UNIVERSIDAD AUTONOMA DE GUADALAJARA

INCORPORADA A LA UNIVERSIDAD NACIONAL AUTONOMA DE MEXICO

ESCUELA DE INGENIERIA EN COMPUTACION

"DISEÑO Y CONSTRUCCION DE UN GRABADOR DE EPROMS PARA LA IBM - PC"

TESIS PROFESIONAL

QUE PARA OBTENER EL TITULO DE

INGENIERO EN COMPUTACION

P R E S E N T A

SERGIO ALFONSO DIAZ DEL RIO

SEPTIEMBRE. 1990

GUADALAJARA , JAL.

UNAM – Dirección General de Bibliotecas Tesis Digitales Restricciones de uso

INTRODUCCION

Necesaria y frecuentemente se escucha hablar dentro del campo de la computación y sistemas digitales de las dos grandes clasificaciones de memoria llamadas RAM y ROM, utilizándose el amplio término de RAM (Random Access Memory) para designar a las memorias donde con frecuencia se realizan operaciones de lectura-escritura y las cuales pierden su contenido al dejar de ser energizadas.

Las memorias ROM (Read Only Memory) o memorias de solo lectura; bien llamadas así debido a que solo en una ocasión o muy esporádicamente son escritas; son el tipo de memoria que nos ocupa.

En memorias de solo lectura ROM, una vez que se ha escrito sobre ellas, no pierden su contenido al retirar el voltaje que las alimenta, sino que conservan la información para ser leída posteriormente, tantas veces como sea necesario.

Existen varios tipos de memorias ROM, con diferentes características, que se han ido desarrollando para ofrecer al usuario la versatilidad que éste requiera en sus proyectos.

Por citar algunos tipos, se tienen las memorias de solo lectura programadas por máscara, identificadas como MROM (Mask ROM), escritas desde la fábrica utilizando un negativo fotográfico o máscara, especial en cada caso, para establecer las conexiones eléctricas internas que satisfagan las especificaciones proporcionadas por el cliente. Obviamente este proceso parecerá caro y lo es, a menos que se manejen grandes cantidades de ROMs idénticas, de manera que resulte costeable.

También existen las memorias de solo lectura programables, PROM (Programmable ROM), que a diferencia de de las MROM, su contenido puede ser programado por el usuario, pero que de igual manera se programan solo una y definitiva vez. Si se requiere de un cambio en el programa o información grabada dentro de ellas, deberá utilizarse una unidad nueva.

En particular son las EPROM (Eraseable and Programmable ROM) las memorias que nos interesan, convertidas en el dispositivo idóneo para almacenar programas o datos para ser usados en proyectos basados en microprocesadores.

Las memorias EPROM permiten tanto al fabricante como al usuario programar su contenido mediante la adecuada aplicación de un nivel especial de voltaje, además de presentar la posibilidad de ser borradas al exponerlas a una fuente de luz ultravioleta durante un determinado periodo de tiempo. Dado que este tipo de memoria ROM puede ser borrada, nos da la oportunidad de hacer modificaciones al programa o información que ésta contenga, o bien borrarla completamente para ser reutilizada en un proyecto posterior.

Las memorias EPROM nunca son escritas cuando éstas están en operación dentro de un sistema digital, tienen que ser retiradas y programadas en el exterior.

Los circuitos necesarios para llevar a cabo la programación y la manera en que se manipula la información que va ser escrita dentro de la EPROM, es precisamente el objeto de esta tésis.

Debido a la popularidad que han tenido y siguen teniendo las microcomputadoras; y en especial la IBM-PC y sus innumerables "clones" que hoy en día existen; hace que sean encontradas en forma muy frecuente, por los estudiantes y profesionistas, en sus escuelas, trabajos o en su propio hogar.

Es por eso que se pensó en la construcción de un grabador de EPROMs para utilizarse en conjunción con una de estas microcomputadoras, de manera que se obtenga una forma fácil y versátil de manipular la información que se quiera almacenar en la memoria EPROM al mismo tiempo que simplifica la implementación de los circuitos que llevan a cabo la grabación en sí.

A lo largo de los capítulos de esta tésis, se pretende proporcionar de la manera más simplificada posible, las instrucciones, diagramas y programas necesarios, para la construcción de este sencillo grabador de EPROMs; además de presentar información sobre las opciones para utilizar el puerto paralelo estandar de la IBM-PC en la lectura de datos, o el tema de la residencia de programas en memoria que podrían resultar interesantes o útiles para algún proyecto propio.

ANTECEDENTES

Cuando se pensó la manera en que se construiría el grabador de EPROMs, se tuvieron que considerar varias opciones, una de las disyuntivas más importantes, fue si el grabador sería paralelo o serial, esto es, la manera en que se comunicaría con la PC.

El puerto serial de la IBM PC; a diferencia del más simple puerto paralelo; puede experimentar varios tipos de errores de transmisión, por lo que la implementación de los circuitos para lograr una transmisión confiable, son más complicados.

No obstante, el manejo del puerto serial, resulta más o menos simple, ya que se cuenta con los servicios proporcionados por el ROM BIOS (ROM Basic Input Output System) a través de la interrupcion 14h, y que facilita la programación del puerto, evitandonos programar el hardware directamente.

Este tipo de comunicación es ampliamente utilizado, debido a que es la forma más barata de conectar dos dispositivos que se encuentran separados por algunos metros de distancia.

Por otro lado, la comunicación en paralelo de algún dispositivo con la computadora (exceptuando a la propia impresora) generalmente se efectúa através de una tarjeta controladora conectada al "bus" de expansión de la computadora.

La mayoria de los grabadores comerciales de EPROMs, que se ven anunciados en revistas, resultan ser de tipo serial o con su tarjeta controladora especial para el grabador, pudiendo grabar en ellos varios tipos de EPROMs, pudiendo éstos costar entre 100 y 1000 dolares.

Es poco frecuente encontrar dispositivos que utilicen el puerto de la impresora para comunicarse con la computadora, sin embargo existen, y resulta la forma más sencilla de entablar comunicación. *

Otra de las desiciones que fue necesario tomar, consistió en encontrar la forma de implementar los

---

* *El sistema de desarrollo, emulador de los microprocesadores 8031/8051; presentado por Steve Ciarcia en sus artículos de la revista Byte, en los meses de Agosto y Septiempre del 88; y el grabador de EPROMs incorporado a ese sistema, son un ejemplo de tal dispositivo.*

circuitos necesarios para llevar a cabo la grabación en sí.

Pensar en el uso de un microprocesador como el 6809, y sus chips de soporte, resultaría demasiado complicado para el fin que se pretende; utilizar quizá un microcontrolador de la familia del Intel8051, sería una más simple y feliz idea.

Otra opción fue, la de implementar nuestro propio controlador, haciendo uso de elementos discretos de lógica digital. El utilizar un registro de desplazamiento como controlador de la secuencia de operaciones, sería una forma usual de llevarlo a cabo. *

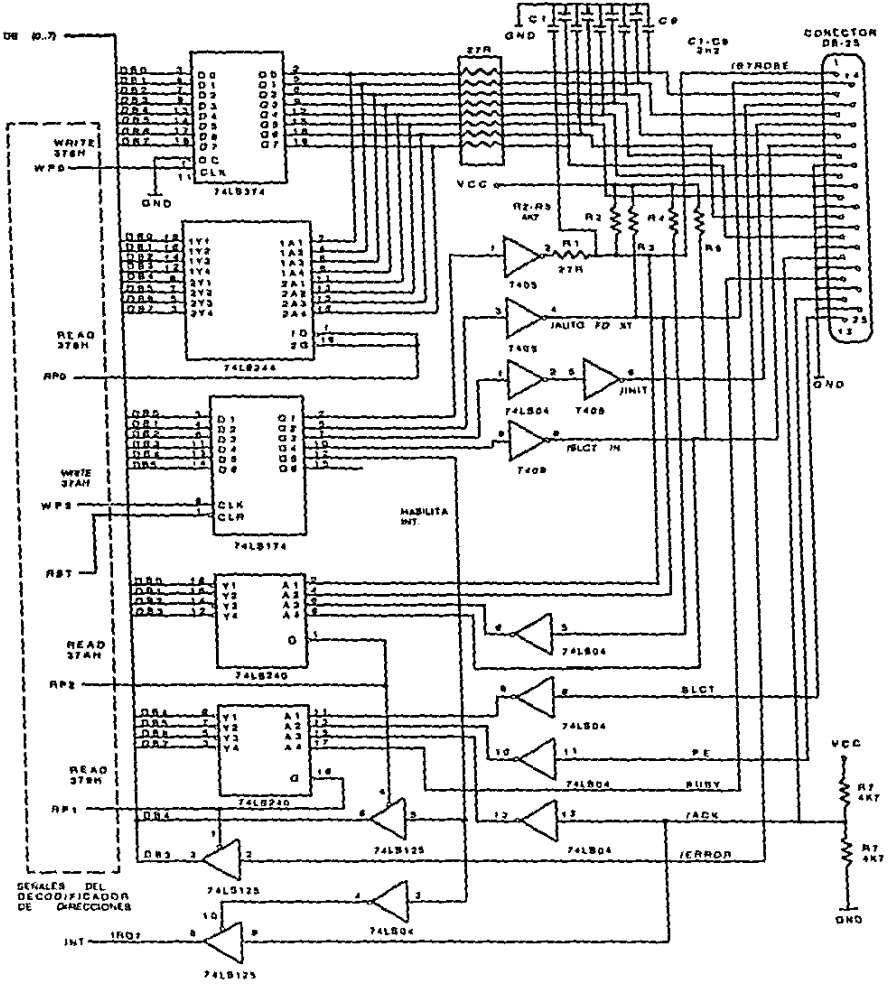
Por ser esta última opción, una alternativa no muy costosa, y de sencilla implementación, se prefirió tomar este enfoque de utilizar las herramientas básicas que proporcinoa el diseño digital, para hacer interfaz con el puerto paralelo de la IBM PC (normalmente puerto de la impresora), que por más básico que sea el sistema disponible, debe incluir necesariamente este puerto como equipo estándar.

Existe un inexplicable e irritante problema con el diseño del puerto paralelo estándar de la IBM-PC, no se puede leer información a través de sus líneas de datos, es de solo salida; sin embargo, el hardware requerido para poder hacerlo está ahí, dentro de la máquina, careciendo solo de unas cuantas conexiones para proporcionarle tal habilidad.

En la figura de la página siguiente, se muestra el diagrama típico de la interfaz con el "bus" de la PC y el conector tipo D de veinticinco pines del puerto paralelo.

Realmente el diagrama no es un asunto tan complejo como podría pensarse, ya que se usan dispositivos estándar LSTTL, que si nos detenemos por un momento para analizarlos nos resultará clara su función.

** En la revista de Radio Electronics del mes de Febrero del 82 en el artículo presentado por Robert N. Beaber, se muestra un grabador manual de EPROMs, donde se hace uso de un contador de registro de desplazamiento, como controlador de secuencia de operaciones necesarias para escribir sobre la EPROM.*



LPT(N): Interfaz del conector DB-25 con el "bus" de la IBM PC

PUERTO DE LA IMPRESORA
DE LA IBM PC

Pin	Sentido	"Centronics"
1	Salida	$\overline{\text{STROBE}}$
2-9	Salida	Bits de Datos 0-7
10	Entrada	$\overline{\text{ACK}}$
11	Entrada	BUSY
12	Entrada	PE
13	Entrada	SLCT
14	Salida	$\overline{\text{AUTO}}$ FD XT
15	Entrada	$\overline{\text{ERROR}}$
16	Salida	$\overline{\text{INIT}}$
17	Salida	$\overline{\text{SLCT}}$ IN
18-25	—	Común

Tabla 1

Direcciones de los puertos de la impresora:

	Datos	Estatus	Control
LPT1:	378	379	37A
LPT2:	278	279	27A
LPT3:	3BC	3BD	3BE

Tabla 2

Bits puerto de estatus:

Bit	Función	
D0-D2	No usados	
D3	$\overline{\text{ERROR}}$	pin 15
D4	$\overline{\text{SLCT}}$	pin 17
D5	PE	pin 12
D6	ACK	pin 10
D7	Busy	pin 11

Tabla 3

Bits puerto de control:

Bit	Función
D0	$\overline{\text{STROBE}}$
D1	AUTO FD XT
D2	$\overline{\text{INIT}}$
D3	SLCT IN
D4	Habilita interrupciones
D5-D7	No usados

Tabla 4

De los veinticinco pines del conector, en total se cuenta con doce señales de salida y cinco de entrada, el resto son de referencia (GND) para las ocho líneas de datos.

De manera que si queremos leer a través del puerto paralelo tenemos tres opciones:

1.- Modificar internamente el puerto, quitando y agregando las pocas conexiones que le faltan. *

2.-Comprar una tarjeta "clon" con un puerto paralelo bidireccional (por lo general todas son así).

3.-De alguna manera utilizar las señales de estátus, que son los únicos pines de entrada que se tienen disponibles.

Para modificar el puerto paralelo de la IBM-PC, observar la figura de la página siguiente, donde se indica la simple conexión y desconexión para convertirlo en un puerto bidireccional.

La modificación se explica fácilmente si se observa que ahora el nivel lógico del pin 1 del integrado 74LS374; que contiene ocho flip-flops D de tri-estado; puede ser puesto en alto a través de una sexta señal de control, almacenada en uno de los seis borrables flip-flop D que contiene el integrado 74LS174. De manera que las salidas del 74LS374, sean deshabilitadas y permitan la lectura de información externa a través de las líneas de datos.

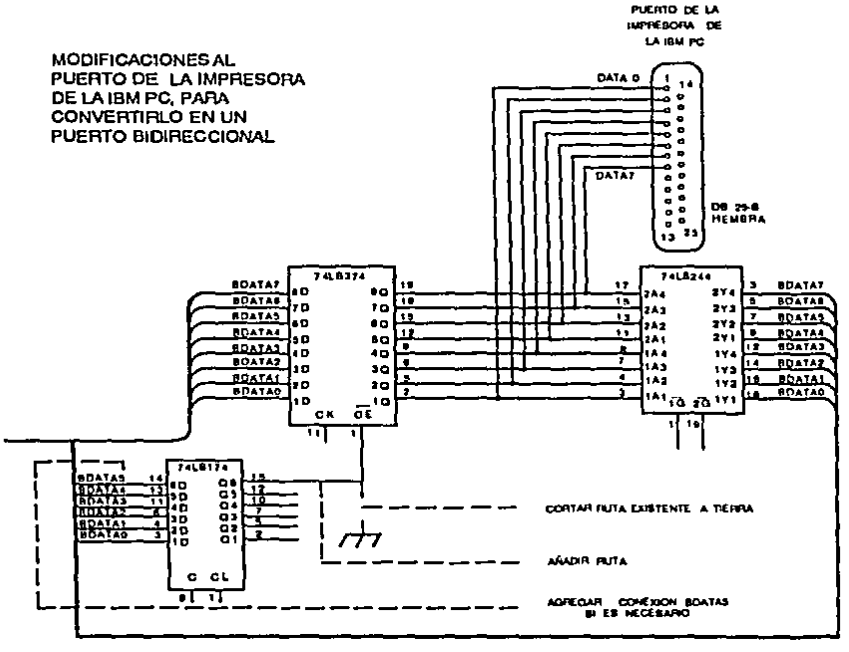
Por supuesto, hay que utilizar software para hacer uso de esta nueva habilidad del hardware, colocando un 1 en el bit D5 de el puerto de control de la impresora (bit normalmente no usado, ver tabla 4); y regresando este bit a su posición original, después de haber leído el dato externo a través del puerto de datos de la impresora (referirse a la tabla 2).

No es de culpar a nadie, el resistirse a abrir su computadora y soldarle y desoldarle algunos alambres, quizá sea la actitud correcta, por lo que el considerar mejor gastar unos dolares en un puerto paralelo que sea ya

---

** Para referencia, la primera opción es tomada por Steve Ciarcia en su artículo de la revista Byte de Septiembre del 88; de cualquier manera, aquí también se describirá.*

MODIFICACIONES AL PUERTO DE LA IMPRESORA DE LA IBM PC, PARA CONVERTIRLO EN UN PUERTO BIDIRECCIONAL



bidireccional y salvaguardar la integridad de nuestra computadora, sea una solución más segura.

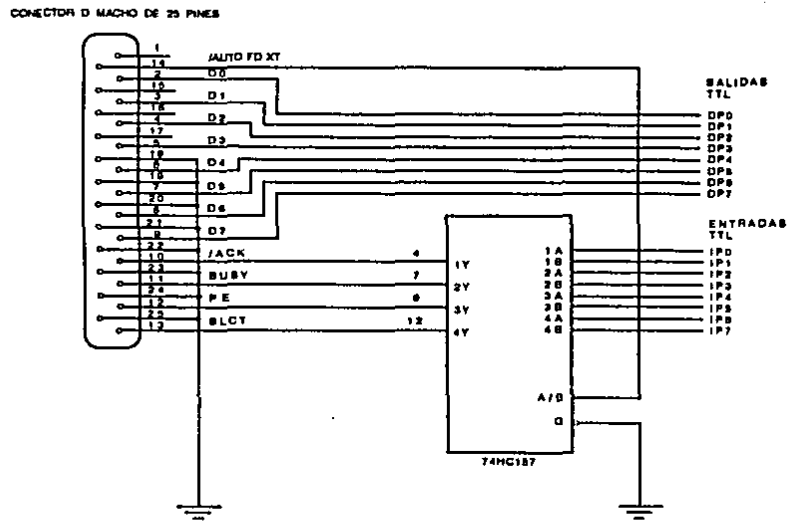
Sin embargo, la solución presentada por la revista Electronics World + Wireless World, en el artículo "Contriving Parallel I/O on the IBM PC" de noviembre del 89, resulta extremadamente simple.

Usan un multiplexor 74HC157 para expander el número de entradas a ocho y leer un byte de cuatro en cuatro bits, (ver la figura de la página siguiente) la señal de control AUTO FD XT es utilizada para seleccionar que bits se van a leer, los cuatro más significativos o los cuatro menos significativos.

Es precisamente esta última opción la que se adoptó para la construcción de nuestro grabador de EPROMs.

La función que regresa el byte leído del exterior, está implementada en el archivo epbio.c, ver capítulo "Software Requerido" de esta tesis.

Leyendo del puerto paralelo
de la impresora, Puerto LPT(N):



DESCRIPCION GENERAL

Para lograr obtener un grabador de EPROMs simple, eficiente y barato, se decidió construirlo de la manera que a continuación se describe.

Los circuitos que llevan a cabo la grabación del EPROM, son elementos discretos, donde la secuencia de operaciones para la escritura del EPROM, es dictada por un simple contador de registro de desplazamiento, utilizado como controlador.

La circuitería puede grabar EPROMs de 2K bytes de capacidad, con voltaje de grabación de 12 Volios, y hace interfaz con el puerto paralelo de la microcomputadora IBM PC, de manera que mantenemos los circuitos necesarios lo más simple posible.

Se buscó implementar el grabador de EPROMs, mediante una mezcla de software y hardware. El software, interviene al proporcionar señales de control a la circuitería del grabador; que no puede funcionar independiente de la computadora; además de alimentar los datos que van a ser escritos en la EPROM.

Lo anterior se programó dentro de un módulo que permanece residente en la memoria principal de la computadora y que es activado al presionar la combinación de las teclas adecuadas.

Este módulo tendrá entonces la capacidad de escribir en la EPROM, programas o datos almacenados en la memoria de la computadora, así como también la hábilidad de leer y cargar en la computadora lo escrito en la EPROM, para de esta manera, poder inspeccionar o verificar el contenido de la EPROM.

El módulo residente en memoria, interactúa con la utileria de Microsoft, llamada DEBUG, y que proporciona (entre algunas de sus caracteristicas), un ambiente controlado para correr programas, modificar registros de el microprocesador de la IBM PC y editar el contenido de su memoria, de manera que servirá como medio para la edición de la información que se grabe en la EPROM.

COMO UTILIZARLO

En este capítulo, describiremos la manera de instalar,usar y desinstalar el software requerido para nuestro grabador de EPROMs.

Instalación.

Primeramente, necesitamos conectar la circuitería necesaria para llevar a cabo la grabación de nuestro EPROM, ver la descripción de este hardware en el capítulo "Hardware Necesario".

Conectamos a la IBM PC, la tarjeta que contiene la circuitería, a través del puerto paralelo, que posee un conector hembra tipo D de 25 pines; es difícil equivocarse y conectarla al serial que si bien tiene un conector del mismo tipo, es por lo general un conector macho.

Colocamos a su vez, la EPROM que se desea grabar es su posición y los niveles de voltaje necesarios para alimentar la tarjeta.

Una vez hecho esto, necesitamos instalar el módulo residente que maneja a la tarjeta donde se encuentra el EPROM, el módulo quedará instalado al correr el programa EPROM.EXE (ver la manera de construir este programa en el capítulo "Software Requerido"),el cual utilizará el puerto paralelo identificado como LPT1. Ejemplo:

```
A> EPROM
```

En algunos equipos, quizá se cuente con más de un puerto paralelo, por lo que si se desea utilizar otro puerto que no se el "default" que es el LPT1, es necesario pasar como perámetro al programa EPROM, la identificación del puerto utilizando mayúsculas.

Por ejemplo, si queremos utilizar el segundo puerto paralelo tendremos que teclear:

```
A> EPROM LPT2
```

No es frecuente, pero en algunas computadoras que poseen solo un puerto paralelo, éste hace uso de las direcciones asignadas para el LPT3, por lo que en este caso será necesario pasar LPT3 como parámetro al instalar el módulo residente.

Una forma de averiguar, si el módulo fue instalado en el puerto correcto es muy simple, si se alimenta la tarjeta antes de instalarse el modulo residente, el LED de lectura deberá estar encendido, una vez que el módulo sea instalado, el LED de lectura deberá apagarse.

Una vez instalado el módulo residente, necesitamos correr el programa donde llevaremos a cabo la edición del programa o datos que se quieran grabar en la EPROM. Se pensó en la utilización del DEBUG, utileria de Microsoft que esta incluida dentro de los discos del sistema operativo MS DOS; pero bien puede usarse el SYMDEB; una versión mejorada del DEBUG, o cualquier programa de este tipo que se desee. La mejor manera de saber si el programa elegido y el módulo residente son compatibles entre sí, es probando.

Mensajes de Error.

Cuando la operación de instalación del módulo residente no tenga éxito, podrán aparecer los siguientes mensajes de error:

"Puerto no disponible".

Si se comete un error al teclear la identificación del puerto, no se tecleé en mayúsculas, o se intente instalar en un puerto diferente a LPT1, LPT2 o LPT3, entonces se desplegará este mensaje de error.

"Demasiados parámetros".

Este mensaje aparecerá, si se intenta hacer la instalación en más de un solo puerto paralelo, lo cual no es posible.

"Versión del DOS inferior ala 3.x".

La interrupción 31h; proporcionada por el sistema operativo DOS (Disk Operating System), utilizada para terminar y dejar residentes en memoria a programas que la llamen; no es encontrada en versiones iniciales del DOS.

"Actualmente instalido".

El programa esta protegido de manera que no pueda instalarse más de una vez, ya que además de no tener objeto, esto ocasionaría la falla del sistema.

Activación e interfaz con el usuario.

Para activar el módulo residente que maneja las operaciones referentes ala EPROM, es necesario presionar simultáneamente las teclas <Cambio-Izquierdo><Cambio-Derecho>.

Al ser activado el módulo, aparecerá una ventana con un menú de tres opciones (menú principal): Leer, Grabar y Cancelar. El usuario podrá seleccionar cualquiera de estas opciones, viajando a través del menú por medio de las teclas del cursor y presionando la tecla de <Retorno>.

Si se elige, ya seala opción de leer como la de grabar, aparecerán nuevas ventanas desplegando una forma pidiendo información que deberá ser llenada por el usuario, para poder llevarse a cabo la operación deseada.

Los datos que se piden son siempre direcciones o cantidades, que deberán ser introducidas en hexadecimal, utilizando exclusivamente los números del 0 al 9 y las letras de la A a la F, no importando si estas últimas son introducidas con mayúsculas o minúsculas. Cuando se presione <Retorno>, indicará que el dato está completo y se pasará a la captura del siguiente dato, también es permitido el uso de la tecla de <Retroceso> o

"backspace" para corregir errores.

Cualquier otra tecla que se presione diferente a las anteriormente citadas, será simplemente ignorada.

Cualquier dato que sea introducido y que no resulte coherente con la operación que se realiza, será marcado como error por medio de un tono en la bocina de la computadora, el dato se borrará y volverá a ser capturado.

Leer.

Si se eligió la opción de leer, aparecerá en la pantalla una ventana con una forma que deberá ser llenada por el usuario, y la cual lucirá como esta:

De: Hasta:

Vaciar en: :

Vo.Bo. Cancelar

El primer dato que se pide en "De", es la dirección dentro de la EPROM, donde se empezará a leer; en "Hasta" se pide la dirección de la última localidad de la EPROM a ser leída. No se admitirán direcciones superiores a 7FF, ni lecturas en forma descendente, esto es, de una dirección alta a una baja.

En la parte marcada como "Vaciar en", se pide el segmento y "offset", que formarán la dirección dentro de la memoria principal de la computadora donde queremos que se empiecen a almacenar los datos leídos de la EPROM.

Dado que lo leído de la EPROM lo cargamos a la memoria principal de la computadora, es recomendable usar un segmento de memoria libre y no por ejemplo el segmento B800, que es el perteneciente al "buffer" de video en sistemas CGA.

Por último, se pide la confirmación de la operación, pudiendo elegir la opción de Vo.Bo. (Visto Bueno) para llevar adelante la lectura o la opción de cancelar para abortar la operación.

Si se decide continuar adelante eligiendo la opción Vo.Bo. , la operación de lectura de la EPROM comenzará, debiéndose encender el LED verde que se encuentra en la tarjeta que contiene a la EPROM, indicando que la EPROM esta siendo leída.

Al finalizar la operación de lectura, aparecerá en la parte baja de la ventana de lectura, el mensaje de "Hecho", indicando que la operación ha sido finalizada.

Grabar

En la opción de grabar, podremos escribir sobre la EPROM programas o datos que se encuentren en la memoria principal de la computadora. Con DEBUG se podrá editar, corregir y probar libremente los programas o datos que serán grabados en la EPROM.

Al seleccionar en el menu principal la opción de grabar, aparecerá la ventana de grabación con la siguiente forma pidiendo información:

Desde: :

Siguientes bytes

Empezando en:

Vo.Bo. Cancelar

En la parte marcada como "Desde", se nos solicita el segmento y el "offset" de la dirección de memoria dentro de la computadora donde empieza la información que se desea grabar en la EPROM. A continuación se pide el número de bytes que se mandarán a escribir, es importante recordar que el dato alimentado está en notación hexadecimal y que el número máximo de bytes que podran ser grabados será de 7FF, que es precisamente la capacidad de la EPROM.

Después se alimenta con la dirección dentro del EPROM donde empezará a escribirse, esto permitirá que pueda escribirse en diferentes partes del EPROM. Sin embargo, no se deberá intentar escribir dos veces

sobre una misma dirección de la EPROM, ya que el resultado será incorrecto.

Si el número de bytes indicados no caben en la EPROM a partir de la dirección introducida en "Empezando en", la dirección no será aceptada y se volverá a pedir una nueva.

Por último, si seleccionamos la opción de Vo.Bo., para proseguir con la grabación, deberá encenderse el LED rojo que se encuentra en la tarjeta que contiene a la EPROM, esto indicará que la EPROM esta siendo grabada. Al terminar la operación, al igual que cuando se lee, aparecerá el mensaje de "Hecho".

Una Buena Idea.

Como una interesante idea, cuando se esté editando la información en el DEBUG y por alguna causa no se termine de hacerlo, se puede mandar a grabar en disco el trabajo realizado hasta el momento con la siguiente secuencia de comandos:

```
-N
nombre del archivo

-R BX
0000

-R CX
número de bytes

-W
```

Con el comando N (de "Name") proporcionamos un nombre al archivo que se guardará en disco, necesitamos también especificar, el tamaño en bytes del archivo, DEBUG usa el par de registros BX:CX para determinar el largo del archivo; puesto que nuestro programa o datos no podrán excederse de los 2K bytes (capacidad máxima de la EPROM), el registro CX será suficiente para almacenar el largo del archivo y BX tendrá que ser puesto en ceros (esto es hecho con el comando R).

Es importante mencionar que el registro IP ("Instruction Pointer") deberá apuntar al primer byte de

nuestro programa o información. Por último, con el comando W (de "Write") completamos la grabación.

Cuando nos encontremos en una nueva sesión, para cargar en memoria, a través del DEBUG, el archivo anteriormente grabado, solo hay que correr el DEBUG con el nombre del archivo como parámetro:

A> DEBUG *nobrearchivo*

El archivo será cargado automáticamente y colocado donde apunta el IP.

En resumen, el conocimiento de los comandos anteriores, nos permitirán guardar también en disco, cualquier trozo de información almacenada en la EPROM, así como la grabación de la EPROM con información que originalmente se encontraba almacenada en disco.

Desinstalación.

Para desinstalar el módulo residente que maneja las operaciones relativas a la EPROM, es necesario oprimir simultaneamente la combinación de tres teclas: <Ctrl> <Cambio-Izquierdo> <Cambio-Derecho>. Al hacerlo, se producirá un tono en la bocina de la computadora, indicando que la desinstalación se ha efectuado. El programa ya no residirá en memoria y no prodrá ser activado nuevamente, a menos que se repita el proceso de instalación.

HARDWARE NECESARIO

En este capítulo, se presenta la descripción y el diagrama eléctrico del hardware necesario para llevar a cabo la grabación y lectura de la EPROM 2716, referirse constantemente al diagrama de la página siguiente durante la explicación.

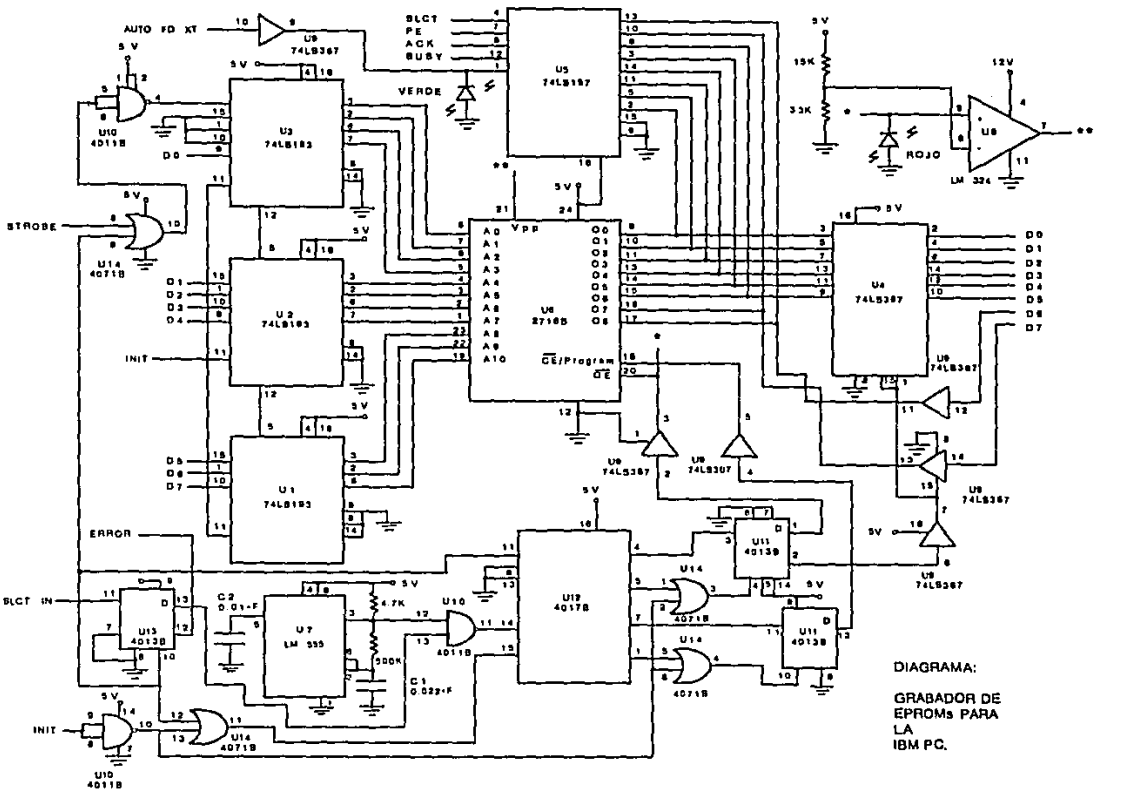
Grabación:

Para grabar una localidad de memoria de una EPROM 2716, es necesario llevar a cabo una secuencia de operaciones, la cual consta de los siguientes pasos:

1. Aplicar la dirección deseada (sobre las líneas de A0-A10 de la EPROM) de la localidad sobre la que se desea escribir.
2. Aplicar en Vpp el nivel de voltaje de programación adecuado.
3. Poner OE ("Output Enable") en alto, de manera que las terminales de datos O0-O7 funcionen como entradas para alimentar el byte que se desea escribir.
4. Aplicar un pulso de bajo a alto, en CE/Program con 50 milisegundos de duración; la localidad seleccionada deberá grabarse con los datos aplicados.

La secuencia anterior empieza a ejecutarse en nuestro circuito, al mandar los bits adecuados a través de las líneas de datos del puerto paralelo (D0-D7) y al aplicar el pulso necesario, por medio de la señal de control INIT, para que los contadores conectados en cascada U1, U2 y U3, almacenen la dirección deseada que será alimentada a la EPROM.

Si se observa, se cuenta con once líneas para direccionar la EPROM (A0-A10) y nosotros intentamos fijar la dirección con solo ocho bits de información (D0-D7), por lo que los tres últimos bits de la dirección son puestos dentro de la rutina de programación "ep_send_add" (Ver capítulo Software Requerido), la cual proporciona el


AUTO FD XT
STROBE
INIT
ERROR
SLCT IN
INIT
SLCT
PE
ACK
BUSY
VERDE
ROJO
15K
33K
12V
LM 324
U1 74LS193
U2 74LS193
U3 74LS193
U4 74LS367
U5 74LS157
U6 27128
U7 LM 555
U8 LM 324
U9 74LS367
U10 4011B
U11 40139
U12 4017B
U13 40139
U14 4071B
C1 0.022-F
C2 0.01-F
4.7K
500K
DIAGRAMA:
GRABADOR DE EPROMs PARA LA IBM PC.

número de pulsos necesarios a través de STROBE de manera que se incremente el contenido del contador en cascada, hasta obtener la dirección deseada.

Después de haber almacenado la dirección en el contador formado por U1, U2 y U3, podemos utilizar las líneas de datos del puerto paralelo para proporcionar el dato que se desea grabar en esa dirección.

Los pasos siguientes, serán controlados por el contador U12 que llevará la secuencia de las operaciones y el cual empieza a operar al mandarse el pulso de control a través de SLCT IN y colocar un 1 lógico en uno de los flip-flops contenidos en U13. Esto habilitará la compuerta NAND, contenida en U10, que permitirá el paso de la señal de reloj (de 40 Hz), proporcionada por U7 a el contador de U12.

El contador contenido en U12 empezará su cuenta manteniendo en alto solo uno de sus pines de salida a la vez, el primer pin de salida de nuestro interés en ponerse en alto será el número 4 que fijará en alto uno de los flip-flops contenidos en U11, el cual a través de un buffer de U9, colocará OE en estado alto y cambiará de estado la salida del OPAM de U8 (el cual se encuentra en configuración de comparador), de manera que se aplique (en el pin 21 de la EPROM) el voltaje necesario para la grabación.

También la señal complementada del flip-flop de U11 habilitará los buffers contenidos en U4 y U9, de manera que dejen pasar a las líneas O0-O7 de la EPROM el dato que se desea grabar.

Después de un período de la señal de reloj (de 40 Hz), o sea 25 milisegundos, el pin que estará en alto será el 7; el cual almacenará este estado en el restante flip-flop de U11, logrando poner CE/Program en estado alto.

Después de haber transcurrido dos períodos de la señal de reloj (50 milisegundos), el pin 1 pasa a ser el activo, borrando uno de los flip-flops de U11 de manera que CE/Program regrese a su estado bajo y se haya de esta forma aplicado el pulso se 50 milisegundos que se requiere para que el dato sea grabado.

A continuación, el pin 5 pasa a ser el alto, removiendo la información de D0-D7 de las líneas O0-O7 de la EPROM, retirando el voltaje de grabación y regresando OE a su estado bajo.

Por último, para completar la secuencia de grabación de un byte, el pin 11 pasa a la actividad, borrando

el flip-flop contenido en U13 de manera que la compuerta NAND quede desactivada y no permita más el paso de la señal de reloj hacia el contador de U12. También al activarse el pin 11 se efectúa la transición necesaria para que la dirección contenida en los contadores U1, U2 y U3 se vea incrementada en uno y sea la próxima dirección a ser grabada.

La señal de estátus, identificada como ERROR, resulta de utilidad a la función de "software" ep_write (contenida en el archivo epbio.c, ver capítulo Software Requerido), al indicar cuando la circuiteria está ocupada realizando las operaciones para una grabación, previniendo al software intentar grabar otro byte antes de terminar de escribir el anterior.

También hay que aclarar la función de la señal de control INIT, la cual al prenderse la computadora, se encontrará en estado bajo, asegurandonos que el contador en U12 se encuentre inicializado y los flip-flops contenidos en U11 se encuentren borrados. Posteriormente la señal es puesta en alto para empezar a operar.

Lectura.

La lectura de una celda de memoria dentro de la EPROM se lleva a cabo mediante la aplicación de la dirección deseada en A0-A7, a través de los contadores U1, U2 y U3 (al igual que cuando se desea grabar), y con OE/Program y CE en estado bajo, de manera que el byte almacenado en la dirección dada aparezca en las líneas O0-O7 de la EPROM.

Para leer el byte leído por la computadora a través de los pines de estatus SLCT, PE, ACK, y BUSY (pertenecinetes al puerto paralelo de la impresora), se hace uso de el mutiplexor 74LS157 contenido en U5 para seleccionar que bits van a ser leídos: los cuatro más significativos o los cuatro menos significativos.

La selección se hace por medio de la señal de control AUTO FD XT, cuando la señal esta en su estado bajo se selecionan los cuatro menos significativos; al estar en su estado alto, los bits seleccionados serán los más significativos.

Nótese que las líneas O0-O7 de la EPROM (en este momento) se encuentran separadas de las líneas D0-D7 del puerto paralelo por medio de seis buffers contenidos en U4 y dos en U9, los cuales se encuentran desconectados.

Una vez leída una localidad de memoria de la EPROM, es necesario mandar un pulso por medio de STROBE para incrementar la dirección, preparandose para leer la siguiente localidad.

Conexiones del Puerto Paralelo de la Impresora.

En la siguiente página repetimos las tablas de información del puerto de la impresora, donde se muestra donde están localizadas las señales dentro del conector de 25 pines, las cuales pertenecen al estándar de interfaz con la impresora denominado "Centronics".

Es importante, al fabricar el cable para conectar la computadora con la tarjeta donde se monten los circuitos, que nos aseguremos que todos los pines de tierra (del 16 al 25) sean conectados a la referencia de nuestro circuito.

PUERTO DE LA IMPRESORA
DE LA IBM PC

Pin	Sentido	"Centronics"
1	Salida	$\overline{\text{STROBE}}$
2-9	Salida	Bits de Datos 0-7
10	Entrada	$\overline{\text{ACK}}$
11	Entrada	BUSY
12	Entrada	PE
13	Entrada	SLCT
14	Salida	$\overline{\text{AUTO}}$ FD XT
15	Entrada	$\overline{\text{ERROR}}$
16	Salida	$\overline{\text{INIT}}$
17	Salida	$\overline{\text{SLCT}}$ IN
18-25	—	Común

Tabla 1

Direcciones de los puertos de la impresora:

	Datos	Estatus	Control
LPT1:	378	379	37A
LPT2:	278	279	27A
LPT3:	3BC	3BD	3BE

Tabla 2

Bits puerto de estatus:

Bit	Función	
D0-D2	No usados	
D3	$\overline{\text{ERROR}}$	pin 15
D4	$\overline{\text{SLCT}}$	pin 17
D5	PE	pin 12
D6	ACK	pin 10
D7	Busy	pin 11

Tabla 3

Bits puerto de control:

Bit	Función
D0	$\overline{\text{STROBE}}$
D1	AUTO FD XT
D2	INIT
D3	SLCT IN
D4	Habilita interrupciones
D5-D7	No usados

Tabla 4

SOFTWARE REQUERIDO

En este capítulo, se proporciona el código fuente en lenguaje C dividido en varios archivos, y las instrucciones necesarias para la compilación del programa, que al correrse deja instalado el módulo necesario para manejar la lectura y escritura de la EPROM.

Se procuró comentar lo máximo posible el código fuente, de manera que el mejor camino para entender como funciona el software resulte ser la lectura del mismo código.

Sin embargo, se consideró necesario comentar y ampliar más la información hacerca de la implementación de la característica de residencia en memoria, ya que quizá esta parte resulte un poco más difícil de comprender.

Contenido de los archivos.

Los archivos que encontramos son:

epbio.c

io_c

screen.c

rw.c

handler.c

eprom.c

eprom.prj

***.h**

Para dar una idea de lo que contiene cada uno (ya que si se desea enterarse de los detalles, es mejor referirse directamente al listado del archivo de interés) describiremos lo que encontrará en cada archivo.

"epbio.c".- Aquí se encontrarán las funciones básicas que se realizan sobre los circuitos que manejan a la EPROM, tales como la selección de una dirección dentro de la EPROM, incrementar esta dirección, leer un byte o escribir un dato.

. Todas estas funciones manipulan directamente las señales de control (INIT,STROBE,AUTO FD XT, SLCT IN), líneas de datos (D0-D7); todas ellas pertenecientes al puerto paralelo de la impresora.

"io.c".- En este archivo se encuentran también funciones básicas, pero referentes a la salida del video y a la entrada de datos.

Dentro del archivo se encuentran funciones para colocar video inverso, desaparecer el cursor, hacer sonar un tono en la bocina de la computadora, lectura de teclas especiales, y captura de números en hexadecimal.

"screen.c".- Aquí se presentan las funciones encargadas de dibujar las ventanas que el modulo hace uso durante su ejecución.

"rw.c".- Contiene principalmente las dos funciones encargadas de realizar la operación de lectura y de escritura de la EPROM.

"handler.c".- Aquí solo se encontrará la función eprom_handler en la cual son llamadas, directa o indirectamente, todas las funciones encontradas en los archivos anteriores. Esta función es el punto de entrada del módulo manejador de la EPROM.

"eprom.c".- Este es el archivo principal, en donde se encuentra la función "main" y se realizan todas las operaciones pertinentes para intentar dejar instalado (residente en memoria) el módulo manejador de la EPROM.

"eprom.prj".- En este archivo se encontrará la información requerida por la utileria "Make" de Turbo C, de manera que compile y encadene automáticamente los archivos necesarios para obtener el archivo ejecutable EPROM.EXE.

Por último, todos los archivos .h, contienen la declaración de las funciones (llamadas prototipos y que proporcionan información al compilador), así como declaración de tipos y definición de constantes necesarias al momento de compilar los archivos .c .

Como Compilarlo.

Ya que se hace uso de algunas funciones exclusivas del compilador de C de Borland, será necesario utilizar Turbo C para la compilación del archivo ejecutable EPROM.EXE.

Específicamente, utilizando Turbo C 2.0, aseguramos no tener ningun problema.

Es importante especificar, que el programa fue pensado para ser compilado bajo un particular módelo de compilación en C, por lo que deberá seleccionarse el modelo denominado como "Small" para un correcto funcionamiento.

Como primer paso para llevar a cabo la compilación, necesitamos asegurarnos que los archivos que contienen el código fuente, así como los archivos de cabecera (*.h), se encuentren en el directorio principal de Turbo C.

Dentro del ambiente integrado de Turbo C (el que se obtiene al correr TC.EXE), seleccionar el menú de "Project", y en la opción de "Project Name", introducir el nombre de *eprom.pri*.

Por último, en el menú de "Compile" elegir la opción de "Make", la cual se encargará de compilar y encadenar automaticamente los archivos necesarios para obtener el ejecutable EPROM.EXE .

Residencia en Memoria.

Dado que el sistema operativo DOS tiene una de sus limitaciones en su incapacidad para soportar "mulutasking", el mismo DOS trata de alguna forma de suavizar esta carencia, proporcionando la función 31h, la

cual termina, pero deja residente en memoria a el programa que la llama.

Un programa residente en memoria, es aquel programa que termina su ejecución y regresa el control al sistema operativo DOS, pero que no le regresa la memoria que el programa mismo ocupaba, la cual ya no prodrá ser utilizada por DOS; permitiendo que el programa pueda ser llamado instantaneamente, sin necesidad de ser cargado nuevamente.

Al presionar las teclas adecuadas, el programa saltará a la actividad interrumpiendo la aplicación que se encontraba corriendo; al terminar de operar, el programa residente devolverá el control a la aplicación anterior como si nada hubiese sucedido.

Al escribir un programa residente en memoria, se necesita tomar en cuenta varios aspectos de la arquitectura y funcionamiento interno de la IBM PC.

Se tendrán que sortear problemas, tales como:

- La correcta sustitución de las rutinas de servicio asociadas a las interrupciones adecuadas, que nos permitan llevar a la actividad a nuestro programa residente.

- Diseño del programa residente de manera que sea compatible con el sistema operativo DOS, para que el programa residente pueda hacer uso de los servicios que el DOS ofrece.

- Coexistir con la actividad del BIOS con el disco (la cual no puede ser interrumpida en ningún momento).

- Y por último, coexistir con la aplicación que el programa residente interrumpe.

Serán estos los puntos que a continuación trataremos.

Rutinas de Servio para las Interrupciones.

En la IBM PC, se hace fuerte uso del sistema de interrupciones para controlar la microcomputadora,

cada interrupción se encuentra numerada y tiene asociada una rutina de servicio, la cual es ejecutada cada vez que se invoque dicha interrupción.

Dentro de la memoria baja de la IBM PC empezando en la dirección 0000:0000, encontramos la tabla de vectores para las interrupciones, esto es, las direcciones de las rutinas de servicio asociadas a cada interrupción.

Dado que la rutina de servicio puede encontrarse en cualquier parte de la memoria, se necesitan cuatro bytes para formar la dirección completa (segmento y offset), de manera que los primeros 1024 bytes de la memoria de la IBM PC, forman la tabla de vectores para un total de 256 interrupciones posibles.

Esto nos da la posibilidad de sustituir la rutina de servicio asociada a una determinada interrupción, por otra rutina que nosotros hayamos preparado para el caso. Lo anterior lo hacemos interviniendo la tabla de vectores donde se encuentran las direcciones de las rutinas de servicio.

Esta es la técnica que usaremos, para instalar nuestras propias rutinas de servicio y checar si las teclas apropiadas han sido presionadas (cuya codificación se encuentra en la variable "hotkey", dentro del archivo "eprom.c"), y si procede o no, llamar al programa residente.

En el archivo "eprom.c" son cambiadas las direcciones de las interrupciones 9h y la 13h, donde la primera es la interrupción de "hardware" generada al presionar cualquier tecla, y la segunda es el número de la interrupción que utiliza en BIOS ("Basic Input Output System") para brindar los servicios de disco.

Las rutinas de sevicio originales, pertenecientes a las interrupciones 9h y 13h, no pueden perderse, ya que no habría quien proporcionara los servicios que éstas ofrecen, por lo que simplemente son asignadas a otro número de interrupción, en este caso la 60h y la 61h (las cuales, normalmente no son utilizadas), para posteriormente ser llamadas.

Esto se lleva a cabo en la líneas:

```
setvect(60,getvect(9));
setvect(61,getvect(13));
```

Para intervenir la tabla de vectores, se hace uso de la función **setvect**, contenida en la biblioteca de Turbo C, esta función desactiva las interrupciones durante el cambio, de manera que resulte segura la intervención de la tabla de vectores.

La función **getvect**, también de la biblioteca de Turbo C, en forma similar regresa la dirección de la rutina de servicio de la interrupción indicada.

Las rutinas de servicio para las interrupcines 9h y 13h, son instaladas por las líneas:

```
setvect(9,int9);
setvect(13,int13);
```

Nótese que en la implementación de las funciones int9 e int13, se antepone el modificador (que Turbo C incluye especialmente) llamado **interrupt**. Este modificador, permite que una función pueda ser usada como rutina de servicio de una interrupción, de manera que el código en ensamblador generado por las funciones, deshabiliten al entrar la ocurrencia de interrupciones, las habiliten de nuevo al salir, y regresen con la instrucción IRET ("Interrupt Return").

También obsérvese que lo primero que estas nuevas rutinas de servicio hacen, es llamar a la antigua rutina mediante la función **geninterrupt**, de manera que la rutina de servicio original,lejos de perderse, en realidad solo es ampliada.

Dentro de la nueva rutina de servicio para la interrupción 9h, se checa una bandera llamada "busy", antes de activar el programa residente (mediante la llamada a la función eprom_handler), esta bandera es puesta "busy=TRUE" (verdadero), cada vez que el programa residente esta activo de manera que se impidan llamadas recursivas del programa.

También queremos aclarar aquí la manera en que se conoce que teclas fueron oprimidas, lo cual es llevado a cabo mediante la inspeción de la palabra (dos bytes) que el BIOS usa para estatus, localizada en la dirección de memoria baja 0000:0417, donde se codifica el estado de algunas teclas especiales (las cuales usamos

para activar y desactivar el programa residente).

Compatibilidad con el DOS.

Uno de los problemas principales, es la caracteristica, de que el código del DOS no es re-entrante, esto significa, que dos programas no pueden llamar servicios del DOS al mismo tiempo.

Las rutinas del sistema operativo DOS, normalmente comienzan su ejecución guardando en memoria la localización del "stack" actual SS:SP, para después cambiarse a uno de los "stacks" internos que utiliza el DOS.

Dado que la activación de nuestro programa residente, es un evento asincrono, éste puede interrumpir en cualquier momento a la aplicación que se este corriendo.

Imaginemos que se intenta activar nuestro programa mientras el código del DOS se encuentra en ejecución, y que el programa residente llama a su vez los servicios del DOS, es probable que al ser invocados los servicios del DOS por el programa residente se haga uso del mismo "stack" que el DOS estaba utilizando antes de ser interrumpido; se utilizará el "stack" para guardar datos, y el programa progresará normalmente, hasta que se regrese el control a la aplicación interrumpida, ésta se encontrará con un "stack" del DOS alterado de manera que los resultados que se obtengan de ahí en adelante sean totalmente impredecibles.

Una opción, es que nuestro programa no haga uso de los servicios del DOS, sin embargo, programar de esta manera resulta una tarea casi imposible, ya que muchas de las funciones de la biblioteca de C, hacen su trabajo mediante llamadas al sistema operativo.

La solución que se adoptó, es la de guardar en memoria el contenido completo del "stack" en uso, antes de hacer nuevas invocaciones al DOS y restaurar dicho "stack", antes de regresar el control a la aplicación interrumpida.

Esto es hecho en las lineas:

```
for (i=0; i<128; i++)
        buffer[i]=peekb(dos_ss,dos_sp+i);
for (i=0; i<128; i++)
        pokeb(dos_ss,dos_sp+i,buffer[i]);
```

Actividad de Disco por el BIOS.

Un momento vulnerable, en el cual no debemos interrumpir el proceso, es cuando se esta haciendo uso del disco, todos los servicios de disco que proporciona el BIOS hacen uso de la interrupción 13h, de manera que el problema fue remediado agregando a la rutina de servicio de la interrupción 13h, la bandera "in_bios", la cual es puesta "in_bios=TRUE" (en verdadero), cada vez que se inicia un servio de disco, y regresa a "in_bios=FALSE" (falso) al finalizar el servicio.

De esta forma, nuestra rutina de servicio de la interupción 9h puede checar esta bandera de manera que no se active el programa residente cuando nos encontremos en una operación de disco.

Coexistir con la Aplicación Interrumpida.

Dado que nuestro programa residente puede interrumpir casi en cualquier momento la aplicación que se este ejecutando, y puesto que después se regresará el control a esta aplicación, es necesario que se restaure el estado original de la máquina, así como también, el estado del video.

Es importante que todos los registros del microprocesador sean guardados y restaurados, por el programa residente. Los únicos registros que no tienen que ser guardados explícitamente, son CS, IP y las banderas del microprocesador, los cuales son guardadas y restauradas automáticamente al entrar y salir de una rutina de servicio de una interrupción.

Para preservar el video, éste es guardado por la rutina "eprom_handler", a través de la función de Turbo C, **gettext**, y restaurado al terminar por medio de la función **puttext**.

La posición del cursor, es guardada, en las variables "cursor_x" y "cursor_y", para después ser restaurada.

Parrafos que se Dejaran Residentes.

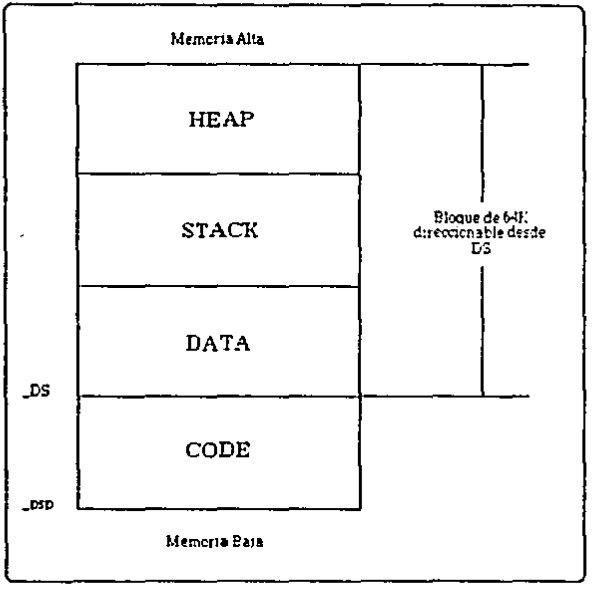
La función 31h, de los servicios que proporciona el DOS, a través de la interrupción 21h, es la utilizada para terminar y dejar a nuestro programa residente en memoria.

La función 31h, necesita que se le pase el número de parrafos (1 parrafo igual a 16 bytes) que serán dejados residentes. Para determinar tal dato, se sigue el siguiente procedimiento (observar durante la explicación, la figura de la siguiente página que muestra la organización de la memoria para el modelo de compilación "Small"):

Se utiliza la función de la biblioteca de C, llamada **sbrk**, la cual regresará el valor del offset, donde se acaba el area de STACK, este valor es convertido al número de parrafos equivalentes, y sumado al valor contenido en DS, el valor resultante será el valor del segmento donde empieza el area del HEAP.

Después, se resta el valor contenido en la variable global predefinida **_psp**, de manera que ahora tenemos la cantidad de parrafos contenidos desde el principio del programa hasta donde se acaba el STACK; por último, se añade, el número de parrafos asignados a la memoria que se usa para almacenamiento dinámico y que esta contenida en el HEAP.

El número de parrafos resultante, es almacenado en la variable llamada "size".



Organización de la memoria para el modelo "Small" de compilación en C.

Desinstalación

Para finalizar, la desintalación del programa residente, es llevada a cabo cuando se presiban las teclas adecuadas (codificadas en la variable "coldkey"), se restaurará en la tabla de vectores las direcciones de las rutinas de servicio originales, y mediante la función número 49h de los servicios del DOS, se liberará la memoria ocupada por el programa.

LISTADO
ARCHIVO: EPROM.C

```
#include<dos.h>
#include<stdio.h>
#include<string.h>
#include<alloc.h>

#define HEAPSIZE 128

typedef enum { FALSE,TRUE} boolean;

char buffer[128];

unsigned int dos_ss,dos_es,dos_ss,dos_sp,indos_seg,indos_off,c_ss,c_sp
             ,c_ds,c_es,c_dta_off,c_dta_seg,d_dta_off,d_dta_seg,
             myax,mybx,mycx,mydx,mybp,mysi,mydi,myds,myes;

unsigned int hotkey,coldkey,spkeys,psp;

unsigned int i,lpt_data,lpt_status,lpt_control;

boolean in_bios=FALSE,busy=FALSE;

void deactivate(void);

void interrupt int9  (unsigned bp, unsigned di, unsigned si,
                      unsigned ds, unsigned es, unsigned dx,
                      unsigned cx, unsigned bx, unsigned ax )
{
geninterrupt(60);


if ( (busy==FALSE) && (in_bios==FALSE) )
   {
    busy=TRUE;

    spkeys=peek(0x0000,0x0417);

    if ( coldkey == (coldkey & spkeys) )
    {
    dos_ss=_SS;
    dos_sp=_SP;

    _SS=c_ss;
    _SP=c_sp;

    myax=ax;
    mybx=bx;
    mycx=cx;
    mydx=dx;
```

```
mybp=bp;
mysi=si;
mydi=di;
myds=ds;
myes=es;

for(i=0; i<128 ;i++)
      buffer[i]= peekb(dos_ss,dos_sp+i);

ds=c_ds;
es=c_es;

deactivate();

for(i=0; i<128 ; i++)
      pokeb(dos_ss,dos_sp+i,buffer[i]);

ax=myax;
bx=mybx;
cx=mycx;
dx=mydx;
bp=mybp;
si=mysi;
di=mydi;
ds=myds;
es=myes;

_SS=dos_ss;
_SP=dos_sp;

}

else if ( hotkey == (hotkey & spkeys) )
{

dos_ss=_SS;
dos_sp=_SP;

_SS=c_ss;
_SP=c_sp;

myax=ax;
mybx=bx;
mycx=cx;
mydx=dx;
mybp=bp;
mysi=si;
mydi=di;
```

```
    myds=ds;
    myes=es;

    for(i=0; i<128 ;i++)
        buffer[i]= peekb(dos_ss,dos_sp+i);

    ds=c_ds;
    es=c_es;

    eprom_handler();

    for(i=0; i<128 ; i++)
        pokeb(dos_ss,dos_sp+i,buffer[i]);

    ax=myax;
    bx=mybx;
    cx=mycx;
    dx=mydx;
    bp=mybp;
    si=mysi;
    di=mydi;
    ds=myds;
    es=myes;

    _SS=dos_ss;
    _SP=dos_sp;

    }

  busy=FALSE;
  } /* if !busy */

} /* int9 */

void interrupt int13()
{
in_bios=TRUE;
geninterrupt(61);
in_bios=FALSE;
} /* int13 */

void desactivate(void)
{
 union REGS regs;
 struct SREGS sregs;

 setvect(9,getvect(60));
 setvect(13,getvect(61));
```

```
setvect(61,getvect(62));

regs.h.ah=0x49;
sregs.es=psp;
intdosx(&regs,&regs,&sregs);
beep();

} /* deactivate */

void tsr (unsigned size)
{
 union REGS r;
 r.h.ah=0x31;
 r.h.al=0;
 r.x.dx=size;
 int86(0x21,&r,&r);
} /* tsr */


int main(int argc,char *argv[])
{
unsigned int break_off,paragraphs;

switch (argc) {

case 1: lpt_data=0x0378;
        lpt_status=0x0379;
        lpt_control=0x037A;
        break;

case 2: if (strcmp(argv[1],"LPT1")==0) {
                lpt_data=0x0378;
                lpt_status=0x0379;
                lpt_control=0x037A;
                break;
                        }

        if (strcmp(argv[1],"LPT2")==0) {
         lpt_data=0x0278;
                lpt_status=0x0279;
                lpt_control=0x027A;
                break;
                        }

        if (strcmp(argv[1],"LPT3")==0) {
         lpt_data=0x03BC;
                lpt_status=0x03BD;
```

```
                    lpt_control=0x03BE;
                    break;
                        }

        cputs("Puerto no disponible");
        exit(2);

default: cputs("Demasiados parámetros");
         exit(1);

                    ) /* switch */

ep_init();

psp=_psp;

c_ss=_SS;
c_sp=_SP;
c_es=_ES;

hotkey=0x0003;
coldkey=0x0007;

if (_osmajor < 3) { printf("Version de DOS, menor que 3.x\n");
                    exit(1);
                  }
if( getvect(61)!=getvect(62) ) { printf("Programa actualmente ya instalado\n");
                                 exit(2);
                               }
setvect(60,getvect(9));
setvect(9,int9);
setvect(61,getvect(13));
setvect(13,int13);

break_off=(int) sbrk(0);
break_off+=0x0F;
break_off>>=4;
paragraphs=break_off + _DS - _psp + HEAPSIZE;

printf("** Grabador de EPROMS **\n\n");
printf("Hecho por :\nSergio A. Diaz Del Rio.\n");
printf("       -oOo-\n\n");
printf("Ahora instalado.\n");
printf("Presione <Cambio-Izquierdo> <Cambio-Derecho> para activar.\n");

ksr(paragraphs);

} /* main */
```

LISTADO
ARCHIVO: HANDLER.C

```
#include<conio.h>
#include "rw.h"
#include "io.h"

char buf[31*12*2];

/* La función eprom_handler:
  Es la función que despacha las opciones
  del menú principal, Leer, Grabar o Cancelar.
*/

void eprom_handler (void)
{
 int cursor_x,cursor_y;
 boolean quit=FALSE;

 cursor_x=wherex();      /* Guardando la posición del cursor */
 cursor_y=wherey();
 cursor_out();
 window(1,1,80,25);
 gettext(25,7,55,18,buf); /* Guardando el contenido del video */

 do
 {
  prt_main_win();
  switch(main_menu())
  {
  case 1:
          myreadstuff();  /* Leer */
          break;

  case 2: mywritestuff(); /* Grabar */
          break;

  case 3: quit=TRUE;      /* Cancelar */
          break;
  }
  puttext(25,7,55,18,buf);

 }while(!quit);

 window(1,1,80,25);          /* Restaurando el video    */
 cursor_in();
 gotoxy(cursor_x,cursor_y); /* Restaurando la posición del cursor */

} /* eprom_handler */
```

LISTADO
ARCHIVO: RW.H

```
void inc_mem_addr(int *seg,int *offst);
int get_ep_addr(char x,char y);
void myreadstuff (void);
void mywritestuff(void);
```

LISTADO
ARCHIVO: RW.C

```
#include <dos.h>

#include "epbio.h"
#include "io.h"
#include "screen.h"
#include "rw.h"


/* La función inc_mem_addr:
  Incrementa en uno la dirección de memoria
  contenida en  seg:offst.
*/

void inc_mem_addr(int *seg,int *offst)
{
 *offst=(*offst) + 1;
 if ( (*offst) == 0x0000) (*seg)= (*seg)+1;
}



/* La función get_ep_addr:
  Lee en las coordenadas (x,y) de la pantalla,
  una dirección válida para la EPROM.
*/



int get_ep_addr(char x,char y)
{
 boolean error;
 int addr;

 do
 {
  error=FALSE;
  addr=get_hex_nbr(x,y,3);
  if (addr > 0x07FF) {
   beep();
   error=TRUE;
                     }
 }while (error);

 return(addr);
} /* get_ep_addr */
```

```
/* La función myreadstuff:
  Es donde se contruye la ventana de Leer, se capturan
  todos los datos necesarios y si el final se da el
  Vo.Bo. se efectua la lectura de la EPROM.
*/

void myreadstuff (void)
{
 boolean wrong;
 int segment,offset,from,to;
 char c;

 read_win();

 from=get_ep_addr(8,2);
 do
 {
 wrong=FALSE;
 to=get_ep_addr(20,2);
 if (to < from) {
     beep();
     wrong=TRUE;
                }
 }while(wrong);

 segment=get_hex_nbr(15,4,4);
 offset=get_hex_nbr(22,4,4);

 if (go_cancel()==GO) {

 ep_sendadd(from);

  for (;from<=to;from++) {
    c=ep_read(from);
    pokeb(segment,offset,c);    /* Vacia en memoria ppal. el byte leído */
    inc_mem_addr(&segment,&offset);
    ep_nextadd();
                          }
    gotoxy(12,10);
    blink_video();
    cputs("Hecho");
    norm_video();
    do
    ;
    while(!kbhit());
    getch();
```

```
                    }

window(1,1,80,25);

} /* myreadstuff */


/* La función mywritestuff:
  Es donde se contruye la ventana de Grabar,se capturan
  todos los datos necesarios y si al final se da el
  Vo.Bo. se efectua la grabación sobre la EPROM.
*/


void mywritestuff(void)
{
boolean error;
int segment,offset,amount,loop,start,limit;
char c;

write_win();
segment=get_hex_nbr(11,2,4);
offset=get_hex_nbr(18,2,4);
amount=get_ep_addr(15,4);

do
{
error=FALSE;
start=get_ep_addr(18,6);
limit=start+amount;
if ( (limit > 0x07FF) || (limit < start) ) {
 beep();
 error=TRUE;
                                          }
}while (error);

if ( go_cancel()==GO)  {

 ep_sendadd(start);

 for(loop=0;loop<amount;loop++)
  {
   c=peekb(segment,offset); /* Toma de la memoria ppal. el byte a grabar */
   ep_write(c);
   inc_mem_addr(&segment,&offset);  /* Siguiente byte a grabar */
  }
   gotoxy(12,10);
   blink_video();
```

```
  cputs("Hecho");
  norm_video();
  do
  ;
  while(!kbhit());
  getch();
                        }

window(1,1,80,25);

} /* mywritestuff */
```

LISTADO
ARCHIVO: SCREEN.H

```
#define VERTICAL_DBAR      179
#define HORIZONTAL_DBAR    196
#define UPPER_RIGHT        191
#define UPPER_LEFT         218
#define LOWER_RIGHT        217
#define LOWER_LEFT         192
#define LEFT_ARROW          75
#define RIGHT_ARROW         77

#define GO  1
#define CAN 2

void frame(int x_top, int y_top, int x_bottom, int y_bottom,char *title);

void mainwin_op (int idx);

void prt_mainwin(void);

void go_can_op(int num);

int go_cancel(void);

void read_win(void);

void write_win(void);
```

LISTADO
ARCHIVO: SCREEN.C

```
#include<graphics.h>
#include<stdio.h>
#include"io.h"
#include"screen.h"


/* La función frame:
  Dibuja un marco para una ventana, y coloca centrado en la parte
  superior, el título de la ventana utilizando video inverso
*/

void frame(int x_top, int y_top, int x_bottom, int y_bottom,char *title)
{
 int len,loop,indent;

 len= x_bottom - x_top;
 gotoxy(x_top,y_top);
 wrtchar(1, UPPER_LEFT);
 gotoxy(x_top+1,y_top);
 wrtchar(len-1,HORIZONTAL_DBAR);
 gotoxy(x_bottom,y_top);
 wrtchar(1,UPPER_RIGHT);

 for(loop=y_top+1; loop <= y_bottom-1; loop++)
 {
  gotoxy(x_top,loop);
  wrtchar(1,VERTICAL_DBAR);
  gotoxy(x_top+len,loop);
  wrtchar(1,VERTICAL_DBAR);
 }

 gotoxy(x_top,y_bottom);
 wrtchar(1,LOWER_LEFT);
 gotoxy(x_top+1,y_bottom);
 wrtchar(len-1,HORIZONTAL_DBAR);
 gotoxy(x_bottom,y_bottom);
 wrtchar(1,LOWER_RIGHT);
 indent=(x_bottom-x_top-strlen(title))/2;
 gotoxy(x_top+indent,y_top);
 reverse_video();
 cputs(title);
 norm_video();
} /* frame */

/* La función mainwin_op:
  Re-escribe en la ventana del menu principal la opción
  que se le indique en idx.
```

```
*/

void mainwin_op (int idx)
{
 switch(idx)
 {
 case 1:gotoxy(4,2);
        cputs("Leer");
        break;
 case 2:gotoxy(10,2);
        cputs("Grabar");
        break;
 case 3:gotoxy(18,2);
        cputs("Cancelar");
        break;
 }
} /* mainwin_op */

/* La función prt_main_win:
  Dibuja la ventana y contenido del menu principal */

void prt_main_win(void)
{
 window(1,1,80,25);
 frame(25,7,54,11,"EPROM");
 window(26,8,53,10);
 clrscr();
 mainwin_op(1);
 mainwin_op(2);
 mainwin_op(3);
} /* prt_main_win */

/* La función main_menu:
  Contruye la ventana del menu principal y regresa
  la opción que sea seleccionada.
*/

int main_menu()
{
 int op=1,oldop=2;

 do
 {
 reverse_video();       /* Actualizando el menu */
 mainwin_op(op);
 norm_video();
 mainwin_op(oldop);
 switch(readkey())
```

```
{
case LEFT_ARROW: oldop=op;
                 if (op!=1) op-=1;
                   else  op=3;
                 break;

case RIGHT_ARROW: oldop=op;
                 if (op!=3) op+=1;
                       else  op=1;
                 break;

case CR: return(op);

}
}while (TRUE);
} /* main_menu */

/* La función go_can_op:
  Re-escribe (en las ventanas hijas del menú principal)
  las opciones de Vo.Bo. y Cancelar.
*/

void go_can_op(int num)
{
 switch(num)
 {
 case 1: gotoxy(6,8);
         cputs("Vo.Bo.");
         break;
 case 2: gotoxy(15,8);
         cputs("Cancelar");
         break;
 }
} /* go_can_op */

/* En la función go_cancel:
  Se presentan las opciones de Vo.Bo. y Cancelar,
  además que regresa el número de la opción que haya
  sido seleccionada .
*/

int go_cancel(void)
{
 int op=1,oldop=2;
 do
 {
 reverse_video();        /* actualizando el menu */
 go_can_op(op);
```

```
norm_video();
go_can_op(oldop);
switch(readspkey())
{
 case LEFT_ARROW:  oldop=op;
                   if (op!= 1 ) op-=1; /* 1 es Vo.Bo.  */
                        else   op=2; /* 2 es Cancelar */
                   break;
 case RIGHT_ARROW: oldop=op;
                   if (op!=2)  op+=1;
                        else  op=1;
                   break;
 case CR:       return(op); /* Con el retorno CR  */
}                  /* se elije la opción */
}while(TRUE);
} /* go_cancel */

/* La función read_win:
  Simplemente dibuja la ventana de Leer en pantalla.
*/

void read_win(void)
{
 window(1,1,80,25);
 frame(26,8,55,18,"Leer");  /* Dibuja su marco y título */
 window(27,9,54,17);
 clrscr();
 window(27,9,54,18);
 gotoxy(4,2);
 cputs("De:    Hasta:");  /* Llena la ventana con su contenido */
 gotoxy(4,4);
 cputs("Vaciar en:    :");
 go_can_op(1);
 go_can_op(2);
 }

 /* La función write_win:
   Dibuja la ventana de Grabar en pantalla */

 void write_win(void)
 {
  window(1,1,80,25);
  frame(26,8,55,18,"Grabar"); /* Dibuja su marco y título */
  window(27,9,54,17);
  clrscr();
  window(27,9,54,18);
  gotoxy(4,2);
  cputs("Desde:    :");   /* Llena la ventana con su contenido */
```

```
gotoxy(4,4);
cputs("Siguientes   bytes");
gotoxy(4,6);
cputs("Empezando en:");
go_can_op(1);
go_can_op(2);
} /* write_win */
```

LISTADO
ARCHIVO: IO.H

```
#define CR (-13)
#define BS (-8)

void wrtchar(int times, char c);

void cursor_out(void);

void cursor_in(void);

void reverse_video(void);

void norm_video(void);

void blink_video(void);

char readspkey(void);

int get_hex_nbr(char col, char row, int len);

typedef enum (FALSE,TRUE) boolean;
```

LISTADO
ARCHIVO: IO.C

```
#include<dos.h>
#include<conio.h>
#include<stdio.h>
#include"io.h"


/* La función wrtchar:
   Escribe el carácter contenido en "c" tantas veces
   como lo indique times.
*/

void wrtchar(int times, char c)
{
 union REGS r;

 r.h.ah=0x09;
 r.h.al=c;
 r.h.bl=0x07;
 r.h.bh=0;
 r.x.cx=times;
 int86(0x10,&r,&r);
} /* wrtchar */

/* La función beep:
   Marca un tono en la bocina de la computadora */

void beep(void)
{
 putchar('\x07');
}

/* La función cursor_out:
   Desaparece el cursor de la pantalla */

void cursor_out(void)
{
 union REGS r;

 r.h.ah=0x01;
 r.h.ch=0x20;
 int86(0x10,&r,&r);
} /* cursor_out */

/* La función cursor_in:
   Hace aparecer el cursor en la pantalla */

void cursor_in(void)
{
```

```
union REGS r;

r.h.ah=0x01;
r.h.ch=0x06;
r.h.cl=0x07;
int86(0x010,&r,&r);
} /* cursor_in */

/* La función reverse_video:
  Coloca el video de manera que el fondo, de los carácteres
  escritos sea brillante y los carácteres mismos sean negros.
*/

void reverse_video(void)
{
 textcolor(BLACK);
 textbackground(LIGHTGRAY);
} /* inverse_video */

/* La función norm_video:
  Coloca el fondo de los carácteres escritos en negro,
  y los carácteres blancos.
*/

void norm_video(void)
{
 textcolor(WHITE);
 textbackground(BLACK);
} /* norm_video */

/* La función blink_video:
  Hace que los carácteres escritos parpaden */

void blink_video(void)
{
 textattr(WHITE + (BLACK<<4) + BLINK);
}

/* La función readspkey:
  Regresa el código de "scan", si es que se ha oprimido una
  de las teclas llamadas especiales (aquellas que regresan
  un código ASCII de 0). También regresa un CR si se
  presionó Retorno.
*/

char readspkey(void)
{
 char spkey;
```

```
do
{
do
;
while (!kbhit());
spkey=getch();
if(spkey==0)
{
 spkey=getch();
 return(spkey);
}
if(spkey==13) return (CR);
}
while(TRUE);
} /* readspkey */


/* La función get_hex_digit:
  Regresa el valor numérico, de cualquier dígito hexadecimal que sea
  presionado, no importando si los dígitos de la A a la F sean
  introducidos en mayúsculas o minúsculas.
    También si se presiona retorno o "backspace" se regresa CR o BS
  respectivamente.
*/

int get_hex_digit(void)
{
char c;

do
{
c=getch();
if (c==13) return (CR);
if (c==8)  return (BS);
if (c>= '0' && c<= '9')  return (c - '0');
if (c>= 'A' && c<= 'F')  return (c - '7');
if (c>= 'a' && c<= 'f')  return (c - 'W');
}
while (1);
} /* get_hex_digit */

/* La función get_hex_nbr:
  Regresa la lectura de un número en hexadecimal de "len" dígitos.
  La lectura se lleva a cabo en las coordenadas (col,row).
  Se puede utilizar la tecla de "backspace" para la edición de
  de los datos.
*/
```

```
int get_hex_nbr(char col,char row,int len)
{
 unsigned int hex_dgt=0x0000;
 int digit,nbrdgts=0;

 do
 {
  gotoxy(col,row);
  printf("%0*X",len,hex_dgt);

  gotoxy(col+len-1,row);
  digit=get_hex_digit();
  switch (digit) {

  case CR : return (hex_dgt);

  case BS : hex_dgt>>=4;
            if (nbrdgts>0) nbrdgts--;
            break;

  default: hex_dgt<<=4;
            hex_dgt+=digit;
            nbrdgts++;
            break;
                  } /* end switch */
 }
 while (nbrdgts < len);
 gotoxy(col,row);
 printf("%0*X",len,hex_dgt);
 return(hex_dgt);
} /* get_hex_nbr */
```

LISTADO
ARCHIVO: EPBIO.H

```
void wait (int times);
unsigned char ep_read();
int ep_sendadd(unsigned int add);
void ep_nextadd(void);
void ep_init(void);
void ep_write(char data);
```

LISTADO
ARCHIVO: EPBIO.C

```
#include <dos.h>
#include "epbio.h"

extern int lpt_data,lpt_status,lpt_control;

void wait (int times)
{
int i;
for (i=0; i<times ; i++)
;
}


/* La función ep_nextadd:
  Incrementa en 1 la dirección actual de la EPROM.
*/

void ep_nextadd(void)
{
 outportb(lpt_control,0x0E);    /* STROBE = 1 */
 wait(200);
 outportb(lpt_control,0x0F);    /* STROBE = 0 */
}



/* La función ep_sendadd:
  Fija la dirección deseada en el contador formado por U1,U2 y U3,
  de manera que sea ésta la nueva dirección actual dentro de la EPROM.
*/

int ep_sendadd(unsigned int add)
{
 char offst,plus,byteadd;

 if (add > 0x07FF) return(1);

 offst= add & 0x0007;
 byteadd = add >> 3;

 outportb(lpt_data,byteadd);
 outportb(lpt_control,0x0B);  /* INIT = 0 */
 outportb(lpt_control,0x0F);  /* INIT = 1 */

 for (plus=0; plus<offst; plus++) ep_nextadd();

} /* ep_sendadd */
```

```
/* La función ep_init:
  Inicializa el grabador de EPROMs fijando:

   STROBE    = 0
   AUTO FD XT = 0
   INIT      = 1
   SLCT IN   = 0

   y pone en 0000 la direccion
   actual dentro de la EPROM.
*/

void ep_init(void)
{
 outportb(lpt_control,0x0F);
 ep_sendadd(0x00);

} /* ep_init */


 /* La función ep_read:
   Regresa el byte leido de la dirección actual de la EPROM */

 unsigned char ep_read()
 {
  int i;
  unsigned char data=0,result=0;

   data=inportb(lpt_status);
   data>>=4;
   result=data;
   outportb(lpt_control,0x0D);       /*AUTO FD XT = 1 */
   data=inportb(lpt_status);
   data&=0xF0;
   result|=data;
   result^=0x88;
   wait(500);
   outportb(lpt_control,0x0F);   /* AUTO FD XT = 0 */
   return (result);
  } /* epread */


  /* La función ep_write:
    Escribe en la dirección actual de la EPROM, el byte
    contenido en "data"
  */
```

```
void ep_write(char data)
{
 char busy;

 outportb(lpt_data,data);
 outportb(lpt_control,0x07);   /* SLCT IN = 1 */
 outportb(lpt_control,0x0F);   /* SLCT IN = 0 */

 do
 {
  busy=inportb(lpt_status); /* Checa la señal de estatus ERROR,     */
  busy=busy&0x08;           /* la cual indica si el grabador se     */
 }                  /* encuentra ocupado.                  */
 while (busy!=0x08);        /* Espera hasta que el grabador finalice */
                            /* su función                          */

} /* ep_write */
```

LISTADO
ARCHIVO: EPROM.PRJ

eprom.c
handler.c (io.h,rw.h)
rw.c (epbio.h,io.h,screen.h,rw.h)
screen.c (io.h,screen.h)
epbio.c (epbio.h)
io.c (io.h)

CONCLUSIONES

Un aspecto interesante en este proyecto, fue la explotación del puerto paralelo de la impresora de una manera generalmente inusual, ya que se utilizó como un puerto bidireccional. La forma de hacerlo fue muy simple, y se obtuvieron buenos resultados, alentandonos en lo sucesivo a utilizarlo para obtener y mandar datos o señales de control para otros proyectos.

Existen en el mercado, grabadores muy sofisticados de EPROMs capaces de grabar una gran variedad de tipos de EPROM, proyectos no precisamente simples y que pueden llegar a costar algunos cientos de dolares.

Si las necesidades no demandan un grabador como el anterior, ya que es común que se utilice repetidadamente un mismo tipo de EPROM, entonces un grabador como el presentado en esta tesis (o una modificación de él, de manera que grabe el tipo de EPROM deseada), resultará ser una opción real, más económica, de interensante auto-construcción, y que simplemente: hará el trabajo !

BIBLIOGRAFIA

- The New Peter Norton Programmer's Reference Guide to the IBM PC & PS/2

Peter Norton
Richard Wilton

Microsoft Press

- Turbo C
User's Guide & Reference Guide

Borland International, Inc.

- C Power User's Guide

Herbert Schildt

Osborne/ Mc Graw-Hill

- Performance Programming Under MS-DOS

Michael J. Young

SYBEX

- Sistemas Digitales
Principios y Aplicaciones

Ronald J. Tocci

Prentice Hall