ;si la hay o de lo
199. 0218: 3F                        CCF              ;contrario salta a XOF
200. 0219: ED 52                     SBC HL,DE        ;para detener el flujo
201. 021B: 11 0F 00                  LD DE,TBUF       ;de datos.
202. 021E: 37                        SCF
203. 021F: 3F                        CCF
204. 0220: ED 52                     SBC HL,DE
205. 0222: D2 FB 01                  JP NC,CHAP2L
206. 0225: C3 EE 01                  JP XOF
207. 0228: 3A D0 0F     LEER         LD A,(0FD0h)     ;Suma a la direccion
208. 022B: C6 80                     ADD A,80h        ;inicial un 8000h como
209. 022D: 67                        LD H,A           ;cero relativo. y la
210. 022E: 3A D1 0F                  LD A,(0FD1h)     ;salva en el stack
211. 0231: 6F                        LD L,A
212. 0232: E5                        PUSH HL
213. 0233: 21 E0 0F                  LD HL,NUBYTE     ;Carga NUBYTES (numero
214. 0236: 56                        LD D,(HL)        ;total de bytes), en
215. 0237: 23                        INC HL           ;BYTES (numero de
216. 0238: 5E                        LD E,(HL)        ;bytes que quedan por
217. 0239: ED 53 F2 0F               LD (BYTES),DE    ;leer o grabar.
218. 023D: E1                        POP HL
219. 023E: 7E           LOOPLE       LD A,(HL)        ;lee un dato (byte),
220. 023F: 4F                        LD C,A           ;lo convierte a dos
221. 0240: E6 F0                     AND 0F0h         ;caracteres ascii,
222. 0242: 0F                        RRCA             ;los transmite a la
223. 0243: 0F                        RRCA             ;computadora, y
224. 0244: 0F                        RRCA             ;espera un eco.
225. 0245: 0F                        RRCA
226. 0246: CD 57 03                  CALL HEXASC
227. 0249: 47                        LD B,A
228. 024A: CD F2 02                  CALL TX
```

```
229. 024D: 78                        LD A,B
230. 024E: CD 61 03                  CALL TRECO
231. 0251: 79                        LD A,C
232. 0252: E6 0F                     AND 0Fh
233. 0254: CD 57 03                  CALL HEXASC
234. 0257: 47                        LD B,A
235. 0258: CD F2 02                  CALL TX
236. 025B: 78                        LD A,B
237. 025C: CD 61 03                  CALL TRECO
238. 025F: 23                        INC HL          ;Incrementa direccion y
239. 0260: 1B                        DEC DE          ;decrementa num. bytes.
240. 0261: ED 53 F2 0F               LD (BYTES),DE
241. 0265: 7B                        LD A,E          ;Si num. de bytes = 0
242. 0266: B2                        OR D            ;salta a REMOTO. Si no
243. 0267: CA 22 00                  JP Z,REMOTO     ;es 0 salta a LOOPLE.
244. 026A: C3 3E 02                  JP LOOPLE
245. 026D: CB 61          COPIAR     BIT 4,C         ;Si es memoria
246. 026F: 20 03                     JR NZ,COPI      ;27256, salta a COPI.
247. 0271: 32 00 28                  LD (2800h),A    ;si no, prende PGM'.
248. 0274: 3E 00          COPI       LD A,00h        .
249. 0276: 32 F6 0F                  LD (FLR),A      ;limpia banderas de
250. 0279: 32 F8 0F                  LD (INT),A      ;loc/rem. e interrup.
251. 027C: DD 21 00 40               LD IX,4000h     ;Inicializa apuntadores
252. 0280: FD 21 00 80               LD IY,8000h     ;fuente y destino.
253. 0284: 06 06                     LD B,06h        ;Carga en DE la
254. 0286: 0A                        LD A,(BC)       ;capacidad de la
255. 0287: 57                        LD D,A          ;memoria a copiar.
256. 0288: 1E 00                     LD E,00h
257. 028A: 05                        DEC B           ;Genera patron de
258. 028B: 0A                        LD A,(BC)       ;senales por HW.
259. 028C: 32 00 18                  LD (1800h),A
260. 028F: 3A 00 20       LOOPCO1    LD A,(2000h)    ;Lee interruptores
261. 0292: E6 1F                     AND 1Fh         ;mientras se enciende
262. 0294: B7                        OR A            ;Vpp y hasta que se
263. 0295: 20 F8                     JR NZ,LOOPCO1   ;remueve la seleccion.
264. 0297: 06 FF                     LD B,0FFh
265. 0299: 3E 0F          LOOPCO2    LD A,0Fh        ;por seguridad,
266. 029B: 3D             LOOPCO3    DEC A           ;se introduce un retardo
267. 029C: 20 FD                     JR NZ,LOOPCO3   ;y se verifica que todos
268. 029E: 10 F9                     DJNZ LOOPCO2    ;los interruptores esten
269. 02A0: 3A 00 20                  LD A,(2000h)    ;removidos antes de
270. 02A3: E6 1F                     AND 1Fh         ;iniciar el copiado.
271. 02A5: B7                        OR A
272. 02A6: 20 E7                     JR NZ,LOOPCO1
273. 02A8: 3A F8 0F       LOOPCO     LD A,(INT)      ;Espera que se termine
274. 02AB: B7                        OR A            ;de grabar un dato.
275. 02AC: 20 FA                     JR NZ,LOOPCO
276. 02AE: DD 7E 00                  LD A,(IX+00h)   ;Graba dato de fuente
277. 02B1: FD 77 00                  LD (IY+00h),A   ;a destino.
278. 02B4: 3E FF                     LD A,0FFh
279. 02B6: 32 F8 0F                  LD (INT),A      ;Pone bandera de Grab.
280. 02B9: DD 23                     INC IX          ;Incrementa direcciones
281. 02BB: FD 23                     INC IY          ;fuente y destino.
282. 02BD: 1B                        DEC DE          ;Decrementa capacidad
283. 02BE: 7B                        LD A,E          ;de memoria y pregunta
284. 02BF: B2                        OR D            ;si llego a cero para
285. 02C0: 28 02                     JR Z,FINCOP     ;salir del loop.
```

```
[illegible]: 02C2: 18 E4                    JR LOOPCO
[illegible]: 02C4: CB 61        FINCOP      BIT 4,C             ;Si no es tipo 27256,
[illegible]: 02C6: 28 29                    JR Z,TERMINO        ;va a TERMINO.
[illegible]: 02C8: 06 06                    LD B,06h            ;Si es memoria 27256.
[illegible]: 02CA: 0A                       LD A,(BC)           ;carga nuevamente
[illegible]: 02CB: 57                       LD D,A              ;la capacidad de la
[illegible]: 02CC: 1E 00                    LD E,00h            ;memoria (27128).
[illegible]: 02CE: DD 21 00 40              LD IX,4000h         ;reinicia cont. fuente
[illegible]: 02D2: 32 00 28                 LD (2800h),A        ;y prende bit A14 fnte.
[illegible]: 02D5: 3A F8 0F     LOOPCO4     LD A,(INT)          ;Espera que se termine
[illegible]: 02D8: B7                       OR A                ;de grabar un dato.
[illegible]: 02D9: 20 FA                    JR NZ,LOOPCO4
[illegible]: 02DB: DD 7E 00                 LD A,(IX+00h)       ;Graba dato de fuente
[illegible]: 02DE: FD 77 00                 LD (IY+00h),A       ;a destino.
[illegible]: 02E1: 3E FF                    LD A,0FFh
[illegible]: 02E3: 32 F8 0F                 LD (INT),A          ;Pone band. de Grabando.
[illegible]: 02E6: 1B                       DEC DE              ;Decrementa capacidad
[illegible]: 02E7: 7B                       LD A,E              ;de memoria y pregunta
[illegible]: 02E8: B2                       OR D                ;si llego a cero para
[illegible]: 02E9: 28 06                    JR Z,TERMINO        ;terminar.
[illegible]: 02EB: DD 23                    INC IX              ;Incrementa direcciones
[illegible]: 02ED: FD 23                    INC IY              ;fuente y destino.
[illegible]: 02EF: 18 E4                    JR LOOPCO4
[illegible]: 02F1: 76           TERMINO     HALT                ;Termina copiado.
[illegible]: 02F2:              ;
[illegible]: 02F2:              ;
[illegible]: 02F2:              ; SUBRUTINA DE INTERRUPCION.
[illegible]: 02F2:              ;
[illegible]: 0066:                      .ORG $0066
[illegible]: 0066:
[illegible]: 0066: F5                       PUSH AF             ;Salvaguarda contexto.
[illegible]: 0067: C5                       PUSH BC
[illegible]: 0068: D5                       PUSH DE
[illegible]: 0069: E5                       PUSH HL
[illegible]: 006A: 3A F6 0F                 LD A,(FLR)          ;Pregunta si es de
[illegible]: 006D: B7                       OR A                ;copiado la subrutina.
[illegible]: 006E: CA F2 00                 JP Z,INTCOP
[illegible]: 0071: DD 23                    INC IX              ;Actualiza AP1.
[illegible]: 0073: 3A F0 0F                 LD A,(FLOF)         ;Pregunta si no ha
[illegible]: 0076: B7                       OR A                ;ocurrido un XOF
[illegible]: 0077: CA C4 00                 JP Z,CHAP1L         ;para saltar a CHAP1L.
[illegible]: 007A: DD E5                    PUSH IX             ;Pregunta si AP1 es
[illegible]: 007C: E1                       POP HL              ;mayor que AP2 para
[illegible]: 007D: FD E5                    PUSH IY             ;saltar a ET7
[illegible]: 007F: D1                       POP DE
[illegible]: 0080: 37                       SCF
[illegible]: 0081: 3F                       CCF
[illegible]: 0082: ED 52                    SBC HL,DE
[illegible]: 0084: D2 AD 00                 JP NC,ET7
[illegible]: 0087: 21 AF 06                 LD HL,BUFTOT        ;Pregunta si existe
[illegible]: 008A: 37                       SCF                 ;area disponible para
[illegible]: 008B: 3F                       CCF                 ;guardar datos en el
[illegible]: 008C: ED 52                    SBC HL,DE           ;buffer. Si la hay,
[illegible]: 008E: DD E5                    PUSH IX             ;continua en XON. De
[illegible]: 0090: C1                       POP BC              ;lo contrario, salta
[illegible]: 0091: 09                       ADD HL,BC           ;a CHAP1L.
[illegible]: 0092: E5                       PUSH HL
```

```
343. 0093: 21 0F 00              LD HL,TBUF
344. 0096: E5                    PUSH HL
345. 0097: D1                    POP DE
346. 0098: E1                    POP HL
347. 0099: 37                    SCF
348. 009A: 3F                    CCF
349. 009B: ED 52                 SBC HL,DE
350. 009D: DA C4 00              JP C,CHAP1L
351. 00A0: 3E 00         XON     LD A,00h          ;Pone band. XOF en
352. 00A2: 32 F0 0F              LD (FLOF),A       ;cero y envia un XON.
353. 00A5: 3E 4E                 LD A,4Eh
354. 00A7: 32 00 10              LD (PDAT),A
355. 00AA: C3 C4 00              JP CHAP1L         ;Salta a CHAP1L.
356. 00AD: DD E5         ET7     PUSH IX           ;Pregunta si hay
357. 00AF: E1                    POP HL            ;lugar para guardar
358. 00B0: FD E5                 PUSH IY           ;datos en el buffer
359. 00B2: D1                    POP DE            ;y saltar a XON en
360. 00B3: 37                    SCF               ;caso afirmativo,
361. 00B4: 3F                    CCF               ;o saltar a CHAP1L
362. 00B5: ED 52                 SBC HL,DE         ;en caso contrario.
363. 00B7: 11 0F 00              LD DE,TBUF
364. 00BA: 37                    SCF
365. 00BB: 3F                    CCF
366. 00BC: ED 52                 SBC HL,DE
367. 00BE: DA C4 00              JP C,CHAP1L
368. 00C1: C3 A0 00              JP XON
369. 00C4: 21 B0 0F      CHAP1L  LD HL,LIMSB       ;Checar si AP1 ya
370. 00C7: DD E5                 PUSH IX           ;llego al limite
371. 00C9: C1                    POP BC            ;para regresarlo
372. 00CA: 37                    SCF               ;al origen. De lo
373. 00CB: 3F                    CCF               ;contrario salta
374. 00CC: ED 42                 SBC HL,BC         ;a FINAR.
375. 00CE: 7D                    LD A,L
376. 00CF: B4                    OR H
377. 00D0: C2 D7 00              JP NZ,FINAR
378. 00D3: DD 21 01 09           LD IX,LIMIB
379. 00D7: DD E5         FINAR   PUSH IX           ;Compara AP1 y AP2,
380. 00D9: E1                    POP HL            ;si son iguales
381. 00DA: FD E5                 PUSH IY           ;significa que ya
382. 00DC: D1                    POP DE            ;termino de grabar
383. 00DD: 37                    SCF               ;y salta a AVISO.
384. 00DE: 3F                    CCF
385. 00DF: ED 52                 SBC HL,DE
386. 00E1: CA 70 03              JP Z,AVISO
387. 00E4: 2A D2 0F              LD HL,(DIRDES)    ;Manda a grabar el
388. 00E7: DD 7E 00              LD A,(IX+00h)     ;siguiente dato,
389. 00EA: 77                    LD (HL),A         ;incrementa direccion
390. 00EB: 23                    INC HL            ;destino, y salta
391. 00EC: 22 D2 0F              LD (DIRDES),HL    ;a RESTAU.
392. 00EF: C3 F7 00              JP RESTAU
393. 00F2: 3E 00         INTCOP  LD A,00h          ;Pone en 0 la bandera
394. 00F4: 32 F8 0F              LD (INT),A        ;de interrupcion.
395. 00F7: E1            RESTAU  POP HL            ;Restaura contexto
396. 00F8: D1                    POP DE            ;y sale de la
397. 00F9: C1                    POP BC            ;interrupcion.
398. 00FA: F1                    POP AF
399. 00FB: ED 45                 RETN
```

```
400. 00FD:                      ;
401. 00FD:                      ; SUBRUTINA PARA PROBAR EL BIT DE RECEPCION
402. 00FD:                      ;
403. 02EA:                              .ORG $02EA
404. 02EA:                      ;
405. 02EA: 3A 01 10             RX      LD A,(PCONT)
406. 02ED: CB 4F                        BIT 1,A
407. 02EF: 28 F9                        JR Z,RX
408. 02F1: C9                           RET
409. 02F2:                      ;
410. 02F2:                      ; SUBRUTINA PARA PROBAR EL BIT DE TRANSMISION
411. 02F2:                      ;
412. 02F2: 3A 01 10             TX      LD A,(PCONT)
413. 02F5: CB 47                        BIT 0,A
414. 02F7: 28 F9                        JR Z,TX
415. 02F9: C9                           RET
416. 02FA:                      ;
417. 02FA:                      ; SUBRUTINA PARA TRANSMITIR EL
418. 02FA:                      ;   CONTENIDO DEL ACUMULADOR
419. 02FA:                      ;
420. 02FA: 4F                   SACA    LD C,A
421. 02FB: CD F2 02                     CALL TX
422. 02FE: 79                           LD A,C
423. 02FF: 32 00 10                     LD (PDAT),A
424. 0302: C9                           RET
425. 0303:                      ;
426. 0303:                      ; SUBRUTINA PARA RECIBIR Y GUARDAR
427. 0303:                      ;       DIGITOS HEXADECIMALES
428. 0303:                      ;
429. 0303: E5                   METE    PUSH HL
430. 0304: 3A 00 10                     LD A,(PDAT)
431. 0307: 57                           LD D,A
432. 0308: CD F2 02                     CALL TX
433. 030B: 7A                           LD A,D
434. 030C: 32 00 10                     LD (PDAT),A
435. 030F: 0E 41                        LD C,41h
436. 0311: B9                           CP C
437. 0312: DA 1C 03                     JP C,ET1
438. 0315: 26 06                        LD H,06h
439. 0317: 6F                           LD L,A
440. 0318: 7E                           LD A,(HL)
441. 0319: C3 1E 03                     JP FIN
442. 031C: E6 0F                ET1     AND 0Fh
443. 031E: E1                   FIN     POP HL
444. 031F: 77                           LD (HL),A
445. 0320: C9                           RET
446. 0321:                      ;
447. 0321:                      ; SUBRUTINA PARA AGRUPAR 4 CARACTERES (Direccion
448. 0321:                      ; Inicial o Numero de Bytes), EN 2 BYTES
449. 0321:                      ;
450. 0321: 7E                   AGRUPA  LD A,(HL)
451. 0322: 07                           RLCA
452. 0323: 07                           RLCA
453. 0324: 07                           RLCA
454. 0325: 07                           RLCA
455. 0326: 23                           INC HL
456. 0327: ED 67                        RRD
```

```
457. 0329: 2B                   DEC HL
458. 032A: 77                   LD (HL),A
459. 032B: 23                   INC HL
460. 032C: 23                   INC HL
461. 032D: 7E                   LD A,(HL)
462. 032E: 07                   RLCA
463. 032F: 07                   RLCA
464. 0330: 07                   RLCA
465. 0331: 07                   RLCA
466. 0332: 23                   INC HL
467. 0333: ED 67                RRD
468. 0335: 2B                   DEC HL
469. 0336: 2B                   DEC HL
470. 0337: 77                   LD (HL),A
471. 0338: C9                   RET
472. 0339:                  ;
473. 0339:                  ;  SUBRUTINA PARA RECIBIR Y CONVERTIR
474. 0339:                  ; CARACTERES ASCII A CODIGO HEXADECIMAL
475. 0339:                  ;
476. 0339: CD EA 02         DIGITO      CALL RX
477. 033C: 3A 00 10                     LD A,(PDAT)
478. 033F: 57                           LD D,A
479. 0340: CD F2 02                     CALL TX
480. 0343: 7A                           LD A,D
481. 0344: 32 00 10                     LD (PDAT),A
482. 0347: 0E 41                        LD C,41h
483. 0349: B9                           CP C
484. 034A: DA 54 03                     JP C,ET2
485. 034D: 26 06                        LD H,06h
486. 034F: 6F                           LD L,A
487. 0350: 7E                           LD A,(HL)
488. 0351: C3 56 03                     JP TER
489. 0354: E6 0F            ET2         AND 0Fh
490. 0356: C9               TER         RET
491. 0357:                  ;
492. 0357:                  ; SUBRUTINA PARA CONVERTIR CARACTERES
493. 0357:                  ;    HEXADECIMALES A CODIGO ASCII
494. 0357:                  ;
495. 0357: C6 30            HEXASC      ADD A,30h
496. 0359: FE 3A                        CP 3Ah
497. 035B: DA 60 03                     JP C,SALIDA
498. 035E: C6 07                        ADD A,07h
499. 0360: C9               SALIDA      RET
500. 0361:                  ;
501. 0361:                  ; SUBRUTINA PARA TRANSMITIR UN CARACTER
502. 0361:                  ;           Y RECIBIR ECO
503. 0361:                  ;
504. 0361: 32 00 10         TRECO       LD (PDAT),A
505. 0364: CD EA 02                     CALL RX
506. 0367: 3A 00 10                     LD A,(PDAT)
507. 036A: C9                           RET
508. 036B:                  ;
509. 036B:                  ; SUBRUTINA PARA AVISAR A LA COMPUTADORA
510. 036B:                  ;      CUANDO SE TERMINA DE GRABAR
511. 036B:                  ;
512. 0370:                          .ORG $0370
513. 0370:                  ;
```

```
514. 0370: CD F2 02     AVISO     CALL TX
515. 0373: 3E 56                  LD A,56h
516. 0375: 32 00 10               LD (PDAT),A
517. 0378: C3 F7 00               JP RESTAU
518. 037B:              ;
519. 037B:              ; TABLAS DE PATRONES DE SENALES PARA
520. 037B:              ; LAS MEMORIAS, Y TABLAS DE CARACTERES
521. 037B:              ;
522. 0501:                        .ORG $0501
523. 0501: 54 68                  .BYTE 54h,68h    :Tabla de patrones
524. 0503:              ;                          ;por hardware para
525. 0504:                        .ORG $0504       ;escritura
526. 0504: 25                     .BYTE 25h        ;de las
527. 0505:              ;                          ;memorias
528. 0508:                        .ORG $0508       ;2716,
529. 0508: A5                     .BYTE 0A5h       ;2732A,
530. 0509:              ;                          ;27c64,
531. 0510:                        .ORG $0510       ;27128, y
532. 0510: A6                     .BYTE 0A6h       ;27256.
533. 0511:              ;
534. 0511:              ;
535. 0530:                        .ORG $0530
536. 0530: 54 68 25               .BYTE 54h,68h,25h      ;Tabla de patrones
537. 0533:              ;                                ;por software para
538. 0534:                        .ORG $0534             ;escritura de
539. 0534: A5                     .BYTE 0A5h             ;las memorias
540. 0535:              ;                                ;2716,2732A,
541. 0538:                        .ORG $0538             ;27c64,27128,
542. 0538: A6                     .BYTE 0A6h             ;y 27256.
543. 0539:              ;
544. 0539:              ;
545. 0601:                        .ORG $0601
546. 0601: 08 10                  .BYTE 08h,10h    :Tabla para
547. 0603:              ;                          ;definir la
548. 0604:                        .ORG $0604       ;capacidad
549. 0604: 20                     .BYTE 20h        ;de memoria
550. 0605:              ;                          ;de cada
551. 0608:                        .ORG $0608       ;tipo
552. 0608: 40                     .BYTE 40h        ;seleccionado.
553. 0609:              ;
554. 0610:                        .ORG $0610
555. 0610: 40                     .BYTE 40h
556. 0611:              ;
557. 0611:              ;
558. 0630:                        .ORG $0630
559. 0630: 00 01 02 03            .BYTE 00h,01h,02h,03h  ;Tabla de
560. 0634:              ;                                ;caracteres
561. 0634:                        .ORG $0634             ;numericos.
562. 0634: 04 05 06 07            .BYTE 04h,05h,06h,07h
563. 0638:              ;
564. 0638:                        .ORG $0638
565. 0638: 08 09                  .BYTE 08h,09h
566. 063A:              ;
567. 063A:              ;
568. 0641:                        .ORG $0641
569. 0641: 0A 0B 0C 0D            .BYTE 0Ah,0Bh,0Ch,0Dh ;Tabla de
570. 0645:              ;                               ;caracteres
```

```
571. 0645:                          .ORG $0645              ;alfabeticos
572. 0645: 0E 0F                    .BYTE 0Eh,0Fh           ;hexadecimales.
573. 0647:              ;
574. 0647:              ;
575. 0730:                          .ORG $0730
576. 0730: 55 65 25                 .BYTE 55h,65h,25h       ;Tabla de patrones
577. 0733:              ;                                   ;de lectura
578. 0734:                          .ORG $0734              ;para las memorias
579. 0734: A5                       .BYTE 0A5h              ;2716, 27128,
580. 0735:              ;                                   ;27c64, 27128,
581. 0738:                          .ORG $0738              ;y 27256.
582. 0738: A5                       .BYTE 0A5h
583. 0739:              ;
584. 0739:              ;
585. 0000:                          .END
```

A P E N D I C E D

CODIFICACION DEL PROGRAMA MONITOR DEL EMULADOR

```
RTCS/UDI V5.0 - PL/M-86 V2.3 COMPILATION OF MODULE EM86_88
OBJECT MODULE PLACED IN EM86TESA.OBJ
COMPILER INVOKED BY:  PLM86 EM86TESA.PSC LARGE OPTIMIZE(3) CODE SYMBOLS


                        /*****************************************/
                        /*   PROGRAMA MONITOR DEL EMULADOR PARA  */
                        /*   LOS MICROPROCESADORES 8086 Y 8088   */
                        /*****************************************/

  1                 EM86_88: DO;

                        /*  Declaracion de variables y constantes  */

  2     1                 DECLARE DC LITERALLY 'DECLARE';
  3     1                 DC INTER(128) WORD;
  4     1                 DC REG(13) WORD AT (240H);
  5     1                 DC INI WORD AT (25EH);
  6     1                 DC I INTEGER;
  7     1                 DC DIR DWORD;
  8     1                 DC (DATO$PTR,DIRIN,DIRFI,NB,FIN,SPACE) WORD;
  9     1                 DC (CHAR,COM,DIRINL,DIRINH,DIRFIL,DIRFIH,NBL,NBH) BYTE;
 10     1                 DC (BINFO,CHAR1,DIRB,DIRM,DIRA,ABD) BYTE;
 11     1                 DC DATO BASED DATO$PTR BYTE;
 12     1                 DC RTRE LITERALLY '0F0H',
                             IERE LITERALLY '0F1H',
                             LCRE LITERALLY '0F3H',
                             LSRE LITERALLY '0F5H',
                             DIVL LITERALLY '0F0H',
                             DIVH LITERALLY '0F1H',
                             BIT_1 LITERALLY '0000$0001B',
                             BIT_5 LITERALLY '0010$0000B';


                        /*****************************************/
                        /*     Procedimiento para Probar         */
                        /*        el Bit de Recepcion            */
                        /*****************************************/

 13     1                 RX: PROCEDURE;
 14     2                   DO WHILE((INPUT(LSRE) AND BIT_1) = 0);
 15     3                   END;
 16     2                 END RX;


                        /*****************************************/
                        /*     Procedimiento para Probar         */
                        /*        el Bit de Transmision .        */
                        /*****************************************/

 17     1                   TX: PROCEDURE;
 18     2                   DO WHILE((INPUT(LSRE) AND BIT_5) = 0);
 19     3                   END;
 20     2                 END TX;
```

```
                /****************************************/
                /*    Procedimiento para Producir       */
                /*            un Retardo                */
                /****************************************/

 21   1         RETARDO:PROCEDURE;
 22   2           DC (CONT,CONT1) WORD;
 23   2           CONT = 0;
 24   2           CONT1 = 0;
 25   2           DO WHILE CONT < OFFFFH;
 26   3             DO WHILE CONT1 < OFFFFH;
 27   4               CONT1 = CONT1 + 2;
 28   4             END;
 29   3             CONT = CONT + 1;
 30   3           END;
 31   2         END RETARDO;


                /****************************************/
                /*  Procedimiento para Recibir Cinco    */
                /*      Digitos de la Computadora       */
                /****************************************/

 32   1         RCDIG: PROCEDURE;
 33   2           CALL RX;
 34   2           DIRB = INPUT(RTRE);
 35   2           CALL TX;
 36   2           OUTPUT(RTRE) = DIRB;
 37   2           CALL RX;
 38   2           DIRM = INPUT(RTRE);
 39   2           CALL TX;
 40   2           OUTPUT(RTRE) = DIRM;
 41   2           CALL RX;
 42   2           DIRA = INPUT(RTRE);
 43   2           CALL TX;
 44   2           OUTPUT(RTRE) = DIRA;
 45   2           DIRA = DIRA AND 0FH;
 46   2           DIR = ((DIRA*256)*256) + (DIRM*256) + DIRB;
 47   2         END RCDIG;


                /****************************************/
                /*    Procedimiento para Transmitir     */
                /*    el Contenido de los Registros     */
                /*          a la Computadora            */
                /****************************************/

 48   1         CONTEX: PROCEDURE;
 49   2           CHAR = 0;
 50   2           DO WHILE CHAR < 14;
 51   3             CALL TX;
 52   3             OUTPUT(RTRE) = LOW(REG(CHAR));
 53   3             CALL RX;
 54   3             CHAR1 = INPUT(RTRE);
```

```
55   3              CALL TX;
56   3              OUTPUT(RTRE) = HIGH(REG(CHAR));
57   3              CALL RX;
58   3              CHAR1 = INPUT(RTRE);
59   3              CHAR = CHAR + 1;
60   3            END;
61   2          END CONTEX;


                   /*****************************************/
                   /*  Procedimiento para Modificar el      */
                   /*     Contenido de los Registros        */
                   /*****************************************/

62   1          MREG: PROCEDURE;
63   2            CHAR = 0;
64   2            DO WHILE CHAR < 14;
65   3              CALL RX;
66   3              NBL = INPUT(RTRE);
67   3              CALL TX;
68   3              OUTPUT(RTRE) = NBL;
69   3              CALL RX;
70   3              NBH = INPUT(RTRE);
71   3              CALL TX;
72   3              OUTPUT(RTRE) = NBH;
73   3              REG(CHAR) = (NBH*256) + NBL;
74   3              CHAR = CHAR + 1;
75   3            END;
76   2          END MREG;


                   /*****************************************/
                   /*     Procedimiento para Transmitir     */
                   /*     y/o Modificar el Contenido de     */
                   /*              los Registros            */
                   /*****************************************/

77   1          REGIS: PROCEDURE;
78   2            IF COM <> 01H THEN      /*com != bloque ?
79   2            DO;
80   3              CALL CONTEX;
81   3              IF COM = 0BH THEN     /*com = trace ? *
82   3                DO;
83   4                CALL RX;
84   4              ABO = INPUT(RTRE);
85   4              CALL TX;
86   4              OUTPUT(RTRE) = ABO;
87   4              IF ABO = 1BH THEN
88   4              CALL MREG;
89   4                END;
90   3                  ELSE
                        CALL RETARDO;
91   3                END;
92   2          END REGIS;
```

```
                /*************************************
                /*    Procedimiento para Cargar un   */
                /*   Archivo de Disco a Memoria RAM  */
                /*************************************/

 93   1         CARGA: PROCEDURE;
 94   2           CALL RX;
 95   2           DIRINL = INPUT(RTRE);
 96   2           CALL TX;
 97   2           OUTPUT(RTRE) = DIRINL;
 98   2           CALL RX;
 99   2           DIRINH = INPUT(RTRE);
100   2           CALL TX;
101   2           OUTPUT(RTRE) = DIRINH;
102   2           DIRIN = (DIRINH * 256) + DIRINL;
103   2           CALL RX;
104   2           NBL = INPUT(RTRE);
105   2           CALL TX;
106   2           OUTPUT(RTRE) = NBL;
107   2           CALL RX;
108   2           NBH = INPUT(RTRE);
109   2           CALL TX;
110   2           OUTPUT(RTRE) = NBH;
111   2           NB = (NBH * 256) + NBL;
112   2           OUTPUT(0FEH) = 00H;
113   2           DO WHILE (NB > 0);
114   3           CALL RX;
115   3           BINFO = INPUT(RTRE);
116   3           CALL TX;
117   3           OUTPUT(RTRE) = BINFO;
118   3           DATO$PTR = DIRIN;
119   3           DATO = BINFO;
120   3           DIRIN = DIRIN + 1;
121   3           NB = NB - 1;
122   3           END;
123   2           OUTPUT(0FEH) = 00H;
124   2         END CARGA;


                /*************************************
                /*  Procedimiento para Proporcionar  */
                /* el Reloj del Emulador al Prototipo */
                /*************************************

125   1         RELEM: PROCEDURE;
126   2           OUTPUT(0FDH) = 00H;
127   2         END RELEM;


                /*************************************
                /*    Procedimiento para Transmitir  */
                /*    y/o Modificar el Contenido de  */
                /*       Localidades de Memoria      */
                /*************************************/
```

```
128   1          MEMORI: PROCEDURE;
129   2            CALL RX;
130   2            DIRINL = INPUT(RTRE);
131   2            CALL TX;
132   2            OUTPUT(RTRE) = DIRINL;
133   2            CALL RX;
134   2            DIRINH = INPUT(RTRE);
135   2            CALL TX;
136   2            OUTPUT(RTRE) = DIRINH;
137   2            DIRIN = (DIRINH * 256) + DIRINL;
138   2            OUTPUT(0FEH) = 00H;
139   2            DATO$PTR = DIRIN;
140   2            CALL TX;
141   2            OUTPUT(RTRE) = DATO;
142   2            CALL RX;
143   2            CHAR = INPUT(RTRE);
144   2            CALL TX;
145   2            OUTPUT(RTRE) = CHAR;
146   2            DO WHILE CHAR <> 2AH;
147   3              IF CHAR <> 0DH THEN
148   3                DO;
149   4                  CALL RX;
150   4                  DATO = INPUT(RTRE);
151   4                  CALL TX;
152   4                  OUTPUT(RTRE) = DATO;
153   4                END;
154   3              DATO$PTR = DATO$PTR + 1;
155   3              CALL TX;
156   3              OUTPUT(RTRE) = DATO;
157   3              CALL RX;
158   3              CHAR = INPUT(RTRE);
159   3              CALL TX;
160   3              OUTPUT(RTRE) = CHAR;
161   3            END;
162   2            OUTPUT(0FEH) = 00H;
163   2          END MEMORI;


                /*****************************************/
                /*  Procedimiento para los 4 Modos de    */
                /*    Ejecucion del Programa a Emular    */
                /*****************************************/

164   1          EJESIM: PROCEDURE;
165   2            DO CODT(*) BYTE DATA
                       ( 089H,02EH,05CH,002H,     /*mov [25c],bp          */
                         055H,                    /*push bp               */
                         089H,0ESH,               /*mov bp,sp             */
                         08BH,05EH,004H,          /*mov bx,[bp+4]         */
                         089H,027H,               /*mov [bx],sp           */
                         083H,007H,002H,          /*add word ptr[bx],02   */
                         05DH,                    /*pop bp                */
                         08CH,016H,030H,002H,     /*mov [230],ss          */
                         089H,026H,064H,002H,     /*mov [264],sp          */
```

```
                0A1H,040H,002H,          /*mov ax,[240]                  */
                08BH,01EH,042H,002H,     /*mov bx,[242]                  */
                08BH,00EH,044H,002H,     /*mov cx,[244]                  */
                08BH,016H,046H,002H,     /*mov dx,[246]                  */
                08BH,026H,048H,002H,     /*mov sp,[248]                  */
                08BH,02EH,04AH,002H,     /*mov bp,[24a]                  */
                08BH,036H,04CH,002H,     /*mov si,[24c]                  */
                08BH,03EH,04EH,002H,     /*mov di,[24e]                  */
                08EH,016H,054H,002H,     /*mov ss,[254]                  */
                08EH,006H,056H,002H,     /*mov es,[256]                  */
                81H,0EH,58H,02H,00H,01H,/*or word ptr[258],0100h*/
                0FFH,036H,052H,002H,     /*push [252]                    */
                0FFH,036H,050H,002H,     /*push [250]                    */
                0FFH,036H,05AH,002H,     /*push [25a]                    */
                08EH,01EH,052H,002H,     /*mov ds,[252]                  */
                0CFH,                    /*iret                          */
                0FFH,0FFH,0FFH,
                0FFH,0FFH,0FFH ),
              APCODT WORD DATA(.CODT);
166   2       DC INCO(*) BYTE DATA
                ( 0FAH,                  /*cli                           */
                  0B8H,000H,000H,        /*mov ax,00                     */
                  08EH,0D8H,             /*mov ds,ax                     */
                  08EH,0C0H,             /*mov es,ax                     */
                  08BH,02EH,05CH,002H,   /*mov bp,[25c]                  */
                  0C2H,002H,000H,        /*ret2                          */
                  0FFH,0FFH,0FFH,
                  0FFH,0FFH,0FFH ),
              APINCO WORD DATA(.INCO);
167   2       DC CODT1(*) BYTE DATA
                ( 089H,02EH,05CH,002H,   /*mov [25c],bp                  */
                  055H,                  /*push bp                       */
                  089H,0E5H,             /*mov bp,sp                     */
                  08BH,05EH,004H,        /*mov bx,[bp+4]                 */
                  089H,027H,             /*mov [bx],sp                   */
                  083H,007H,002H,        /*add word ptr[bx],02           */
                  05DH,                  /*pop bp                        */
                  08CH,016H,030H,002H,   /*mov [230],ss                  */
                  089H,026H,064H,002H,   /*mov [264],sp                  */
                  0A1H,040H,002H,        /*mov ax,[240]                  */
                  08BH,01EH,042H,002H,   /*mov bx,[242]                  */
                  08BH,00EH,044H,002H,   /*mov cx,[244]                  */
                  08BH,016H,046H,002H,   /*mov dx,[246]                  */
                  08BH,026H,048H,002H,   /*mov sp,[248]                  */
                  08BH,02EH,04AH,002H,   /*mov bp,[24a]                  */
                  08BH,036H,04CH,002H,   /*mov si,[24c]                  */
                  08BH,03EH,04EH,002H,   /*mov di,[24e]                  */
                  08EH,016H,054H,002H,   /*mov ss,[254]                  */
                  08EH,006H,056H,002H,   /*mov es,[256]                  */
                  0FFH,036H,058H,002H,   /*push [258]                    */
                  0FFH,036H,050H,002H,   /*push [250]                    */
                  0FFH,036H,05AH,002H,   /*push [25a]                    */
                  08EH,01EH,052H,002H,   /*mov ds,[252]                  */
                  0CFH,                  /*iret                          */
```

```
                    OFFH,OFFH,OFFH,
                    OFFH,OFFH,OFFH ),
                  APCODT1 WORD DATA(.CODT1);
 168   2          OUTPUT(OFEH) = 00H;
 169   2          CALL MREG;
 170   2          IF COM = 04H THEN  CALL APCODT1(.SPAUX);  /*   T. rea
*/
 172   2            ELSE
                    CALL APCODT(.SPAUX);
 173   2          CALL APINCO;
 174   2          OUTPUT(OFEH) = 00H;
 175   2        END EJESIM;


                /****************************************/
                /*  Procedimiento para Proporcionar     */
                /* el Reloj del Prototipo al Emulador   */
                /****************************************/

 176   1        RELUS: PROCEDURE;
 177   2          OUTPUT(OFCH) = 00H;
 178   2        END RELUS;


                /****************************************/
                /*  Procedimiento para Salvaguardar     */
                /*       un Programa en Disco           */
                /****************************************/

 179   1        SALVA: PROCEDURE;
 180   2          CALL RX;
 181   2          DIRINL = INPUT(RTRE);
 182   2          CALL TX;
 183   2          OUTPUT(RTRE) = DIRINL;
 184   2          CALL RX;
 185   2          DIRINH = INPUT(RTRE);
 186   2          CALL TX;
 187   2          OUTPUT(RTRE) = DIRINH;
 188   2          DIRIN = (DIRINH * 256) + DIRINL;
 189   2          CALL RX;
 190   2          DIRFIL = INPUT(RTRE);
 191   2          CALL TX;
 192   2          OUTPUT(RTRE) = DIRFIL;
 193   2          CALL RX;
 194   2          DIRFIH = INPUT(RTRE);
 195   2          CALL TX;
 196   2          OUTPUT(RTRE) = DIRFIH;
 197   2          DIRFI =(DIRFIH * 256) + DIRFIL;
 198   2          OUTPUT(OFEH) = 00H;
 199   2          DO WHILE (DIRIN <= DIRFI);
 200   3            DATO$PTR = DIRIN;
 201   3            CALL TX;
 202   3            OUTPUT(RTRE) = DATO;
 203   3            DIRIN = DIRIN + 1;
 204   3          END;
```

```
205   2             OUTPUT(0FEH) = 00H;
206   2           END SALVA;


                  /*************************************/
                  /*  Procedimiento para Transmitir y/o */
                  /*  Modificar el Contenido de Puertos */
                  /*************************************/

207   1           PUERTO: PROCEDURE;
208   2             CALL TX;
209   2             CHAR = INPUT(RTRE);
210   2             CALL TX;
211   2             OUTPUT(RTRE) = CHAR;
212   2             IF CHAR =01H THEN CALL MEMORI;
214   2               ELSE
                      DO;
215   3                   CALL RX;
216   3                   DIRINL = INPUT(RTRE);
217   3                   CALL TX;
218   3                   OUTPUT(RTRE) = DIRINL;
219   3                   CALL RX;
220   3                   DIRINH = INPUT(RTRE);
221   3                   CALL TX;
222   3                   OUTPUT(RTRE) = DIRINH;
223   3                   DIRIN = (DIRINH * 256) + DIRINL;
224   3                   OUTPUT(0FEH) = 00H;
225   3                   CHAR = INPUT(DIRIN);
226   3                   CALL TX;
227   3                   OUTPUT(RTRE) = CHAR;
228   3                   CALL RX;
229   3                   CHAR = INPUT(RTRE);
230   3                   CALL TX;
231   3                   OUTPUT(RTRE) = CHAR;
232   3                   DO WHILE CHAR <> 2AH;
233   4                     IF CHAR = 01H THEN DIRIN = DIRIN + 1;
235   4                   ELSE
                            DO;
236   5                           CALL RX;
237   5                           CHAR = INPUT(RTRE);
238   5                           CALL TX;
239   5                           OUTPUT(RTRE) = CHAR;
240   5                           OUTPUT(DIRIN) = CHAR;
241   5                           DIRIN = DIRIN + 1;
242   5                         END;
243   4                         CHAR = INPUT(DIRIN);
244   4                         CALL TX;
245   4                         OUTPUT(RTRE) = CHAR;
246   4                         CALL RX;
247   4                         CHAR = INPUT(RTRE);
248   4                         CALL TX;
249   4                         OUTPUT(RTRE) = CHAR;
250   4                       END;
251   3                     OUTPUT(0FEH) = 00H;
```

```
252   3           END;
253   2         END PUERTO;


                /*****************************************/
                /*   Procedimiento para Inicializar      */
                /*      el Sistema por Hardware          */
                /*****************************************/

254   1         RESHA: PROCEDURE;
255   2           OUTPUT(0FBH) = 00H;
256   2         END RESHA;


                /*****************************************/
                /*       RUTINA DE INTERRUPCION          */
                /*****************************************/
257   1         RINTRACE: PROCEDURE;
258   2           DC CODRIT(*) BYTE DATA
                    ( 01EH,                       /*push ds                  */
                      050H,                       /*push ax                  */
                      006H,                       /*push es                  */
                      0B8H,000H,000H,             /*mov ax,00                */
                      08EH,0D8H,                  /*mov ds,ax                */
                      08EH,0C0H,                  /*mov es,ax                */
                      08FH,006H,056H,002H,        /*pop [256]      ;es       */
                      08FH,006H,040H,002H,        /*pop [240]      ;ax       */
                      08FH,006H,052H,002H,        /*pop [252]      ;ds       */
                      08FH,006H,066H,002H,        /*pop [266]      ;IPregis*/
                      08FH,006H,04AH,002H,        /*pop [24a]      ;bp       */
                      08FH,006H,05AH,002H,        /*pop [25a]      ;ip       */
                      08FH,006H,050H,002H,        /*pop [250]      ;cs       */
                      08FH,006H,058H,002H,        /*pop [258]      ;f        */
                      089H,026H,048H,002H,        /*mov [248],sp   ;sp       */
                      08CH,016H,054H,002H,        /*mov [254],ss   ;ss       */
                      089H,01EH,042H,002H,        /*mov [242],bx   ;bx       */
                      089H,00EH,044H,002H,        /*mov [244],cx   ;cx       */
                      089H,016H,046H,002H,        /*mov [246],dx   ;dx       */
                      089H,036H,04CH,002H,        /*mov [24c],si   ;si       */
                      089H,03EH,04EH,002H,        /*mov [24e],di   ;di       */
                      0B9H,008H,000H,             /*mov cx,8                 */
                      050H,                       /*alfa:push ax             */
                      0E2H,0FDH,                  /*loop alfa                */
                      0B9H,008H,000H,             /*mov cx,8                 */
                      058H,                       /*beta:pop ax              */
                      0E2H,0FDH,                  /*loop beta                */
                      08EH,016H,030H,002H,        /*mov ss,[230]   ;sist     */
                      08BH,026H,064H,002H,        /*mov sp,[264]   ;sist     */
                      0FFH,036H,066H,002H,        /*push [266]               */
                      0C3H,                       /*ret                      */
                      0FFH,0FFH,0FFH,
                      0FFH,0FFH,0FFH          ),
                    APCODRIT WORD DATA(.CODRIT);
259   2           DC CODSRIT(*) BYTE DATA
```

```
                    ( 0A1H,040H,002H,          /*mov ax,[240]          */
                      08BH,01EH,042H,002H,     /*mov bx,[242]          */
                      08BH,00EH,044H,002H,     /*mov cx,[244]          */
                      08BH,016H,046H,002H,     /*mov dx,[246]          */
                      08BH,026H,048H,002H,     /*mov sp,[248]          */
                      08BH,02EH,04AH,002H,     /*mov bp,[24a]          */
                      08BH,036H,04CH,002H,     /*mov si,[24c]          */
                      08BH,03EH,04EH,002H,     /*mov di,[24e]          */
                      08EH,016H,054H,002H,     /*mov ss,[254]          */
                      08EH,00eH,056H,002H,     /*mov es,[256]          */
                      0FFH,036H,058H,002H,     /*push [258]            */
                      0FFH,036H,050H,002H,     /*push [250]            */
                      0FFH,036H,05AH,002H,     /*push [25a]            */
                      08EH,01EH,052H,002H,     /*mov ds,[252]          */
                      0CFH,                    /*iret                  */
                      0FFH,0FFH,0FFH,
                      0FFH,0FFH,0FFH           ),
                    APCODSRIT WORD DATA (.CODSRIT);
260   2           DC SAL(*) BYTE DATA
                    ( 055H,                    /*push bp               */
                      089H,0E5H,               /*mov bp,sp             */
                      08BH,05EH,004H,          /*mov bx,[bp+4]         */
                      0B8H,000H,000H,          /*mov ax,0              */
                      08EH,0D0H,               /*mov ss,ax             */
                      036H,08BH,027H,          /*mov sp,ss:[bx]        */
                      0C3H,                    /*ret                   */
                      0FFH,0FFH,0FFH,
                      0FFH,0FFH,0FFH         ),
                    APSAL WORD DATA (.SAL);
261   2           CALL APCODRIT;
262   2           CALL REGIS;
263   2           IF COM <> 0BH THEN
264   2           DO;
265   3           CALL RX;
266   3           ABO = INPUT(RTRE);
267   3           CALL TX;
268   3           OUTPUT(RTRE) = ABO;
269   3           END;
270   2               IF (ABO=2AH)OR(ABO=03H)
                    THEN
271   2             DO;
272   3             IF ((COM = 01H)AND(ABO=2AH))  THEN CALL CONTEX;
274   3             CALL APSAL(.SPAUX);
275   3             END;
276   2           ELSE
                    CALL APCODSRIT;
277   2         END RINTRACE;


                /****************************************/
                /*   Procedimiento para Inicializar     */
                /*      el Vector de Interrupcion       */
                /****************************************/
```

```
276   1         INIVEC: FROCEDURE;
279   2           DC VECTOR(*) BYTE DATA
                     ( 0BBH,004H,000H,          /*mov [illegible]                    *
                       0C7H,007H,0FFH,0FFH,     /*mo.[illegible]                  *
                       0BBH,006H,000H,          /*mov b[illegible]                  *
                       0C7H,007H,000H,00EH,     /*mo. [illegible]                   *
                       0C3H,                    /*ret                               *
                       0FFH,0FFH,0FFH,
                       0FFH,0FFH,0FFH      /*
                   AFVECTOR WORD DATA(.VECTOR);
280   2           CALL AFVECTOR;
281   2         END INIVEC;

                /*        Inicializacion del 8250          */

282   1         OUTPUT(LCRE) = 80H;
283   1         OUTPUT(DIVH) = 00H;
284   1         OUTPUT(DIVL) = 0CH;
285   1         OUTPUT(LCRE) = 07H;
286   1         OUTPUT(IERE) = 00H;


                /****************************************/
                /*     Cuerpo Principal del Programa    */
                /****************************************/

287   1         CALL INIVEC;
288   1         CHAR = 07H;
289   1         DO WHILE CHAR <> 05H;
290   2           CALL RX;
291   2           CHAR = INPUT(RTRE);
292   2         END;
293   1         CALL TX;
294   1         OUTPUT(RTRE) = 06H;
295   1         I = 0;
296   1         DO WHILE I < 14;
297   2         REG(I) = 0;
298   2         I = I + 1;
299   2         END;
300   1         REG(4) = 600H;
301   1         REG(8) = 0E00H;
302   1         DO WHILE 0FFH;
303   2           CALL RX;
304   2           CHAR = INPUT(RTRE);
305   2           CALL TX;
306   2           OUTPUT(RTRE) = CHAR;
307   2           COM = CHAR AND 0FH;
308   2           DO WHILE (00H <= COM)AND(COM <= 0FH);
309   3             DO CASE COM;
310   4               CALL CARGA;
311   4               CALL EJESIM;
312   4               CALL RELEM;
313   4               CALL MEMORI;
```

```
314    4                  CALL EJESIM;
315    4                  CALL RELOJ;
316    4                  CALL SALVA;
317    4                  ;
318    4                  ;
319    4                  CALL PUERTO;
320    4                  ;
321    4                  CALL EJESIM;
322    4                  ;
323    4                  ;
324    4                  CALL EJESIM;
325    4                  CALL RESHA;
326    4                END;
327    3                COM = OFFH;
328    3              END;
329    2            END;
330    1      END EM86_88;
```

A P E N D I C E E

MODULO DE INTERACCION CON EL USUARIO DEL EMULADOR

```
        /*****************************************************
        /*   MODULO DE INTERACCION CON EL USUARIO DEL EMULADOR   */
        /*****************************************************/


#define NULL '\0'

        /*  Protocolo de comunicacion y llamado al menu principal  */

        main()
             {
          extern unsigned v[15];
          extern char i;
             int c,cont;
             iniciDes();
             clears();
             setci();
             c=inp(0x3F8);
             sal(5);
          printf("\n\n\n                    ESPERANDO ACKNOWLEDGE DEL
EMULADOR...");
             c=ent();
            if(c=!6)
              {
           printf("\n\n\n\n\n          ERROR : EMULADOR DE SISTEMAS
DIGITALES NO RESPONDE\n\n\n");
           printf("              CORRECTAMENTE ACKNOWLEDGE FAVOR DE");
           printf("\n\n\n                        EL EMULADOR.");
           }
            else{
           for(i=0; i<14; i++)
           v[i] = 0;
           v[4] = 0x600;
           v[8] = 0x0e00;
           for(cont=0;cont<=25;cont++)
           regant[cont]=0;
              logo();
                 do{
                   c=menu(); } while(c!=NULL);
                 clears();
           }
             }


        /*  Procedimiento para desplegar el menu principal  */

      menu()
            {  char op;
               int i=0;
            pos(10,8);
        printf("IMMMMMMMMMMMMMMMMMMMMMMMMMM/");
        printf("\033[%dm",1);
        printf("MENU PRINCIPAL");
        printf("\033[%dm",0);
```

```
        printf(".MMMMMMMMMMMMMMMMMMMMM;");
        pos(12,20);printf("1.CARGAR Y/O SALVAGUARDAR UN ARCHIVO");
        pos(13,20);printf("2.LEER Y/O MODIFICAR
REGISTROS,MEMORIA,PUERTOS");
        pos(14,20);printf("3.EJECUTAR");
        pos(15,20);printf("4.CONDICIONES DE CONTROL");
        pos(16,20);printf("5.INICIALIZACION");
        pos(17,20);printf("6.REGRESO AL SISTEMA (DOS)");
        pos(20,47);op=getch();
        while(op<'1'  || op> '6'){
        putch(7);
        op=getch();    };
        switch(op){
            case  '1':
               log1();
                   arch();
               log1();
                   break;
               case '2':
               log1();
                   leemod();
               log1();
                   break;
           case '3':
               log1();
                   ejecuta();
               log1();
                   break;
           case '4':
               log1();
                   condcont();
               log1();
                   break;
           case '5':
               log1();
                   inicia();
               log1();
                   break;
              case '6':  return(NULL); }
                   return(1);
            }

        /*  Limpia la pantalla  */

        clears()
              { printf("\033[2J");
              }


        /* Procedimiento para desplegar condiciones de control */

        condcont()
       {
            int deci=0;
```

```
        char opc;
        do{
                pos(10,8);
                printf("IMMMMMMMMMMMMMMMMMMM/");
                printf("\033[%dm",1);
                printf("CONDICIONES DE CONTROL");
                printf("\033[%dm",0);
                printf(",MMMMMMMMMMMMMMMMMMM:");
                pos(12,35);printf("a. RESET");
                pos(14,35);printf("b. RELOJ EMULADOR");
                pos(16,35);printf("c. RELOJ USUARIO");
                pos(18,35);printf("d. REGRESO A MENU PRINCIFAL");
                pos(20,47);
                opc=getch();
              while(opc<'a' || opc>'d')
         {
         putch(7);
         opc=getch();
         }
         switch(opc){
         case'a': res();
         logo();
                        break;
         case'b': rice();
         logo();
                        break;
         case'c': reus();
         logo();
                  break;
         case'd': deci=1;
             }
             }while(!deci);
    }


    /*  Comando de RESET por hardware  */

    res()
          {
          extern unsigned v[15];
          extern char i;
          int c,ban;
          char opc;
          do{
          ban=1;
          sal(15);
          for(c=0;c<30000;c++);
          setci();
          c=inp(0x3F8);
          sal(5);
          c=ent();
          clears();
          if(c==6){ printf("\n\n\n\n\n\n\n                                    ***
RESET HARDWARE HECHO ***");
```

```
                    for(i=0; i<14; i++)
                    v[i] = 0;
                    v[4] = 0x600;
                    v[8] = 0x0e00;
                  putch(7);
                  printf("\n\n\n\n\n\n                    Presione
cualquier tecla para continuar");
                  getch();
                 }
                 else{
                 printf("\n\n\n\n\n\n\n\n                    *** RESET
HARDWARE NO HECHO ***");
                 putch(7);
                 printf("\n\n\n\n\n\n\n                Presione  RETURN   para
volver a realizar el comando o");
                 printf("\n                    cualquier otra tecla para
salir del comando");
                 opc=getch();
                 if(opc==13)ban=0;
                     }
                 }while(!ban);
                 }


          /*  Comando reloj del emulador  */

          rice()
                {
                int ca;
                sal(18);
                ca=ent();
                                clears();
                printf("\n\n\n\n\n\n\n\n                    *** RELOJ DEL
EMULADOR ENTREGADO ***");
                putch(7);
                printf("\n\n\n\n\n\n\n                    Presione cualquier
tecla para continuar");
                getch();
                }


          /*  Comando reloj del usuario  */

          reus()
            {
              int c,br;
              char re;
              br=0;
              clears();
              printf("\n\n\n          Cuenta su prototipo con reloj
(S/N)   ?");
             do {
               re=getch();
               if(re=='n' || re=='N'){
```

```
            printf("\033[%dm",5);
            printf("\n\n\n                    *** ERROR : EMULADOR NO PUEDE
TRABAJAR SIN RELOJ ***");
            printf("\033[%dm",0);
            putch(7);
            printf("\n\n\n\n\n                                Presione cualquier
tecla para continuar");
            getch();
              }
          else{
            if(re=='s'  || re=='S'){
            sal(21);
                c=ent();
              printf("\n\n\n\n\n                                *** EMULADOR
TRABAJANDO CON RELOJ DE USUARIO ***");
            printf("\n\n\n                               Presione cualquier
tecla para continuar");
            putch(7);
            getch();
            }
            else{
               putch(7);
               br=1;
                 }
               }
            }while(br);
         }


         /*  Comando de inicializacion  */

         inicia()
              {
            int out=0;
            char opcion;
         do{
           pos(10,8);printf("IMMMMMMMMMMMMMMMMMMMMMMM;");
           printf("\033[%dm",1);
           printf("INICIALIZACION");
           printf("\033[%dm",0);
           printf(".MMMMMMMMMMMMMMMMMMMMMM4M:");
           pos(13,28);printf('a.SALTOS DE INST. RST");
           pos(15,28);printf("b.CONFIGURAR PTO. SERIE");
           pos(17,28);printf("c.REGRESO A MENU PRINCIPAL");
           pos(20,47);
           opcion=getch();
           while(opcion< a  || opcion>'c ) {
                 putch(7);
                 opcion=getch();
                }
              switch(opcion)
              {
              case 'a' : condrst1();
                 logo();
```

```
                    break;
                 case 'b' : veloc();
                    logo();
                    break;
                 case 'c' : out=1;
                 }
                 }while(!out);
                   }


          /*  Procedimiento para salvar archivos  */

#include "stdio.h"
  char aux[8192];

    sarch()
  { char *outfile=NULL;
    unsigned first,last;
    int total;
    int byIfirst=0,byhfirst=0,byllast=0,byhlast=0;
    clears();
    printf("\n                              ==> SALVAGUARDAR ARCHIVO <==");
    putch(7);
    printf("\n\n   Nombre del archivo a crear (sin extension):  ");
    scanf("%s",outfile);
    printf("\n\n   Direccion Inicial(en hexadecimal)= ");
    scanf("%04x",&first);
    printf("\n\n   Direccion Final(en hexadecimal)= ");
    scanf("%04x",&last);
    if(last<first)
      { printf("\n\n\n\n\n\n                  *** ERROR: DIRECCION INICIAL >
DIRECCION FINAL ***");
          putch(7);
     printf("\n\n\n\n\n                       Presione cualquier tecla para
continuar");
        getch();
        return(0);
      }
     if(last>63487L){
                printf("\n\n\n\n\n\n    *** ERROR: DIRECCION FINAL FUERA
DE RANGO ***");
                putch(7);
                printf("\n\n\n\n\n       Presione cualquier tecla para
continuar");
                getch();
                return(0);
                     }
    total=(last-first)+1;
    strcat(outfile,".hex");
    bylfirst=first&0xFF;
    byhfirst=(first&0xFF00)>>8;
    byhfirst&=0xFF;
    byllast=last&0xFF;
    byhlast=(last&0xFF00)>>8;
```

```
    byhlast&=0xFF;
    printf("\n\n\n\n\n                          SALVAGUARDANDO ARCHIVO....");
    putch(7);
    info(bylfirst,byhfirst,byllast,byhlast);
    vacia(total);
    crearch(outfile,first,total);
    printf("\n\n\n\n\n\n                    Presione cualquier tecla para
continuar.");
    getch();
  }


     /*  Procedimiento para transmitir 4 bytes  */

    info(low1,high1,low2,high2)
    int low1,high1,low2,high2;
  { int v;
    sal(6);
    v=ent();
    sal(low1);
    v=ent();
    sal(high1);
    v=ent();
    sal(low2);
    v=ent();
    sal(high2);
    v=ent();
  }

    vacia(tot)
    int tot;
  { int ind;
    for(ind=0;ind<=tot-1;ind++)
    aux[ind]=ent();
  }


     /*  Procedimiento para crear archivo  */

     crearch(salida,inicio,todos)
     char *salida;
     unsigned inicio;
     int todos;
  {
    FILE *fp,*fopen();
    unsigned after,bound;
    int track=0,bytes,i,chk;
    char buffer[17];
    fp=fopen(salida,"w");
    if(fp==NULL){ printf("\n\n\n\n\n\n\n     *** ERROR: NO SE PUEDE CREAR
EL ARCHIVO ***");
            putch(7);
             printf("\n\n\n\n\n\n          Presione cualquier tecla para
continuar");
```

```
            return(0);
              }
  bound=inicio+todos;
  for(after=inicio;after!=bound;after+=bytes)
     {  if((bound-after)>=16) bytes=16;
        else bytes=bound-after;
        for(i=0;i<=bytes-1;i++)
        buffer[i]=au[track+i];
        track+=i;
        chk=checksum(buffer,bytes);
        fprintf(fp,":");
        fprintf(fp,"%02x",bytes);
        fprintf(fp,"%04x",after);
        fprintf(fp,"00");
        for(i=0;i<=bytes-1;i++)
        fprintf(fp,"%02x",buffer[i]);
        fprintf(fp,"%02x",chk);
        fprintf(fp,"\n");
     }
  fprintf(fp,":00000001FF\n");
  fclose(fp);
}


   /*  Procedimiento para cargar y/o salvaguardar archivo  */

   arch()
   {
      int out=0;
      char opa;
   do{
   pos(10,8);printf("IMMMMMMMMMMMMMMM/");
   printf("\033[%dm",1);printf("CARGAR Y/O SALVAGUARDAR ARCHIVO");
   printf("\033[%dm",0);
   printf(".MMMMMMMMMMMMMMMM;");
   pos(13,30);printf("a. CARGAR ARCHIVO");
   pos(15,30);printf("b. SALVAGUARDAR ARCHIVO");
   pos(17,30);printf("c. REGRESO A MENU PRINCIPAL");
   pos(20,47);opa=getch();
                while(opa<'a'  || opa>'c'){
                   putch(7);
                   opa=getch();          }
                switch(opa){
          case 'a' :     carch();
               logo();
                         break;
          case 'b' :     sarch();
               logo();
                         break;
          case 'c' :     out=1;      }    /* del switch */
                      }while(!out);       /* del do */
   }                                      /* de arch */
```

```
          /*  Procedimiento para cargar archivo  */

#define NULL '\0'
#define BYH         208
#define MAXSZ         5
#define DEF      53248L
#define LOWLIM     0000
#define HIGLIM   65536L
  int totbyt;
  unsigned lower;

    carch()
  {
    int i=0,c=0,flg=1,flg2=0,lap=0,ind=0,an;
    int byth=0,byt1=0;
    int totlow=0,tothigh=0;
    char memoria[32500];
    int *k;
    char filname[20];
    char indir[MAXSZ];
     do{
     an=0;
    clears();
    printf("\t \t                    ==> CARGAR ARCHIVO <==\n");
    putch(7);
    printf("\n\t  Indique direccion inicial (en hexadecimal) ");
    printf("\n\t  'RETURN' direccion %04x  H por default ",DEF);
    printf("\n\t  (Limite inferior %04x H) =",LOWLIM);
    c=inname(indir,MAXSZ);
      if(c==0){
               lower=DEF;
               byth=BYH;
               byt1=0;
              }
       else { i=stch_i(indir,k);
              if(!i) flg2=1;
              lower=*k;
              byt1=lower&0xFF;
              byth=(lower&0xFF00)>>8;
              byth&=0xFF;
             }
      if(flg2 || lower<LOWLIM || lower >= HIGLIM){
            printf("\n\n\n\n\n\n                                    ==>  Direccion
Incorrecta <==");
         printf("\n\n\n\n                          Presione cualquier tecla p
continuar");
         putch(7);
            getch();
         an=1;
          }
     }while(an);
    senddir(byt1,byth);
    printf("\n\n\t  Nombre del archivo (Con extension .HEX):");
    scanf("%s",filname);
```

```
     dump(filname,memoria);
     totlow=totbyt&0xFF;
     tothigh=(totbyt&0xFF00)>>8;
     tothigh&=0xFF;
     sendtot(totlow,tothigh);
     senddata(memoria,0,totbyt);
     printf("\n\n\t
*************************************************************";
     printf("\n                Extension total del archivo: %d bytes de
datos",totbyt);
     printf("\n                Extension total del archivo: %04x H  bytes de
datos",totbyt);
     printf("\n\n\t
*************************************************************";
     printf("\n\n\n\n                                   ***  ARCHIVO CARGADO ***");
     putch(7);
     printf("\n\n\n\n                        Presione cualquier tecla para
continuar.");
     getch();
   }

#define RET     13
     inname(addr,limit)
       char addr[];
       int limit;
   {
    int nolin=1,b=0;
    addr[b]=getch();
    if(addr[b]==RET){  printf("\n\n                                          0000 H    =>
FOR DEFAULT <=");
                putch(7);
                       nolin=0;
                       return(nolin);
                    }
     putch(addr[b]);
     b++;
     do{
        addr[b]=getch();
        putch(addr[b++]);
        if((addr[b-1]==8)||(addr[b-1]==0x7F))
           b--;
              }while(addr[b-1]!=RET&&b<limit);
     addr[b]='\0';
     return(nolin);
   }


     /* Procedimiento para desplegar el submenu correspondiente a */
     /* lectura y/o modificacion de registros, memoria y puertos  */

unsigned byme=0;
int sigdat;
extern char cont[5];
   leemod()
```

```
{ int fin=0;
    char opt;
    do{
    pos(10,8);printf("IMMMMMMMMMMMMMMMMMMMMMM/");
    printf("\033[%dm",1);
    printf("LEER Y/O MODIFICAR");
    printf("\033[%dm",0);
    printf(".MMMMMMMMMMMMMMMMMMMMMM;");
    pos(12,29);printf("a. REGISTROS");
    pos(14,29);printf("b. LOCALIDADES DE MEMORIA");
    pos(16,29);printf("c. PUERTOS DE E/S");
    pos(18,29);printf("d. REGRESO A MENU PRINCIPAL");
    pos(20,47);opt=getch();
         while(opt<'a' || opt> d') {
                   putch(7);
                   opt=getch();
                                    }
    switch(opt)
    { case 'a':     limpia();
                sal(0x0c);
                desp_ventanas();
                regs();
                logo();
                      break;
      case 'b':     locamemo(0);
                logo();
                      break;
      case 'c':     locamemo(1);
                logo();
                      break;
      case 'd':     fin=1;
    }
      }while(!fin);
  }
#define DEFME 53248
#define DEFPTO 0
locamemo(arg)
int arg;
{ int carmem,salmem=1,salpto=1,carpto;
  char vermem,pto;
if (arg==0) { do {
                  clears();
                  salmem=1;
              printf("                          ==>   LEER Y/O MODIFICAR
MEMORIA   <==");
              putch(7);
              printf("\n\n                                    *  para salir del
comando");
              printf("\n\n             OPCIONES  :\n
      1.- POR PAGINAS.\n");
              printf("                                    2.- SECUENCIAL.\n");
              printf("\t\t\t\t\t\t   ? ");
              do { carmem=1;
                   switch(vermem=getch())
```

```
                    {
                         case '1' : putch(vermem);
                                    emom();
                                    clears();
                                    salmem=0;
                                    break;
                         case '2' : putch(vermem);
                                    sec(4);
                                    clears();
                                    salmem=0;
                                    break;
                         case  *  : return(0);
                         default  : putch(7);
                                    carmem=0;
                    }
                  }while(carmem==0);
                }while(salmem==0);
              }
     else {
             do { clears();
                  salpto=1;
                  printf("                              ==>  LEER Y/O MODIFICAR
PUERTOS DE E/S <==");
                  putch(7);
                  printf("\n                                   ( '*  para salir del
comando )");
                  printf("\n\n                        Estan los puertos mapeados
en memoria (S/N) ? ");
                  do { carpto=1;
                       switch(pto=getch())
                          {
                             case  's' :
                             case 'S' : putch(pto);
                                        sec(4);
                                        clears();
                                        salpto=0;
                                        break;
                             case 'n' :
                             case 'N' : putch(pto);
                                        sec(2);
                                        clears();
                                        salpto=0;
                                        break;
                             case '*' : return(0);
                           default    : putch(7);
                                        carpto=0;
                          }
                    }while(carpto==0);
                }while(salpto==0);
       }
   }

sec(bydir)
 int bydir;
```

```
{ int opc,i,*k,bymen=0,bymay=0,eco;
  printf("\n\n                          Indique la direccion inicial (en
hexadecimal)");
  printf("\n              o 'RETURN' para la direccion inicial ");
  if(bydir==4) { printf("%04x H por default = ",DEFME);
               }
 else printf("%02x H por default = ",DEFPTO);
 opc=recdat(bydir,1);
 switch(opc)
  {
     case 0 : i=stch_i(cont,k);
              byme=*k;
              bymen=byme & 0xFF;
              if(bydir==4) { bymay=(byme & 0xFF00) >> 8;
                             bymay&=0xFF;
                             sal(3);
                             eco=ent();
                             sal(bymen);
                             eco=ent();
                             sal(bymay);
                             eco=ent();
                             sigdat=ent();
                             rdatm(1);
                           }
              else {  sal(9);
                      eco=ent();
                      sal(byme);
                      eco=ent();
                      sigdat=ent();
                      rdatm(0);
                   }
          return;
        case 1 :  if (bydir==4) {byme=DEFME;
                                 printf("\n\n
==> 0000 H por default <==\n");
                                 putch(7);
                                 bymen=byme & 0xFF;
                                 bymay=(byme & 0xFF00)>>8;
                                 bymay&=0xFF;
                                 sal(3);
                                 eco=ent();
                                 sal(bymen);
                                 eco=ent();
                                 sal(bymay);
                                 eco=ent();
                                 sigdat=ent();
                                 rdatm(1);
                                }
                  else { byme=DEFPTO;
                         printf("\n\n                                    ==>
00 H por default <==\n");
                         putch(7);
                         sal(9);
                         eco=ent();
```

```
                        sal(byme);
                        eco=ent();
                        sigdat=ent();
                        rdatm(0);
                      }
                return;
        case 3 : return;
        case 4 : i=stch_i(cont,k);
                 byme=*k;
                 bymen=byme & 0xFF;
                 if (bydir==4) { bymay=(byme & 0xFF00) >> 8;
                                 bymay&=0xFF;
                                 sal(3);
                                 eco=ent();
                                 sal(bymen);
                                 eco=ent();
                                 sal(bymay);
                                 eco=ent();
                                 sigdat=ent();
                                 rdatm(1);
                               }
                 else { sal(9);
                        eco=ent();
                        putch(8);
                        printf(" ");
                        putch(8);
                        printf("%02x H",byme);
                        sal(byme);
                        eco=ent();
                        sigdat=ent();
                        rdatm(0);
                      }
                 return;
      }
   }

#define LIMSUPM 0xFFFF
#define LIMSUPF 0xFF
rdatm(esmem)
int esmem;
{ int regre=0,carac,i=0,*k,eco,dato,salte=0;
printf("\n");
do {
 if(esmem==1) {printf("\n\n %04x H = %02x H\n %04x H =
",byme,sigdat,byme);
              }
else {printf("\n\n %02x H = %02x H\n %02x H = ",byme,sigdat,byme);
     }
   carac=recdat(2,1);
   switch(carac)
     {  case 0 : i=stch_i(cont,k);
                 dato=*k;
                 dato=dato&0xFF;
                 sal(0);
```

```
                    eco=ent();
                    sal(dato);
                    eco=ent();
                    break;
          case 1 :  sal(0x0D);
                    eco=ent();
                    break;
          case 3 :  sal(0X2A);
                    eco=ent();
                    salte=1;
                    break;
          case 4 :  i=stch_i(cont,k);
                    dato=*k;
                    sal(0);
                    eco=ent();
                    sal(dato);
                    eco=ent();
                    putch(8);
                    printf(" ");
                    putch(8);
                    printf("%02x",dato);
     }
  if(salte==0){
   sigdat=ent();
   if (esmem==1) { if (byme==LIMSUPM) byme=0;
                    else ++byme;
                 }
    else { if(byme==LIMSUPP) byme=0;
           else ++byme;
         }
      }
    }while(salte==0);
}

#define RET     13
     inname2(addres,limite)
       char addres[];
       int limite;
   {
    int nolin=1,b=0;
    addres[b]=getch();
    if(addres[b]==RET){
                        nolin=0;
                        return(nolin);
                       }
     if(addres[b]== *'){
                putch(7);
                        nolin=-1;
                        return(nolin);
                        b++;
                       }
      else {putch(addres[b]);
            b++;
           }
```

```
    do{
       aodres[b]=getch();
       putch(addres[b++]);
     if(addres[b]== *'){
                putch(7);
                nolin=-1;
                return(nolin);
                    }
       if((addres[b-1]==8)||(addres[b-1]==0x7F))
          b--;
              }while(addres[b-1]!=RET&&b<limite);
    addres[b]='\0';
    return(nolin);
  }


     /*  Procedimiento para desplegar el logotipo del emulador  */

    logo()
  { int j;
    clears();
    printf("\033[2J");
    pos(1,21);
    printf("\xDB\xDB\xDB\xDB \xDB\xDD\xDE\xDB \xDB  \xDB \xDB
\xDB\xDB\xDB\xDB \xDB\xDB\xDB\xDC \xDB\xDB\xDB\xDB ");
    printf("\xDB\xDB\xDB\xDB");
    pos(2,21);
    printf("[    [] [ [  [ [    [  I [  [ [  [ [  [");
    pos(3,21);
    printf("\xDB\xDB\xDB  \xDB\xDE\xDD\xDB \xDB  \xDB \xDB
\xDB\xDB\xDB\xDB \xDB  \xDB \xDB  \xDB \xDB\xDB\xDB\xDB");
    pos(4,21);
    printf("\xDB    \xDB\xDE\xDD\xDB \xDB  \xDB \xDB    \xDB  \xDB \xDB
\xDB \xDB  \xDB \xDB  \xDB\xDD");
    pos(5,21);
    printf("\xDB\xDB\xDB\xDB \xDB\xDE\xDD\xDB \xDB\xDB\xDB\xDB
\xDB\xDB\xDB\xDB \xDB  \xDB \xDB\xDB\xDB\xDF \xDB\xDB\xDB\xDB");
    printf(" \xDB  \xDB\xDD");
    printf("\033[%dm",1);
    pos(7,20);
    printf("D E  S I S T E M A S  D I G I T A L E S");
    pos(8,31);
    printf("8 0 8 8  -  8 0 8 6");
    printf("\033[%dm",0);
    frame11();
    id();
  }
   frame11()
  {
   int i;
   for(i=11;i<=19;i++)
    { pos(i,8);
      printf("\xBA");
      pos(i,73);
```

```
        printf("\xBA");
      }
    pos(20,8);
    printf("HMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMM/");
    printf("\033[%dm",5);
    printf("OPCION?");
    printf("\033[%dm",0);
    pos(20,48);
    printf(".MMMMMMMMMMMMMMMMMMMMMMMM<");
   }
    id()
   { pos(22,23);
     printf("\033[%dm",1);
     printf("UNIVERSIDAD AUTONOMA METROPOLITANA");
     pos(23,32);
     printf("UNIDAD AZCAPOTZALCO");
     pos(24,31);
     printf("PROYECTO  H-PRESIDIUM");
     pos(25,40);printf("1987");
     printf("\033[%dm",0);
   }
logi()
{
      int i;
      for(i=12;i<=18;i++){
      pos(i,12);
      printf("                                                                ");
      }
}


      /*  Procedimiento para el submenu correspondiente  */
      /*          a los modos de ejecucion               */

extern int bandera;

    ejecuta()
  { char op;
    int end=0;
    do{
     bandera=0;
    pos(10,8);printf("IMMMMMMMMMMMMMMMMMMMMMMMMMMMM/");
    printf("\033[%dm",1);
    printf("EJECUCION");printf("\033[%dm",0);
    printf(".MMMMMMMMMMMMMMMMMMMMMMMMMMMM;");
    pos(13,13);printf("a. EJECUCION POR ETAPA SIMPLE");
    pos(14,13);printf("b. EJECUCION POR ETAPA SIMPLE AUTOMATICA");
    pos(15,13);printf("c. EJECUCION POR BLOQUE");
    pos(16,13);printf("d. EJECUCION EN TIEMPO REAL");
    pos(17,13);printf("e. REGRESO A MENU PRINCIPAL");
    pos(20,47);
          op=getch();
          while(op<'a'||op>'e')
           {  putch(7);
```

```
            op=getch();
          }
         switch(op)
          { case 'a' :  choice1();
              logo();
                        break;
            case 'b':   choice2();
              logo();
                        break;
            case  c':   choice3();
              logo();
                        break;
            case 'd':   choice3();
              logo();
                        break;
            case  'e' :  end=1;
          }
       }while(!end);
  }

  extern unsigned lower;

    caps(team,bboi,bbai,bbof,bbaf)
     int team;
     int *bboi,*bbai,*bbof,*bbaf;
  { int c=0,i=0,*k;
    char dire[5];
    unsigned lowaux;
    printf("\n\t          Indique direccion inicial (en hexacecimal)");
    printf("\n       'RETURN' para tomar la direccion inicial por
default ( %04X H) =",lower);
    c=inname(dire,5);
    if(!c){ *bboi=lower&0xFF;
            *bbai=(lower&0xFF00)>>8;
            *bbai&=0xFF;
          }
     else { i=stch_i(dire,k);
            if(!i) { printf("\n\n\n\n         Direccion Incorrecta\n");
              putch(7);
                    printf("\n\n\n\n       Presione cualquier tecla
para continuar.');
                    getch();
                    return(-1);
                  }
            lowaux=*k;
            *bboi=lowaux&0xFF;
            *bbai=(lowaux&0xFF00)>>8;
            *bbai&=0xFF;
          }
    if(team)
      { printf("\n\n          Indique direccion final (en hexadecimal)
=");
        c=inname(dire,5);
        if(!c){ if(team==2) *bbof=-1;
```

```
                    else{ printf("\n\n\n     Direccion Final
Incorrecta\n");
                  putch(7);
                         printf("\n\n\n\n      Presione cualquier tecla
para continuar.");
                         getch();
                         return(-1);
                       }
                  }
           else { i=stch_i(dire,k);
                  if(!i) { printf("\n\n\n      Direccion Incorrecta\n");
                  putch(7);
                           printf("\n\n\n\n    Presione cualquier tecla
para continuar.");
                           getch();
                           return(-1);
                         }
                  lowaux=*k;
                  *bbof=lowaux&0xFF;
                  *bbaf=(lowaux&0xFF00)>>8;
                  *bbaf&=0xFF;
                  }
      }
  }


     /*  Procedimiento del comando de ejecucion por etapa simple  */

      choice1()
  { extern char cat;
    int bajo,alto,v,eco;
      clears();
      putch(7);
      sal(0x1B);
      v=ent();
      printf("\n       ==> EJECUCION POR ETAPA SIMPLE <==  ");
      desp_ventanas();
      regs2();
      putch(7);
      do{
        regs();
        putch(7);
        }while(cat!='*');
      sal(0x2A);
      v=ent();
  }

     /*  comando de ejecucion por etapa simple automatica  */

   choice2()
  { extern char cat;
    int bajo,alto,v,eco;
      clears();
      putch(7);
```

```
      sal(0x1B);
      v=ent();
      printf("\n                    ==> EJECUCION POR ETAPA SIMPLE
AUTOMATICA <==");
      desp_ventanas();
      regs2();
      putch(7);
      pos(23,9);
      printf("
");
      do{
        regs3();
        putch(7);
        }while(cat!='*');
      sal(0x2A);
      v = ent();
 }

    /*  procedimiento para el comando de ejecucion por bloque  */

   choice3()
 { extern char cat;
   int bajo,alto,v,eco;
     clears();
     putch(7);
     sal(0x11);
     v = ent();
     printf("\n                         ==> EJECUCION POR BLOQUE <==");
     printf("\n                           Condiciones Iniciales");
     desp_ventanas();
     regs2();
     pos(23,9);
     printf("        Direccion Final de Bloque (en Hexadecimal) ?  ");
     captel();
     pos(3,20);
     printf("                                                  ");
     pos(24,9);
     printf("             EJECUTANDO BLOQUE                    ");
     do{
       regs4();
       } while (cat !=  *');
 }

   sale(code,low,high)
    int code,low,high;
 { int cap;
   if(code<1000){ sal(code);
                  cap=ent();
                }
   sal(low);
   cap=ent();
   sal(high);
   cap=ent();
 }
```

```
#include <defn.ext>

/* Definicion de puertos para el 8250 de la PC */

#define CONTROL 0x3FB
#define DLH 0x3F9
#define DLL 0x3F8
#define STATUS 0x3FD
#define BUFFER 0x3F8


     /* ATOHTOI.- Convierte una cadena de ascii's ( de digitos   */
     /*  hexadecimales) a un numero entero, el cual regresa.     */

unsigned  atohtoi (a)
char *a;
{
unsigned  n=0;
int i=0;

while (a[i] != NULL)
{
  if ( digito(a[i]) )
     n= 16 * n + a[i++] - '0';
  else if ( min(a[i]) )
     n= 16 * n + a[i++] - 'a' + 10;
  else if ( may(a[i]) )
     n= 16 * n + a[i++] - 'A' + 10;
}
return(n);
}


     /* CONFIGURA.- Programacion por default del puerto serie   */
     /*             de la PC.:                                  */
     /*             8 bits, 2 de parada, sin paridad, 2400 baud s. */

configura ()
{
outp(CONTROL, 0x80);
outp(DLH, 0x00);
outp(DLL, 0x30);
outp(CONTROL, 0x07);
}


     /*  IN_WORD.- Toma del puerto serie 2 bytes y los ensamola  */
     /*            para formar y devolver un numero de 16 bits.  */

unsigned in_word()
{
    char baja;
    unsigned alta;
```

```
    baja = rec();
    trans(baja);
    alta = rec();
    trans(alta);
    alta = alta * 256 + (int)baja;
    return ( alta );
}


     /* LEE_WORD.- Lee del teclado una serie hasta de 4 digitos */
     /*            hexadecimales, colocandolos en el apuntador  */
     /*            que recive como parametro.                   */
     /*            Regresa 0 si no se tecleo nada o 1 en caso   */
     /*            contrario.                                   */

char lee_word (cadena)
char *cadena;
{
  char algo = 1;
  char  g, j = 0;

  while ((g = getch()) != CR )
  {
    if ( digito(g) || may(g) || min(g) )
    {
        if ( j < 4 )
        { cadena[j++] = g;
          putch(g);
        }
    }
    else if ( g == BS )
      { if ( j > 0 )
        { putch( g );
          putch ( 0x20 );
          putch ( 0x08 );
          j--;
        }
      }
  }
  if ( j == 0 )
    algo = 0;
  cadena [j] = NULL;
  return ( algo );
}


     /*  LIMPIA.- Borra la pantalla de la terminal.

limpia()
{
 printf("\033[2J");
}
```

```
      /*  OUT_WORD.- Transmite por el puerto serie un numero de   */
      /*             16 bits, en forma de 2 bytes  consecutivos   */
      /*             primero la parte baja y luego la alta.       */

out_word(num)
unsigned    num;
{
    char baja,
      alta;
    baja = num & 0xff;
    trans( baja );              /* Parte baja */
    baja = rec();
    alta = (num >> 8) & 0xff;
    trans( alta );              /* Parte alta */
    alta = rec();
}


      /*  POSC.- Posiciona el cursor de la pantalla en la linea   */
      /*         y columna deseadas.                              */

posc(a,b)
 int a,b;
 {
  printf("\033[%d;%dH",a,b);
 }


      /*  PUTF_HEXA.- Escribe en pantalla un decimal en  forma    */
      /*              de digitos hexadecimales, ajustandolo al    */
      /*              campo deseado.                              */
      /*  Si el campo es mayor que el espacio  necesario  para    */
      /*  imprimir el numero, se completa con ceros,  en  caso    */
      /*  contrario, se ignora el campo.                          */

putf_hexa ( num , campo )
unsigned    num;
char    campo;
{
  char  a[10];
  int   i = 0;
  while ( num > 0 )
  {
    a[ i++ ] = num % 16;
    num = num / 16;
  }
  while ( campo > i )                   /* Completa con ceros */
  {  campo--;
    putch ( '0' );
  }
  while ( i > 0 )
  { i--;
    if ( a[i] > 9 )  a[i] = a[i] +  'A' - 10 ;
    else a[i] =a[i] +  '0';
```

```
    putch (a[i]);
  }
}

     /*  REC.-Devuelve  un caracter leido del puerto serie.      */

rec()
{
    int st;
    do                              /*Polling sobre el STATUS del Rx */
    { st=inp(STATUS);
      st &=0x01;
    }while(!st);
    st= inp(BUFFER);
    return(st);
}

     /*  TRANS.-Efectua un outp por el puerto serie              */
     /*          (un solo caracter).                             */

trans(x)
char x;
{
  int st;
  do                                /*Poolling sobre el STATUS del Tx */
   { st = inp (STATUS);
     st &= 0x20;
   }while (!st);
   outp(BUFFER, x);
}

     /* Procedimiento para recibir, desplegar y/o modificar *
     /*  el contenido de los registros del microprocesador  */

#include <stdio.h>
#include <defn.ext>

unsigned  v[15];
char i,l;
char cat;
regs()
{
    unsigned        atohtoi();
    char        c,
                k,
                algo,
                app[10],
                no_cambios = 1,
                lee_word();

    lee_registros();
```

```
    desp_registros();
    separa_banderas( v[12] );
    posc( 23 , 9 );
    printf("        [M].- Modifica        [ESC].- Ejecuta
Instruccion");
    posc( 24 , 29 );
    printf("[*].- Regreso a Menu");
    posc(  6 , 16 );
    printf("/");
    putch(BS);

    /*   Interpretador   */

    i = 0;
    l = 6;
    k = 16;
    while (((c = getch()) != ESC) && (c != ABORT) && (c !=  * ))
    { posc ( l , k );
      printf(" ");
      if ( c == NULL )
      {
        c = getch();
        if ( c == BAJA)
          baja_cursor();
        else if ( c == SUBE )
          sube_cursor();
      }
      else if  ( c == 'M' || c ==  m  )
      {
         no_cambios = 0;
         posc( l , 33);
         printf("    ");
         posc( l , 33 );
        algo = lee_word(app);
         if  ( algo != 0 )
           v[i] = atohtoi (app);
         if ( i == 12 )
         {
           v[12] &= 0xFEFF;                    /* Apaga la bandera T *
           separa_banderas(v[12]);
         }
         posc ( l , 33 );
         putf_hexa ( v[i] , 4 );
         if ( i == 3 || i == 7 || i == 9 )
         {  l = l + 2;
            i = i + 2;
         }
         else if ( i == 12 )
         {  l = 6;
            i = 0;
         }
         else
         {
           l++;
```

```
            i++;
          }
      }
      posc ( l , k );
      printf("/");
      putch(BS);
    } /* while (interpretador) */
    if ( c != '*' )
    if ( c == ABORT || no_cambios == 1)/* Termina sin actualizar *
      {
      trans( CR );
      c = rec();
      }
    else        /* Termina y envia el
                                  contenido actualizado
                                  de los registros      */
    {
      trans ( ESC );
      c = rec();
      for ( i = 0 ;  i < 14 ;  i++ )
        out_word ( v[i] );
    }
      cat = c;
}


    /*    Procedimiento para desplegar y modificar el     */
    /*   contenido de los registros del microprocesador   */

regs2()
{
    unsigned        atohtoi();
    char        c,
                k,
                algo,
                app[10],
                no_cambios = 1,
                lee_word();

    desp_registros();
    separa_banderas( v[12] );
    posc( 23 , 9 );
    printf("[M].- Modifica    [ESC].- Actualiza y Ejecuta");
    posc( 24 , 29 );
    posc(  6 , 16 );
    printf("/");
    putch(BS);

    /* Interpretador */

    i = 0;
    l = 6;
    k = 16;
    while (((c = getch()) != ESC) && (c != ABORT) && (c !=  * ))
```

```
    { posc ( l , k );
      printf(" ");
      if ( c == NULL )
      {
        c = getch();
        if ( c == BAJA)
          baja_cursor();
        else if ( c == SUBE )
          sube_cursor();
      }
      else if  ( c ==  'M' || c ==  'm' )
      {
         no_cambios = 0;
         posc( l , 33);
         printf("    ");
         posc( l , 33 );
         algo = lee_word(app);
         if  ( algo != 0 )
           v[i] = atohtoi (app);
         posc ( l , 33 );
         putf_hexa ( v[i] , 4 );
         if ( i == 3 || i == 7 || i == 9 )
         {  l = l + 2;
            i = i + 2;
         }
         else if ( i == 12 )
         {  l = 6;
            i = 0;
         }
         else
         {
           l++;
           i++;
         }
      }
      posc ( l , k );
      printf("/");
      putch(BS);
   }  /* while (interpretador) */
   if ( c != '*' )
     for ( i = 0 ;  i < 14 ;  i++ )
       out_word ( v[i] );
     cat = c;
}

     /*     Procedimiento para recibir y desplegar el      *
     /*   contenido de los registros del microprocesador   *

regs3()
  {
    int con,con1;
    char c,lee_word();
    lee_registros();
    desp_registros();
```

```
    separa_banderas (v[12]);
    posc( 23 , 29);
    printf("[*].- Regreso a Menu");
    c = CR;
    if (kbhit() )
      c = getch();
    if (c != '*')
      {
      con = 0;
      do  {
     con1 = 0;
     do {
       con1 = con1 + 1;
       } while ( con1 < 400 );
        con = con + 1;
     } while ( con <1000 );
      trans(CR);
      c = rec();
      }
    cat = c;
  }


     /* Procedimiento para recibir y desplegar el contenido de */
     /*  los registros con el comando de ejecucion por bloque  */

regs4()
  {
    char c,lee_word();
    c = CR;
    if (kbhit())
      c = getch();
    if (c == '*')
      trans(0x2A);
      else
     {
     trans(CR);
     c = rec();
     if (c == * )
       {
       lee_registros();
       desp_registros();
       separa_banderas(v[12]);
       pos(24,9);
       printf("
       c = getch();
       while(c !=  *')
          { putch(7);
            putch(7);
            c = getch();
          }
       }
    }
    cat = c;
 }
```

```
          /*     Lectura de registros    */

lee_registros()
{
    char   i;

    for ( i = 0; i < 14 ; i++)
      {
      v[i] = in_word();
      }
}


     /*   Despliegue de ventanas     */

desp_ventanas()
{
    char i, 1, k;

    posc( 2 , 16 );
    printf("\033[%dm", 1 );
    printf( "REGISTROS:   CONTENIDO:");
    printf("\033[%dm", 0 );
    posc( 5 , 14 );
    printf( "IMMM;IMMMMMM;    IMMMMMMMM;");
    for ( i = 6; i < 20; i++)
    {  posc ( i , 14 );
       printf( ":   ::       :   :         :")
    }
    posc( 20 , 14 );
    printf( "HMMM<HMMMMMM<    HMMMMMMMM<");
    posc( 10 , 47 );
    printf("\033[%dm", 1 );
    printf("BANDERAS");
    printf("\033[%dm", 0 );
    posc( 11 , 42 );
    printf( "IMMMMMMMMMMMMMMMM;");
    posc( 12 , 42 );
    printf( ":                 :");
    posc( 13 , 42 );
    printf( "LMMMMMMMMMMMMMMMM9");
    posc( 14 , 42 );
    printf( ":                 :");
    posc( 15 , 42 );
    printf( "HMMMMMMMMMMMMMMMM<");

    1 = 6;
    k = 22;
    posc( 1 , k );
    printf("AX");
    posc ( 1 + 1 , k);
    printf ("BX");
    posc ( 1 + 2 , k);
    printf ("CX");
```

```
      posc ( l + 3 , k);
      printf ("DX");
      posc ( l + 4 , k);
      printf ("SP");
      posc ( l + 5 , k);
      printf ("BP");
      posc ( l + 6 , k);
      printf ("SI");
      posc ( l + 7 , k);
      printf ("DI");
      posc ( l + 8 , k);
      printf ("CS");
      posc ( l + 9 , k);
      printf ("DS");
      posc ( l + 10 , k);
      printf ("SS");
      posc ( l + 11 , k);
      printf ("ES");
      posc ( l + 12 , k);
      printf (" F");
      posc ( l + 13 , k);
      printf ("IP");
      posc( 12 , 43 );
      printf("O D I T S Z A P C");
}

      /* Despliega el contenido de los registros en la pantalla */

desp_registros()
{
     char  i, l, k = 33;

     for ( i = 0, l = 6 ; l < 20 ; l++ , i++)
     { posc( l , k );
       putf_hexa ( v[i] , 4 );
     }
}


      /*  Procedimiento para manejar el cursor hacia abajo  */

baja_cursor()
{
  if ( i < 13 )
     {  l = l + 1;
        i = i + 1;
     }
 }


      /*  Procedimiento para manejar el cursor hacia arriba  */

sube_cursor()
```

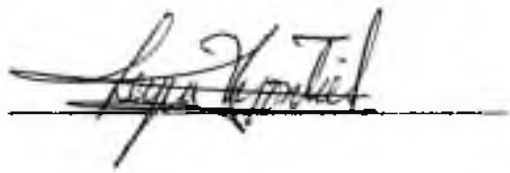
```
{
  if ( i > 0 )
    {  l = l - 1;
       i = i - 1;
    }
}


     /*   procedimiento para separar las banderas    */

separa_banderas(num)
unsigned  num;
{ char ix , cadena[18];
  for ( ix = 0 ; ix < 15 ; cadena[ix++] = 0 );
  for ( ix = 0 ; num > 0 ; ix++ )
  {
    cadena[ix] = num % 2;
    num /= 2;
  }
    posc ( 14 , 43 );
    printf ( "%d", cadena[11] );
    posc ( 14 , 45 );
    printf ( "%d", cadena[10] );
    posc ( 14 , 47 );
    printf ( "%d", cadena[9] );
    posc ( 14 , 49 );
    printf ( "0" );
    posc ( 14 , 51 );
    printf ( "%d", cadena[7] );
    posc ( 14 , 53 );
    printf ( "%d", cadena[6] );
    posc ( 14 , 55 );
    printf ( "%d", cadena[4] );
    posc ( 14 , 57 );
    printf ( "%d", cadena[2] );
    posc ( 14 , 59 );
    printf ( "%d", cadena[0] );
}


     /*  Procedimiento para leer del teclado la direccion  */
     /*                    final de bloque                 */

#include <stdio.h>
captel()
  {
  int i;
  char dirb,dirm,dira,dir[6];
    gets(dir);
    for(i=0; i<5; ++i)
      dir[i] = (dir[i] >= '0' && dir[i] <= '9') ? dir[i] - '0' :
      (dir[i] = (dir[i] >= 'A' && dir[i] <= 'F') ? dir[i] - 'A' + 10 :
      (dir[i] = (dir[i] >= 'a' && dir[i] <= 'f') ? dir[i]-
'a'+10:dir[i]));
```

```
    dirb = ((dir[3] << 4 ) & 0xf0) | (dir[4] & 0xf);
    dirm = ((dir[1] << 4 ) & 0xf0) | (dir[2] & 0xf);
    dira = dir[0] & 0xf;
    trans(dirb);
    rec(dirb);
    trans(dirm);
    rec(dirm);
    trans(dira);
    rec(dira);
    printf("%d %d %d",dirb,dirm,dira);
}
```

El jurado designado por la Seccion de Computacion del Departamento de Ingenieria Eléctrica del Centro de Investigacion y de Estudios Avanzados del Instituto Politecnico Nacional, aprobó esta tesis el dia 5 de Agosto del año de mil novecientos ochenta y ocho.

Dr. Manuel E. Guzmán Renteria

Dr. Armando Maldonado Talamentes



Dr. José Luis Leyva Montiel

AUTOR	JIMENEZ FLORES, A.
TITULO	SISTEMA PARA DIAGNOSTICO Y DESARROLLO DE APLICACIONES ...
CLASIF.	XM 88.8
NOTRO.	81 11,049

NOMBRE DEL LECTOR	FECHA PREST	FECHA DEVOL
Alberto Pineda S.	29-11-95	30/04/96
Augusto Garcia P.	10/04/96	11/04/96