TESIS DE GRADO PREVIA A LA OBTENCION

DEL TITULO DE INGENIEROS EN

ELECTRONICA Y TELECOMUNICACIONES

DEPURADOR DE PROGRAMAS PARA EL MICROPROCESADOR i8086

FERNANDO ORTEGA LOPEZ

IVAN ORDOÑEZ REINOSO

ESCUELA POLITECNICA NACIONAL

1989

C O N T E N I D O

I. INTRODUCCION.-

1.1. Justificación del tema.-

El tema desarrollado por nosotros como tesis responde a una necesidad de nuestra Facultad (y tal vez, de otras facultades e inclusive otras instituciones educativas en el país), que es la de contar con un depurador para los microprocesadores de la serie del i8086 amistoso al usuario y de fácil utilización; dicho microprocesador es de uso muy extendido, pues constituye el cerebro de las computadoras de la serie de IBM PC y compatibles, que son las computadoras personales más vendidas en nuestro país, y para las cuales se desarrolla la gran mayoría de nuevos paquetes de software; además, son las máquinas con las cuales contamos en la Facultad, y en éllas se desarrolla algo de software en lenguaje ensamblador. Creemos que el paquete que presentamos en esta tesis favorecerá el desarrollo de este tipo de software, pues es una muy útil herramienta de depuración; además, podrá ser utilizado como base para el laboratorio de Microcomputadoras II, para el cual no se contaba hasta el momento con un depurador apropiado (en Microcomputadoras I se tiene el AVSIM Z80, para el microprocesador Z80).

Podemos comparar nuestro paquete con otros depuradores existentes en el mercado: DEBUG, parte integrante del DOS, y Codeview, proporcionado con el Macro Assembler 5.0 de Microsoft:

El primero es un programa interactivo pero no orientado a pantalla, es decir, no tiene actualización automática de código ni de zonas de memoria, y solo presenta la información explícitamente solicitada por el usuario, a excepción del contenido de los registros, que se presenta cada vez que se ejecuta una porción del código del programa que está siendo depurado. Por otro lado, utiliza la pantalla normal de video para presentar sus resultados, por lo que interfiere con el programa del usuario si éste trabaja con el video.

El segundo es un depurador de uso general para lenguajes de Microsoft, que tiene actualización automática de código, pero no de zonas de memoria; tiene varias ventajas de depuración, como la posibilidad de colocar puntos de ruptura en el código y otras, pero es de uso algo complejo, y requiere de memorización de comandos.

Nuestro paquete, al que hemos denominado 'PRO', es especifico para programas en lenguaje de máquina, pero

presenta muchas ventajas, además de la facilidad de manejo, como la posibilidad de manipular memoria, código y archivos en forma sencilla, y además tiene muchas ventajas de depuración, como puntos de ruptura visibles en el código y accesibles a través de una tabla, ejecución hasta el cumplimiento de una condición determinada, posibilidad de no ingreso a subrutinas y varias otras.

Por todo ésto creemos que el tema está bastante bien justificado, y esperamos que se constituya en una herramienta útil para el desarrollo de software en lenguaje ensamblador.

1.2. Planteamiento de objetivos.-

El objetivo de esta tesis es la elaboración de un depurador para programas escritos en lenguaje de máquina del microprocesador 8086 de INTEL, basado en el sistema operativo DOS. El programa debe ser amistoso al usuario, es decir, de fácil utilización y de rápida familiarización para nuevos usuarios, a fin de que sea una herramienta didáctica y profesional útil.

II. FUNDAMENTOS SOBRE EL DOS Y EL BIOS.-

2.1 Fundamentos del DOS.-

El DOS (Disk Operating System) es un sistema operativo desarrollado por la casa Microsoft para IBM, y se creó a propósito de la salida al mercado de la IBM PC original. La versión 1.0 del DOS era simplemente un CP/M (sistema operativo muy popular hasta entonces, para máquinas basadas en Z80 u 8080) traducido al lenguaje del microprocesador de aquella máquina, el 8088. A partir de entonces se liberaron varias actualizaciones del paquete, pero solo se hicieron cambios realmente importantes desde la versión 2.10, y a partir de la 3.00 se incluyó un nuevo conjunto de servicios manejadores de disco. Es a ésta versión y a las posteriores a las que nos referiremos de aquí en adelante.

El DOS consiste en cuatro componentes:

a) El registro de inicialización: Comienza en la pista 0, sector 1, lado 0 de todo diskette que esté en formato DOS, y se coloca allí para producir un mensaje de error si se intenta cargar el sistema operativo desde un diskette que no lo contiene. En discos duros, este registro reside en

el primer sector de la partición del DOS. Todos los medios soportados por DOS tienen un sector para registro de inicialización.

b) La interfaz con el ROM BIOS: El archivo IBMBIO.COM desempeña este papel, proveyendo una interfaz de bajo nivel con las rutinas manejadoras de dispositivos del ROM BIOS.

c) El archivo de programa del DOS: Este archivo es el IBMDOS.COM. Provee una interfaz de alto nivel para programas de usuario. Consiste en un grupo de rutinas de manejo de archivos, manejo de disco, y una variedad de funciones fácilmente accesibles por programas de usuario. Cuando un programa invoca a una de estas rutinas, estas aceptan información de alto nivel en contenidos de registros o de bloques de memoria. Para operar dispositivos, las funciones traducen los requerimientos en una o más llamadas a la interfaz de bajo nivel IBMBIO.COM.

d) El procesador de comandos, COMMAND.COM: Consiste de las siguientes partes:

- Una porción residente, que se instala en memoria inmediatamente a continuación del IBMDOS.COM y su área de datos. Contiene los servicios de las funciones de terminación y de manejo de errores, rutinas para cargar la parte transitoria del COMMAND.COM, y la rutina para cargar y ejecutar comandos externos, es decir archivos con extensiones .COM o .EXE.

- Una porción de inicialización, que sigue a la porción residente y toma el control durante la inicialización. Esta porción contiene la rutina que procesa al AUTOEXEC.BAT, y determina la dirección de segmento en que se pueden cargar programas. Esta porción es destruida en cuanto COMMAND.COM carga un programa tras la inicialización, pues ya no es necesaria, y la memoria que utilizaba es ocupada.

- Una porción transitoria, que se carga en el extremo más alto de la memoria. Este es el procesador de comandos en sí, que procesa todos los comandos externos y los archivos de extensión .BAT. Además, produce la señal de espera de comandos del DOS (como A>), lee el comando del teclado (o del archivo .BAT),

y lo ejecuta. Para comandos externos, construye una linea de comandos y llama a la función EXEC para cargar y transferir control al programa.

El sistema operativo se inicializa cada vez que se realiza un reset o que se enciende la máquina. Primero, el ROM BIOS toma el control, y busca el registro de inicialización en la unidad A. De no encontrarlo, busca la partición activa en el disco duro. Una vez encontrado dicho registro, es leido a memoria y se le pasa el control. Este busca entonces en el directorio raíz a fin de asegurarse de que los dos primeros archivos son el IBMBIO.COM y el IBMDOS.COM. Entonces, se carga en memoria el primero de estos archivos y se le transfiere el control.

De inmediato, IBMBIO.COM carga IBMDOS.COM, determina el estado de la máquina, inicializa los dispositivos periféricos, carga los manejadores de dispositivos (device drivers), crea la parte baja de la tabla de vectores de interrupción, relocaliza IBMDOS.COM más abajo en memoria e invoca al primer byte del DOS.

Entonces, el DOS inicializa sus tablas internas de trabajo y los vectores de interrupción desde el 0FH hasta el 3FH, y crea un prefijo de segmento de programa (Program Segment Prefix, de aquí en adelante referido como PSP).

Finalmente, IBMBIO.COM utiliza la función EXEC para cargar y ejecutar el procesador de comandos de alto nivel, normalmente el COMMAND.COM. Este procesador contiene, además, un manejador de errores críticos (accesible mediante la interrupción 24H), que da al usuario las opciones de abortar, reintentar o ignorar cuando se produce una falla. El usuario puede utilizar un manejador propio de errores interceptando la interrupción 24H (mediante la función 25H del DOS, Set Vector).

2.2 Funciones del DOS.-

El DOS proporciona al usuario un conjunto de funciones y procedimientos de alto nivel, accesibles mediante la interrupción 21H, que permiten manejar dispositivos, memoria, subprocesos y otros. Se ejecuta cada función proporcionando su número en el registro AH, los parámetros requeridos en otros registros, según cada

función, e invocando a la interrupción. Algunas funciones de manejo de archivos requieren de lo que se denomina un 'ASCIIZ', que es una tira de caracteres terminados con un 0, que suele representar al identificador de un archivo; por ejemplo, la siguiente línea de un programa en Assembly es un ASCIIZ:

```
nombre      db    'c:\dos\keybsp.com',0
```

Contamos con documentación de funciones numeradas desde la 0 hasta la 62H, y en el apéndice D exponemos brevemente las más importantes utilizadas para la elaboración del programa objeto de la presente tesis.

2.3 Funciones del BIOS.-

El ROM BIOS proporciona servicios de bajo nivel de control de dispositivos tales como el teclado, el video, y los controladores de disco. Normalmente, el programador puede prescindir de estos servicios y utilizar únicamente las funciones del DOS antes expuestas, pero el uso directo del BIOS es aconsejable en ciertos casos, cuando se desea tener programas muy rápidos y eficientes. Estos servicios

III. DESARROLLO DEL SOFTWARE.-

3.1 Definición del problema.-

Los productos disponibles en nuestro medio para depuración de programas escritos en lenguaje Assembler del 8086 tienen inconvenientes que fueron ya señalados en la introducción. Son las dificultades que esos depuradores presentan las que proporcionaron las metas para el programa que hemos desarrollado. El depurador debe ser de fácil uso, de rápida familiarización, y debe mostrar en la pantalla los elementos del microcomputador que interesan a quien escribe programas en lenguaje de bajo nivel: registros, banderas, memoria y código, siempre actualizados y en forma simultánea.

El objetivo inicial fue hacer un programa que realice las funciones básicas de depuración: desensamblar código, presentar registros, banderas y memoria, ensamblar instrucciones, correr el programa en prueba en varias modalidades (como paso a paso o ejecución hasta una dirección o ejecución total), colocar puntos de ruptura, etc... Para facilidad del usuario decidimos que el control sea en base a menúes y que los resultados se presenten en

áreas (ventanas) de la pantalla destinada cada una a un elemento específico.

Pero este objetivo inicial fue complementado al empezar la creación del depurador por las consideraciones que se exponen a continuación:

La etapa de desarrollo de un programa de cualquier tipo se caracteriza por los continuos cambios, arreglos, aumentos y eliminaciones que el programa sufre. En el caso de un programa interactivo con uso intensivo de la pantalla este problema es más agudo por la relativa complejidad que implica el establecimiento de una comunicación lógica y sencilla con el usuario del programa, y que lleve de una manera obvia a la realización de una función. Muchas veces quien empieza a usar una aplicación no conoce exactamente la forma de hacer cierto trabajo y allí el programa debe indicar el camino.

Por los motivos expuestos podemos decir que el programador emplea gran cantidad de su tiempo y esfuerzo a tareas que esencialmente son repetitivas: reorganizar el flujo de su aplicación, mejorar los interfaces que el programa presenta al usuario, cambiar la presentación de

su programa, etc.. Pronto se hizo evidente que existían dos posibilidades para el desarrollo de nuestro Tema de Tesis:

La primera, más directa, era escribir un flujo adecuado para el depurador, con todo lo que ésto acarrea: principalmente una gran cantidad de tiempo destinada a satisfacer cualquier modificación que la calidad del programa hiciera necesaria.

La segunda, y por la que finalmente optamos, consistía en sistematizar el mecanismo de un programa manejado en base a menúes (un controlador de flujo) y en crear un medio más fácil de alterar la presentación del depurador (un generador interactivo de ventanas). Esta opción implicaba, sin embargo, una etapa previa en las dos direcciones indicadas para luego volver a la idea central del depurador. Las ventajas de este método son la reducción del esfuerzo destinado a las modificaciones, por profundas que éstas sean, y, sobre todo, que la etapa previa da un resultado utilizable por cualquier aplicación interactiva y orientada a pantalla.

Una vez terminadas estas dos herramientas vino la etapa de planificar el flujo del depurador y definir con claridad que funciones debe realizar el programa. Esto no significa, desde luego, que no surgieron modificaciones posteriores.

Luego vino la creación de las rutinas que hacen las funciones planeadas y su inmediato enlace a un controlador de flujo. La presentación externa de la función se hizo sencilla y de calidad utilizando el generador interactivo de ventanas.

Estas herramientas, sumadas al uso de la programación modular, permitieron conseguir el objetivo de flexibilizar la extensa etapa de creación y modificación del depurador.

Esta descripción de la forma en que realizamos la Tesis indica cuál fue nuestra definición del problema y el método que seguimos. Las partes descritas arriba serán desarrolladas en los próximos puntos.

3.2 Desarrollo del programa principal.-

El núcleo del depurador es el controlador de flujo. Es el encargado de mantener información sobre todas las opciones del programa. Invoca a las rutinas en el momento adecuado y recibe de éllas el control cuando han terminado.

La idea del controlador de flujo nace de un breve análisis de qué es lo que hace un programa interactivo manejado por medio de menúes. Al empezar la ejecución se inicializan ciertas variables, se leen archivos, en fin, se realizan tareas que son la base para el trabajo posterior. Luego se presenta un menú en él cual podemos elegir alguna de sus opciones. Es posible que la opción nos lleve a un nuevo menú o al mismo del cual partimos, hay muchas posibilidades, pero lo importante es que una vez que entramos en el primer menú el medio que se presenta al usuario es el mismo y será el mismo esté en el menú que esté.

Este medio és el que se puede sistematizar: Pasada la rutina inicial el programa se transforma en una especie de lazo en el que pocas variables diferencian un estado de

otro: El nivel o profundidad en que nos encontramos respecto del menú principal y la posición de una opción dentro del árbol formado por opciones y subopciones.

Finalmente, cuando se quiere terminar el trabajo con el programa, se debe realizar alguna rutina final, como grabar algún archivo, dar mensajes, etc..

Cuando al usuario se le presenta un menú puede tomar alguna de estas acciones:

- Puede presionar una de las flechas y moverse a una opción contigua.

- Puede digitar la tecla ENTER entrando con ésto a cumplir rutina asociada a la opción sobre la que estaba ubicado y luego a recibir el tratamiento adecuado de acuerdo al tipo de función de que se trate. De ésto hablaremos, con mayor detalle, más adelante.

- Puede digitar la letra clave de una opción, con lo que entra directamente a esa opción sin necesidad de ubicarse sobre élla. Realiza la rutina asociada y

luego recibe el tratamiento adecuado al tipo de función.

- Cualquier otra alternativa no es tomada en cuenta.

Una vez ejecutada la rutina asociada a una opción existen varias alternativas (A ésto se referían las palabras "tipo de función" indicadas anteriormente). Estas alternativas son:

- Se cumple la rutina asociada y se regresa al mismo menú que contiene la opción. Estas rutinas podrían llamarse "normales".

- Se cumple la rutina asociada y se emerge un nivel en el árbol de opciones, es decir, se transfiere el control al menú "padre" del que contenía la opción. Estas funciones serán llamadas "inversas".

- Se ejecuta la rutina y se profundiza un nivel, mostrando un nuevo menú, "hijo" de la opción que fue digitada. Estas opciones son del tipo "rama".

En cualquiera de estos casos la rutina asociada puede no realizar ninguna función (rutina "dummy") más que la de

devolver el flujo al controlador. Esto facilita muchisimo la prueba de los módulos que se van desarrollando para el programa, pues, mientras éllos no existan, una rutina dummy tomará su lugar, y cuando estén listos podrán ser probados de inmediato. Además, puesto que el flujo ya está definido, la posición que la rutina pase a ocupar en el programa será la definitiva, sin importar la inclusión posterior de otros módulos.

Este sería un cuadro bastante rígido de un programa interactivo. En la implementación tomada en esta Tesis existen otras características para mayor facilidad de uso:

- El menú aparece en la penúltima línea de la pantalla (línea 23) y una explicación sobre la opción actual en la última línea (la 24).

- Una tecla (ESC), permite siempre salir al nivel superior de menúes. Si se está en el menú principal, ESC significa que se desea salir del programa. En todo caso se hace una pregunta para confirmar esa decisión.

- Una combinación (Ctrl-O o ^O), presenta una ventana sobre la opción en la que estamos ubicados actualmente con las subopciones que ésta posee, si existen. En el momento que aparece esa ventana de subopciones, las flechas hacia arriba y hacia abajo son aceptadas para moverse en ese menú. La explicación que aparece en la línea 24 es la asociada a la subopción en la que estamos actualmente. Si se presiona ENTER sobre una subopción se ingresa a la opción que la contiene y el cursor queda ubicado sobre la subopción seleccionada, listo para que una nueva presión de la tecla ENTER la mande a ejecutar. El mismo ^O desactiva la ventana de subopciones si no se la necesita.

- Una combinación (^P), presenta una ayuda acerca de la manera de manejar los menúes.

- La combinación ^K permite ingresar a las opciones de macros que el controlador de flujo provee: grabación, prueba, corrida y eliminación de macros. Esta opción solo puede ser invocada desde el menú principal, por razones que en su momento se expondrán.

- Una última característica, muy útil, es la llamada "rutina de escape", que es la posibilidad, a criterio del usuario del controlador de flujo, de tener una opción del menú que reciba como datos todas las teclas que se hayan digitado y que no sean opciones válidas del menú.

Desarrollo del controlador de flujo.

En el apéndice A consta el diagrama de bloques del controlador. Lo primero que anotaremos es que el flujo y el tamaño del mismo son constantes, independientes de la magnitud de la aplicación que va a ser controlada, y que la explicación es también general. Los detalles del programa, ya en profundidad, son comentados en los listados. Aquí solo indicaremos las funciones que los distintos bloques realizan, no la manera cómo lo hacen.

En este programa se usó una técnica soportada por el Macro Assembler 5.0 que es la de usar macros de instrucciones. Un macro, en este contexto, es un conjunto de nemotécnicos que se define una vez y luego puede ser

llamado con solo escribir su nombre, como si fueran nuevos nemotécnicos del lenguaje. En la definición de un macro se puede indicar parámetros, que son reemplazados al momento del ensamblado.

Por ejemplo, vamos a hacer un macro llamado PRUEBA que pone un valor X en el registro AL y X+1 en el registro BH. La definición del macro sería la siguiente:

```
prueba    macro     x
          mov       al,x
          mov       bh,x
          inc       bh
          endm
```

Las palabras MACRO y ENDM son las que determinan que PRUEBA es un macro. X es un parámetro que puede ser puesto por el programador, y puede ser un valor inmediato, una localidad de memoria o un registro. Dada la forma en que ha sido usado, X debe ser un byte (8 bits). Invocaciones válidas a este macro serían:

```
prueba    5
prueba    cl
prueba    valor1     ;siendo VALOR1 un byte
prueba    byte ptr [si]
```

Invocaciones inválidas serían:

```
prueba    1234          ;mayor que 0ffh
prueba    dx            ;registro de 16 bits
prueba    word ptr [si]       ;16 bits
```

Es importante el hecho de que un macro se define en el tiempo de ensamblado, no en el de ejecución. Esto significa que si hay N llamadas al macro, el correspondiente código aparecerá N veces, solo con las alteraciones producidas por los parámetros. En los puntos siguientes hablaremos de los macros como funciones y, para abreviar la notación, presentaremos los parámetros entre paréntesis aunque, recalquemos ésto, no es ésa la manera de llamar al macro.

Con esta explicación pasaremos a describir algunas variables y macros, su uso o su función:

- Nivel: variable que indica la profundidad a la que está el menú actual dentro del árbol de opciones y subopciones del programa. El menú principal tiene NIVEL = 0.

- Nivel1: variable, indica la profundidad de la opción actual. Es igual a NIVEL + 1.

- Nivel2: variable, indica la profundidad de la subopción actual. Es igual a NIVEL1 + 1.

- X: variable, indica la posición de la opción actual, aquella sobre la que nos encontramos ubicados. La primera opción de un menú tiene X = 1.

- Y: variable. Indica la posición de la subopción actual. Si la opción no tiene subopciones, o si la ventana de subopciones no está habilitada o si el cursor de subopciones no está sobre ninguna de éllas, Y vale 0. En otro caso Y vale 1 para la primera subopción y sucesivamente para las siguientes.

- Buffer: variable de 8 bytes. Es una secuencia de números que describe la manera cómo se llegó al menú actual. Cada número se va almacenando en medio byte (un "nibble"), en orden desde el menos significativo. El menú principal, por ejemplo, tiene su nibble menos significativo igual a 1 y el resto son ceros. Si allí se eligió la opción 6, el buffer tendría este valor en su siguiente nibble y así sucesivamente.

- Buffer1: variable de 8 bytes similar a BUFFER, pero en la que se ha incluido la opción actual X en el nibble cuyo número está dado por NIVEL1. Los nibbles más significativos siguen llenándose con ceros.

- Buffer2: variable de 8 bytes similar a las anteriores, con la adición de que incluye la subopción actual Y en el nibble cuyo número está dado por NIVEL2. Nótese que si Y = 0, BUFFER1 y BUFFER2 son iguales.

- Tabbuf: la manera cómo el usuario indica al controlador de flujo la manera de correr su aplicación es creando una tabla llamada TABBUF. Esta tabla contiene los buffers que indican cómo llegar a cada opción del programa. Esos buffers se construyen igual que lo indicado anteriormente para la variable BUFFER.

- Tabinf: es una tabla creada también por el usuario en la que constan las direcciones de las explicaciones, menúes y rutinas asociadas a cada opción, así como el tipo de opción de la que se trata, codificada de esta forma: Normal = 1, Reversa

= 2 y Rama = 3. En total se destinan 7 bytes (3 words y 1 byte) para cada opción. Esta tabla debe guardar estrictamente el mismo orden de TABBUF para que el programa funcione como es debido.

- Nclave: variable. Indica la posición de BUFFER en la tabla TABBUF y, por consiguiente, la posición de la información asociada a la opción en la tabla TABINF.

- Nclave1 y Nclave2: variables. Indican lo mismo que NCLAVE para BUFFER1 y BUFFER2 respectivamente.

- Bmac: bandera que indica que se está corriendo o probando un macro de comandos del programa. Los macros de comandos serán discutidos más adelante en detalle.

- Bgmac: bandera para indicar que se está grabando un macro.

- Bver: bandera que indica que se está probando un macro.

- Bopc: bandera que sirve para saber si hay que mostrar o no la ventana de subopciones. ^O la pone en 1 y 0 alternativamente.

- Bufcla(var,buf): macro que busca la ubicación de un buffer cualquiera BUF en TABBUF. Entrega su resultado en una variable cualquiera VAR. Con esta función, por ejemplo, encontramos NCLAVE en base a BUFFER.

- Movbuf(buf1,buf2): macro que copia un buffer BUF2 en BUF1.

- Nunivel(buf,niv,val): macro que coloca el valor VAL (que debe ser menor que 16 para caber en un nibble) en el buffer BUF, en el nibble cuyo número lo da NIV. Sirve, por ejemplo, para construir BUFFER1: copiamos BUFFER y luego incluimos X en BUFFER1 con esta función.

- Explica(n): este macro presenta en la línea 24 la explicación asociada al N-ésimo buffer de TABBUF.

- Menuvent(n): macro que dibuja una ventana que

contiene el menú asociado al N-ésimo buffer de TABBUF.

- Menulinea(n): macro que pone en la línea 23 el menú asociado al N-ésimo buffer de TABBUF.

- Rutina(n): ordena la ejecución de la rutina asociada al N-ésimo buffer de TABBUF.

- Retorno: macro que devuelve el control del flujo una vez que la rutina del usuario ha terminado.

- Opción?: es quizá la parte más crítica del controlador. Recibe datos del teclado, determina cuál fue la opción (flechas, ENTER, ESC, ^O,-^P, ^K u otra) y da un valor de salida que indica esa opción. Como lo veremos más tarde, tiene un papel muy importante en el manejo de los macros de comandos.

- Actuar: permite saltar a la AL-ésima dirección de una tabla de direcciones. La rutina anterior y ésta forman el núcleo del controlador.

- Statusf: muestra en el extremo derecho de la línea 23 el status del controlador, que puede ser : Normal, Macros, Grabando macro, Probando macro o Corriendo macro.

- Statusv: muestra una secuencia de 6 caracteres dada por la aplicación en el extremo derecho de la línea 24. Este es el único acceso del programa controlado a la línea 24 mientras se están mostrando los menúes. Aquí se podría mostrar el status del programa o algún valor que se estime necesario presentar simultáneamente con el menú y las explicaciones.

- Válida?: Analiza si una tecla pulsada que no es una flecha, ni ENTER, ni ESC, ni ^O, ^P o ^K es, en cambio, una opción válida del menú actual. Si así ocurre al salir AL tiene el número de la opción y la bandera de cero (ZF) está en 1.

Existen muchas otras variables y macros pero son los descritos anteriormente los que realmente influyen en el comportamiento del programa, cuya descripción empezamos (ver diagrama de flujo A1 en el apéndice A).

Lo primero que se hace es, entre otras cosas, determinar la dirección de la memoria de video y limpiar la pantalla. Enseguida se llama a la rutina inicial del usuario para que el realice las tareas que sean necesarias antes de entrar al lazo principal del programa.

Luego se continúa mostrando una pantalla de ayuda para que el usuario que no está muy familiarizado con el programa sepa cómo manejar los menúes y las opciones especiales. Después se colocan valores adecuados a las variables y buffers. Por ejemplo, se pone NIVEL en 0, todas las banderas de macros en cero, la bandera de subopciones en 0. Se inicializa BUFFER con un 1 en el nibble menos significativo y el resto ceros, para que al inicio pasemos, como es obvio, al menú principal. Además, para que la opción actual al iniciarse el programa sea la primera del menú principal, se hace X=1 y se pone un 1 en el nibble menos significativo y en el siguiente en BUFFER1.

En este momento entramos al lazo principal. En el programa este punto está marcado como "a_nivel" porque cada vez que cambiamos de nivel debemos venir a este sitio. Acá debemos llegar ya con el valor de X definido

(esa será la nueva opción actual), y no haremos más que poner Y=0 para que el buffer de la subopción actual coincida forzosamente con el de la opción actual.

Luego pasamos a un punto llamado "b_subnivel" en el programa. El nombre se debe a que cualquier acción que no produzca un cambio de nivel, como digitar una flecha, pedir la pantalla de ayuda, o la ventana de subopciones, o correr una rutina de tipo "normal", etc., debe regresar a esta dirección. Acá ya deben llegar X y Y listos.

Comenzamos encontrando la posición de BUFFER en TABBUF (en la variable NCLAVE). Hacemos NIVEL1 = NIVEL + 1 y construimos BUFFER1 copiándolo de BUFFER y añadiéndole el valor X en el nibble dado por NIVEL1. Procedemos de manera similar para construir BUFFER2. Encontramos las posiciones de ambos buffers y las guardamos en NCLAVE1 y NCLAVE2. En este momento BUFFER2 está asociado a la subopción actual, la que debe ser explicada al usuario del programa. Si BUFFER1 = BUFFER2, entonces sus números clave serán iguales y se verá la explicación de la opción actual. En todo caso se realiza la función Explica(Nclave2).

Pasamos al sitio etiquetado como "e_ventana" en el listado. Aquí se lee el estado de la bandera BOPC para ver si se debe mostrar o no la ventana de subopciones, si la opción actual las tiene. Esto se hace con la función Menuvent(Nclave1). No se muestra una ventana si las subopciones no existen. Antes de mostrar la ventana se preserva las 23 líneas superiores de la pantalla, pues le pertenecen a la aplicación.

De nuevo estamos en un lugar de paso obligado ("b_s5" en el programa) en el cual se presenta el menú en la línea 23. Este menú es el asociado a BUFFER, por lo cual se llama a la función Menulinea(Nclave). Luego se muestra el status del programa por medio de Statusf y Statusv.

En al dirección etiquetada como "c_opción" se acepta desde el teclado la elección del usuario. Con el valor que Opción? entrega vamos a la función Actuar para saltar a la rutina respectiva. Hablaremos brevemente sobre cada posibilidad.

Si el usuario digita "flecha a la izquierda" o "flecha a la derecha" hay que aumentar o disminuir X, según al caso. Si se llega a uno de los extremos del menú

hay que pasar al otro directamente. Hace Y = 0 siempre. No es permitido el valor X = 0. Luego se salta a b_subnivel.

Si se ha digitado "flecha arriba" o "flecha abajo" se debe ver primero si BOPC vale 1 (con lo que es posible que haya una ventana de subopciones). Si ésto no ocurre, se regresa a pedir desde teclado una nueva opción. Además se debe chequear si la opción actual tiene o no subopciones. Solo si se han pasado estos dos controles se entra a una rutina que aumenta o disminuye Y. Igual que en el caso de las otras dos flechas, si se llega a uno de los extremos se debe saltar al otro. Si es permitido el valor Y = 0, en cuyo caso los buffers de la opción y la subopción son iguales. Al acabar se regresa a b_subnivel.

Si se digita ^O se cambia el estado de la bandera BOPC,de 1 a 0 y viceversa. Esto hace que en la parte correspondiente del flujo se muestre la ventana de subopciones de la opción actual. Puesto el valor de la bandera, se salta a b_subnivel.

Si se digita ^P se presenta una pantalla de ayuda sobre la manera de usar el programa. Luego se espera que

se pulse alguna tecla para restaurar la pantalla del programa y saltar finalmente a la dirección b_subnivel.

Existen dos formas de ingresar a una rutina. La primera es pulsar la tecla ENTER. En este caso empezamos por incrementar NIVEL e incluir X en BUFFER (Esto hace BUFFER = BUFFER1). Si Y era distinto de 0 antes de entrar, ese valor pasa a X en el nuevo nivel. Si Y era 0, pero la opción era de tipo "rama", el valor de X será 1. Luego de colocar estos valores se salta al mismo punto al que se llega directamente si no se cumplió ninguna de las condiciones anteriores. Esta dirección tiene la etiqueta "c_002" en el programa. Pocas líneas después hablaremos de lo que ocurre en esta dirección, pues también forma parte de la otra forma de ingresar a una rutina.

Esta segunda forma es digitar una tecla distinta de las flechas o ESC o ENTER o ^K, ^P o ^O. El código ASCII de la opción digitada es analizado para ver si se trata de una de las opciones del menú por la rutina Válida?. Si la opción no fue correcta se regresa para recibir otra entrada del teclado. Pero si fue válida el número de opción regresa en AL y ese valor es ingresado en BUFFER (previamente se incrementa NIVEL en 1). Nótese que en este

caso BUFFER puede no ser igual a BUFFER1. Para uso posterior se pone en uno una bandera que indica el paso por este camino y no por el de la digitación de la tecla ENTER. En este punto está la dirección "c_002" que fue mencionada hace poco.

En la dirección c_002 se calcula la posición del nuevo BUFFER formado por medio de la función Bufcla(Nclave,Buffer). Obtenido NCLAVE se ordena la ejecución de la rutina asociada a este número clave, usando la función Rutina(Nclave).

La rutina se corre y al terminar debe devolver el control por medio del macro Retorno. Aquí todavía seguimos dentro del macro Rutina, que por último nos entrega en AL el tipo de opción que se eligió: 1 para rutinas normales, 2 para rutinas inversas y 3 para rutinas rama. Con el valor que está en AL se corre el macro Actuar para cumplir los pasos respectivos al tipo de opción.

- En la rutinas normales se recuperan las variables BUFFER y NIVEL de los respaldos que habían sido elaborados antes de entrar en la parte en que el

usuario hace su opción. Con los valores previos se regresa a b_subnivel.

- En las opciones inversas se procede del mismo modo que para las rutinas normales, pero en lugar de ir a b_subnivel se salta a "d_supernivel", donde se emerge al nivel superior. Esta parte del código será explicada más adelante cuando hablemos sobre la opción ESC.

- En las opciones rama se ve primero si la forma de entrar fue por medio de la tecla ENTER o por una opción válida del menú. Si esto último fue lo ocurrido (lo que se puede saber pues se puso una bandera que indicaba el paso por la parte que atiende a las opciones válidas del menú) se hace X=1 para el nuevo menú. Recordemos que si lo digitado fue ENTER ya en la parte inicial de la rutina respectiva se preparó el valor de X. Una vez puesto el valor de X se salta a la dirección a_nivel.

Otra de las opciones disponibles es la tecla ESC. Es atendida en la dirección d_supernivel, ya mencionada antes. El nombre se debe a que se encarga se subir un

nivel en los menúes. Esta rutina primero busca en BUFFER el número de opción que el "padre" tenía en el menú superior. Luego se introduce un 0 en el buffer en vez del último nibble diferente de 0, que representa al menú del que se desea salir, se disminuye NIVEL en 1 y se hace X igual al número de opción del padre en el nuevo menú. El objetivo de ésto es que al emerger quedemos ubicados sobre la opción que originó al menú del que salimos.

Al ejecutarse esta parte del código puede darse el caso de que al disminuir NIVEL en 1 se llegue a obtener NIVEL = 0FFH. Esto significa que estábamos en NIVEL = 0 y en este caso ESC o una rutina inversa indica que se desea abandonar el programa. En este caso se pide una confirmación de esa decisión ("Quieres terminar?"). Si la respuesta es negativa se ponen condiciones similares a las de iniciación y se salta a a_nivel. Pero si la respuesta es afirmativa el programa llama a la rutina final de la aplicación, para que el usuario realice cualquier función necesaria antes de abandonar el programa. De esta subrutina se regresa por un simple RETN y de inmediato se termina el programa.

Este es el controlador de flujo. Una opción más nos resta por explicar, pero por ser más compleja que las otras, requiere una explicación aparte.

Macros de comandos en el controlador de flujo. La idea de los macros en este programa aparece porque hay una rutina central en el programa que se encarga de recibir las opciones hechas por el usuario. Esta rutina puede complicarse un poco para que no solo sea capaz de leer el teclado sino que, al detectarse la presencia de la bandera de macros, pueda leer también de algún buffer presente en memoria y ejecutar de corrido. Cuando la bandera de grabación esté "puesta", la rutina debe guardar las opciones en un buffer. Cuando la bandera de prueba de macros esté en 1, debería leer sus opciones del buffer y esperar cierta tecla para continuar u otra en especial para abandonar la prueba.

En el controlador de flujo se ingresa a los macros con ^K, digitado en el menú principal. Por qué solo desde ese menú? La razón es la forma sencilla que se ha querido dar a estos macros, como una simple secuencia de opciones guardadas en un área de memoria. Así, esta secuencia solo tiene sentido si su grabación y su ejecución empiezan

desde el mismo punto y en las mismas condiciones. Por éso se elige como sitio de partida el menú principal y, como se puede notar si se analiza el listado, se hace X = 1 (la primera opción), Y = 0, y se pone la bandera de opciones en 0.

El área destinada a macros tiene el formato dado en la tabla 1. El primer espacio es un índice en el que a cada macro se le destina 8 bytes, el primero para almacenar el número de opciones que tiene el macro y los 7 restantes para el nombre del macro. Ocho macros es el máximo que se puede tener simultáneamente en memoria aunque, como se explicará luego, la aplicación puede hacer un uso mucho mayor en extensión y en calidad de los macros. Después del índice está el área de datos de los macros, en la que se asignan 64 words a cada uno. Cada word (2 bytes) contendrá una opción hecha por el usuario en respuesta a un menú presentado por el controlador de flujo.

Tabla 1: Area de macros del controlador de flujo

Areamac		
Offset	Parámetro	
0 1 8 9 .. 56 57	# de opciones macro 1 Nombre macro 1 # de opciones macro 2 Nombre macro 2 ... # de opciones macro 8 Nombre macro 8	Indice
64 192 .. 960	64 words de macro 1 64 words de macro 2 ... 64 words de macro 8	Datos
1088 1089	# de macro actual Contador de posición en macro actual y byte de interfaz.	

En este momento conviene aclarar que los macros del controlador solo almacenan las opciones que son dirigidas a él, y no los valores, respuestas, etc. ingresados a rutinas de la aplicación. Sobre éllas no tiene ningún conocimiento el controlador. Es la propia aplicación la que deberá manejar esas respuestas.

Al programa que es controlado le es entregado un grupo de variables declaradas PUBLIC en el controlador, que le ayudarán en la propia implementación que cada programador quiera hacer. Los datos pasados son: AREAMAC, que es la etiqueta colocada al inicio del área de macros,

y BMAC, BGMAC y BVER, que son las banderas de macros. Aparte de la información dada hasta aquí, hay dos bytes más de suma importancia para el uso de macros del programa controlado. Al final del área de datos de los macros está un byte en que el controlador pone el número de macro que se está grabando, probando o corriendo. Este dato puede ser utilizado por la rutina de macros de la aplicación para saber el número de macro actual. El siguiente byte es el único medio de comunicación entre las rutinas internas y externas de macros. El controlador usa este byte para llevar la cuenta de la posición en al que estamos dentro del macro, ya sea en grabación o ejecución. Pero si la rutina externa de macros pone este byte en 0FFH, le indica a la rutina interna que la aplicación considera imperativo terminar el macro en ese momento. Es por éso que este byte es llamado byte de interfaz. Cada vez que se digita ^K este byte se pone en cero, listo para empezar a grabar o correr macros. No se debe alterar ningún otro valor (como las banderas) para ordenar al controlador la terminación del macro.

Otro detalle interesante es este: Hay rutinas, especialmente de tipo rama, que antes de permitir el paso al menú del nivel inferior, hacen alguna pregunta o algún

chequeo crítico. Si no se pasa este chequeo, no se debería permitir la profundización. Pero hasta ahora hemos visto que el flujo es predefinido y no hemos visto forma alguna de alterarlo en caso necesario. Esto se puede hacer cambiando la variable TIPRUT declarada PUBLIC en el controlador. Esta variable tiene el tipo de la opción que estamos corriendo. Si alteramos este valor, el flujo se altera. Pero ésto tiene una implicación para los macros. Habíamos dicho que el conjunto de opciones que conforman un macro tiene sentido si las condiciones iniciales son las mismas. Aquí encontramos otra condición: un macro pierde su sentido cuando el flujo es alterado en medio de la ejecución. Para evitar problemas, la rutina de macros del controlador chequea siempre si TIPRUT vuelve con el mismo valor con el que ingresó a la rutina del programa controlado. Si ésto no ocurre se da un mensaje y se ordena la inmediata terminación de la grabación o ejecución del macro, porque el macro deja de ser confiable el momento en que puede haber ambigüedad en el flujo.

Lo anteriormente dicho explica las características de los macros desde el punto de vista del controlador. La aplicación externamente puede tener muchas variantes. La adoptada para el depurador es tener un área externa de

macros en la que se guardan las respuestas del usuario a requerimientos ajenos al controlador. Se destinan 256 bytes a cada macro. Tanto el área de macros interna como la externa son grabados en disco al final de la sesión de trabajo con el depurador, de manera que los macros que hayan estado definidos no se pierdan. Al inicio de cada sesión se busca el archivo de macros y se lo lee en las áreas correspondientes: interna y externa. Si el archivo no es encontrado, se crea uno.

Otras variantes pueden ser: tener varios archivos de macros (a manera de una libreria) que puedan ser leídos en cualquier momento. Esta opción fue probada en el depurador pero no resultó útil pues, ocurriendo en el proceso de depuración continuas terminaciones de programas, los archivos abiertos durante la depuración de estos programas se cerraban también continuamente y sin dar la posibilidad de salvar la información. Solo los archivos abiertos no en el nivel del programa depurado (Child Process, según la convención del DOS) sino en el nivel superior (Parent Process) se mantienen abiertos y son, por lo tanto, útiles. Más sobre ésto se hablará al tratar sobre el depurador. Una variante más para los macros puede ser

tener un archivo que contenga más macros, pero que los traiga y los salve en grupos de ocho.

Expliquemos las rutinas de macros internas del controlador. Cuando digitamos ^K en el menú principal aparece una pantalla en al que se explica con brevedad las características de los macros y se pide el nombre de un macro. Si el nombre digitado no existe se empieza la grabación del nuevo macro, siempre que aún exista espacio para otro nombre. Si el nombre es preexistente, se pregunta si se desea probar, correr o borrar el macro, y se hace lo que la respuesta del usuario indique.

Como ya se dijo antes, la grabación de un macro no es más que el almacenamiento de las opciones pulsadas en el buffer respectivo. Por supuesto se debe revisar continuamente si no se ha excedido el tamaño máximo del buffer o si el usuario ha digitado otra vez ^K para terminar el macro.

La prueba y la ejecución son muy parecidas excepto por el hecho de que al probar un macro se debe digitar la tecla ENTER para seguir viendo cómo trabaja el macro. Aquí la rutina que lee las opciones debe hacerlo de un buffer

previamente grabado. En el caso de la corrida de un macro ésta no termina hasta la producción de una condición de error (opción ambigua detectada o byte de interfaz en 0FFH) o el fin natural del macro. En la prueba de macros, también se puede hacerlo si en vez de ENTER se digita ^K.

La rutina de escape opcional. Esta caracteristica está contenida en los macros Opción? y Válida?. El macro Opción? envía cualquier tecla que posiblemente pertenezca a un menú (es decir que no sea una de aquellas con función predefinida) con el mismo código de salida. El macro Actuar hace entonces un salto a la rutina que maneja las posibles opciones válidas del menú. Aquí pasamos primero por el macro Válida? que busca en el formato del menú si al final de las opciones existe un byte con el valor 0FFH, que indica que existe la opción de escape. Si la tecla pulsada es parte del menú, sale con el número de la opción respectiva, y si no lo es, sale con ZF = 0 que indica error o con ZF = 1 y el número de la última opción si se había habilitado la rutina de escape. La utilidad de esta rutina es manejar todos los códigos ASCII, normales o extendidos, que no son parte de un menú ni funciones predefinidas del controlador. Por ejemplo, si lo que se está haciendo es un editor de texto, todas las teclas

normales serán manejadas por una misma rutina y solo los controles establecidos por el propio usuario son manejados por rutinas diferentes. En el programa de flujo se dio a la rutina de escape la responsabilidad de manejar las teclas de función del teclado. Todas las demás teclas que se dirijan a esta rutina, excepto "F", "f" o ENTER, son descartadas sin procesar. Y esas tres teclas despliegan en pantalla un cuadro del uso de las teclas de función. Se puso esa rutina de escape en todos los menúes.

Flujo del depurador.

El módulo que maneja (por medio del controlador de flujo) a todas las rutinas del depurador es PRO.ASM. Es este módulo el que debe contener las tablas indicadas anteriormente y que se definen en base a las labores que debe hacer el depurador.

En la tabla 2 presentamos el esquema de menúes y opciones del depurador. Es un esquema que puede servir de modelo a quien desee usar el controlador. La información que el diagrama entrega es la siguiente:

- El número que antecede al nombre de la opción es su posición dentro del menú.

- Luego está el nombre de la opción en el cual, escrita con mayúscula, está la letra con la que va a ser llamada la opción. Anotemos que no debe ser necesariamente una letra del nombre, ni siquiera una letra. Cualquier código ASCII no extendido puede ser usado como "tecla clave" para llamar a una opción.

- Lo siguiente que vemos es un número entre paréntesis, que representa el tipo de función, numerado de la forma que ya hemos indicado en páginas anteriores.

- Si la opción tiene subopciones (observe que ésto ocurre siempre en las rutinas de tipo 3), se escriben a continuación éstas, con la misma notación indicada pero recorridas algunos espacios a la derecha para diferenciarlas del nivel superior. El punto puesto a la izquierda indica que todas esas rutinas descienden de la opción bajo la cual están los puntos.

- Finalmente, se observa en algunas opciones, que siempre son las últimas de su respectivo menú, un

asterisco antecediendo al nombre. Este asterisco indica que se trata de la rutina de escape del menú.

Tabla 2: Esquema de opciones del depurador

```
1: Archivos (3)
     .          1: Nombre (1)
     .          2: Tamaño (1)
     .          3: Lee (2)
     .          4: Graba (2)
     .          5: *Funciones (1)

2: Información (1)

3: Depurador (3)
     .          1: Paso (1)
     .          2: Corre (1)
     .          3: eJecuta (1)
     .          4: Rupturas (3)
     .               .          1: Crea (1)
     .               .          2: Activa/Desactiva (1)
     .               .          3: Borra (1)
     .               .          4: Muestra (1)
     .               .          5: *Funciones (1)
     .          5: Ins (1)
     .          6: Ens (3)
     .               .          1: Ensambla (1)
     .               .          2: Busca (1)
     .               .          3: *Funciones (1)
     .          7: Mem (3)
     .               .          1: Llena (1)
     .               .          2: Busca (1)
     .               .          3: Copia (1)
     .               .          4: *Funciones (1)
     .          8: *Funciones (1)

4: Termina (2)

5: *Funciones (1)
```

Los buffers que aparecen en el listado son solo una descripción de cómo llegar a cada opción del programa

expresada en forma de números de opción. El menú principal tiene este buffer:

01h,00h,00h,00h,00h,00h,00h,00h

Y todos los demás buffers tendrán ese 1 ubicado en el mismo nibble, pues todos descienden del menú principal.

Por ejemplo el buffer correspondiente a Depurador se forma a partir del principal incluyendo un 3 en el segundo nibble:

31h,00h,00h,00h,00h,00h,00h,00h

Y el correspondiente a la opción Ins del menú del Depurador se forma poniendo un 5 en el tercer nibble:

31h,05h,00h,00h,00h,00h,00h,00h

De esta manera se construye la tabla TABBUF del depurador, como se la ve en el listado del archivo PRO.ASM. Esta tabla puede construirse en el orden que se quiera, por ejemplo podrían ir primero todas las opciones del menú principal, incluso puede hacerse en desorden. Pero la siguiente tabla, TABINF, se debe construir, como

ya se dijo antes, estrictamente en el mismo orden que TABBUF. TABINF es una colección de las direcciones (offsets dentro de los segmentos de código, para las rutinas, y datos, para los menúes y explicaciones) de las informaciones asociadas a cada opción. Además, consta el tipo de opción.

Por último, este es el formato de un menú:

- Primero, un string con las opciones tal como se quiere que sean vistas en la línea 23. Las opciones deben estar separadas al menos por un espacio, y el string no debería exceder los 74 bytes, pues los últimos 6 espacios de la línea se usan para mostrar el status del programa.

- Un byte de 0 para indicar el fin del menú.

- Una lista de las opciones válidas del menú, sea en hexadecimal o en forma de códigos ASCII encerrados entre comillas. Las opciones válidas no deben ser necesariamente letras, pero si una letra se usa se debe indicar el código de la letra mayúscula.

Cualquier código ASCII no extendido puede ser una opción válida.

- Finalmente está la posición para un byte de 0FFH si se quiere que la última opción sea de escape. Como una seguridad se puede poner un byte de 0 si no se necesita la rutina de escape para evitar que algún dato que esté después del menú adopte por azar el valor de 0FFH.

Como ejemplo, el menú principal del depurador es:

'Archivos Información Depurador Termina Funciones',0,
'AIDTF',0ffh

Con ésto hemos completado nuestra descripción del ambiente en que se desarrolló el depurador y pasamos a la descripción de la manera en que funciona.

3.3 Desarrollo de las funciones: módulos individuales.-

Todos los módulos están comentados en los listados que se incluyen en el apéndice A; sin embargo, a continuación exponemos las funciones que desempeña cada módulo, e incluimos diagramas de los flujos principales en dicho apéndice.

Módulo Flujo.Asm:

Es el que controla el flujo de este programa. Su trabajo fue ampliamente explicado en páginas previas.

Módulo Pro.Asm:

Este módulo es el encargado de invocar a todas las rutinas de depuración. En él se encuentran las tablas usadas por Flujo.Asm. El esquema de las opciones de este módulo fue definido anteriormente; de él hablaremos luego, como el módulo que integra a los demás.

Módulo Desens.Asm:

Este es el desensamblador, capaz de convertir cualquier código de máquina del 8086 en una tira alfanumérica que contenga su mnemotécnico equivalente. Contiene varios submódulos, siendo DesensDet el de más alto nivel; lo que hace es llamar a Desens, y luego reformatea la tira de caracteres que este último pasa a fin de responder a la necesidad del módulo Pro, que la requiere de cierta forma para poder colocarla en la ventana de código.

Desens es el que realmente genera la tira alfanumérica que representa el mnemotécnico, y adicionalmente proporciona la longitud en bytes del código de máquina desensamblado (para que el procedimiento invocante pueda localizar la siguiente instrucción).

Hay varios otros submódulos utilizados por Desens cuya función no se explica aquí. Para ver su detalle, referirse a los diagramas de flujo adjuntos en el apéndice A.

Módulo Assem.Asm:

Este es el módulo ensamblador, capaz de convertir cualquier mnemotécnico válido del 8086 en su código de máquina correspondiente, y además, proporcionar la longitud de este código en bytes. Su submódulo de más alto nivel es Assem, que invoca sucesivamente a los dos pasos de compilación.

Estos dos pasos son EnsPaso1 y EnsPaso2; el primero genera lo que denominamos un "pseudocódigo", es decir, una representación codificada del mnemotécnico que no es el código de máquina; simplemente, se reduce cada símbolo a un código numérico. El segundo es el que convierte el pseudocódigo al código de máquina verdadero.

El pseudocódigo es totalmente arbitrario, y se elaboró buscando simplemente una forma sencilla de interpretar una tira de caracteres respetando su sintaxis. A continuación incluimos la convención utilizada para cada uno de los diferentes símbolos que puede encontrarse en un mnemotécnico de 8086. La convención está a nivel de bits (ver tabla 3).

Tabla 3 (Definición del pseudocódigo)

(Las X son No Importa)

Instrucción:

0	0	0	X	X	J	U	L	Número

J = 1 si la instrucción es JMP o CALL
0 en otro caso.
U = 1 si la instrucción admite únicamente un parámetro (como por ejemplo NOT o DEC)
0 en otro caso.
L = 1 si la instrucción no admite parámetros (por ejemplo AAD o CBW)
0 en otro caso.
Número es el número de la instrucción, arbitrariamente predefinido.

Registro:

0	0	1	X	W	Num

W = 1 si se trata de un registro de 16 bits (no se incluye registros de segmento)
0 si se trata de un registro de 8 bits.
Num son 3 bits que representan el número del registro.

Registro de segmento:

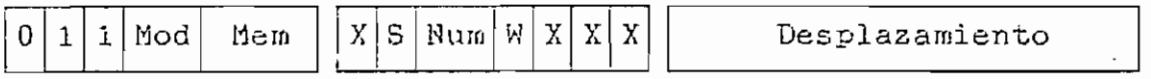
0	1	0	X	X	X	Num

Num son 2 bits que representan el número del registro de segmento.

sigue...

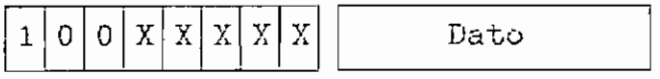
Tabla 3 ... Fin

Memoria:



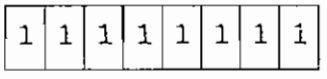
Mod son 2 bits que representan el modo de direccionamiento.
Mem son 3 bits que representan la forma de acceso a memoria.
S = 1 si hay un prefijo de redefinición de segmento
0 de lo contrario.
Num es el número del registro de segmento del prefijo.
W = 1 si el desplazamiento es de 16 bits (de haberlo)
0 si es de 8 bits.

Dato de 8 bits:



Dato de 16 bits:

Fin de pseudocódigo:



En todos los casos X = No Importa. Por ejemplo, la instrucción

```
MOV CX,ES:[SI+BP+1437],13D
```

tendrá el siguiente pseudocódigo en hexadecimal, haciendo 0 todos los No Importa:

00 88 29 72 48 37 14 A0 3D 01 FF

Los números de las instrucciones están listados en la siguiente tabla:

Tabla 4: Números de instrucciones

ADD	00	JNAE	72	MOV	88	RCR	D3
PUSH	06	JB	72	LEA	8D	AAM	D4
POP	07	JC	72	NOP	90	AAD	D5
OR	08	JNB	73	CBW	98	XLAT	D7
ADC	10	JAE	73	CWD	99	ESC	D8
SBB	18	JNC	73	CALL	9A	SHL	DC
AND	20	JZ	74	WAIT	9B	SAL	DC
DAA	27	JE	74	PUSHF	9C	SHR	DD
SUB	28	JNZ	75	POPF	9D	SAR	DF
DAS	2F	JNE	75	SAHF	9E	LOOPNE	E0
XOR	30	JNA	76	LAHF	9F	LOOPNZ	E0
AAA	37	JBE	76	MOVSB	A4	LOOPE	E1
CMP	38	JNBE	77	MOVSW	A5	LOOPZ	E1
NEAR	39	JA	77	CMPSB	A6	LOOP	E2
FAR	3A	JS	78	CMPSW	A7	JCXZ	E3
PTR	3B	JNS	79	STOSB	AA	IN	E4
BYTE	3C	JPE	7A	STOSW	AB	OUT	E6
WORD	3D	JP	7A	LODSB	AC	JMP	EA
DWORD	3E	JPO	7B	LODSW	AD	LOCK	F0
AAS	3F	JNP	7B	SCASB	AE	REPNE	F2
INC	40	JNGE	7C	SCASW	AF	REPNZ	F2
DEC	48	JL	7C	RET	C2	REP	F3
SHORT	50	JNL	7D	RETN	C2	REPE	F3
QWORD	51	JGE	7D	LES	C4	REPZ	F3
NOT	52	JNG	7E	LDS	C5	HLT	F4
NEG	53	JLE	7E	RETF	CA	CMC	F5
MUL	54	JNLE	7F	INT	CC	CLC	F8
IMUL	55	JG	7F	INTO	CE	STC	F9
DIV	56	DB	80	IRET	CF	CLI	FA
IDIV	57	DW	81	ROL	D0	STI	FB
JO	70	TEST	84	ROR	D1	CLD	FC
JNO	71	XCHG	86	RCL	D2	STD	FD

Debe notarse que, por conveniencia, se asignaron pseudocódigos iguales a los códigos de máquina siempre que fue posible, y cuando no, se tomaron números que se podían transformar fácilmente en los códigos verdaderos.

Los registros tienen los siguientes números binarios:

Tabla 5: Números de registros

Num	w = 1	w = 0
000	AX	AL
001	CX	CL
010	DX	DL
011	BX	BL
100	SP	AH
101	BP	CH
110	SI	DH
111	DI	BH

Los registros de segmento tienen los siguientes códigos binarios:

Tabla 6: Números de registros de segmento

Num	Reg
00	ES
01	CS
10	SS
11	DS

Los modos de direccionamiento y las formas de acceso a memoria son los siguientes en binario:

Tabla 7: Modos de direccionamiento y formas de acceso a memoria

Mem	Mod 00	01	10
000	SI+BX	SI+BX+Desp8	BX+SI+Desp16
001	DI+BX	DI+BX+Desp8	DI+BX+Desp16
010	SI+BP	SI+BP+Desp8	SI+BP+Desp16
011	DI+BP	DI+BP+Desp8	DI+BP+Desp16
100	SI	SI+Desp8	SI+Desp16
101	DI	DI+Desp8	DI+Desp16
110	Desp16	BP+Desp8	BP+Desp16
111	BX	BX+Desp8	BX+Desp16

Estos bits se generan en el pseudocódigo haciendo un análisis lógico de la expresión entre corchetes, a fin de determinar los símbolos existentes en esta expresión. Este análisis es realizado por el submódulo EncMem de la siguiente manera:

Tabla 9: Obtención del modo de direccionamiento

FEDCBA	Mod	Mem	Error
000001	00	100	0
000010	00	101	0
000100	00	111	0
001000	01	110	0
000101	00	000	0
000110	00	001	0
001001	00	010	0
001010	00	011	0
010001	01	100	0
010010	01	101	0
010100	01	111	0
011000	01	110	0
010101	01	000	0
010110	01	001	0
011001	01	010	0
011010	01	011	0
100001	10	100	0
100010	10	101	0
100100	10	111	0
101000	10	110	0
100101	10	000	0
100110	10	001	0
101001	10	010	0
101010	10	011	0
010000	00	110	0
100000	00	110	0
Otro	XX	XXX	1

De esta tabla de verdad, mediante reducción de expresiones booleanas a través de mapas de Karnaugh se obtuvo las fórmulas lógicas que constan en el listado del procedimiento que las evalúa. A manera de ejemplo analizaremos manualmente la expresión [bx+si+17]. Veremos que están presentes los registros

BX, SI y un desplazamiento de 8 bits, por lo que A = C = E = 1, mientras que B = D = F = 0. Por tanto, la entrada en la tabla es 010101, a la que corresponden Mod = 01 y Mem = 000.

Módulo Ejecutor.Asm:

Contiene un conjunto de procedimientos de control de flujo de subprogramas (a fin de posibilitar la depuración) y de manejo de archivos y de memoria. A continuación explicamos brevemente los más importantes:

El procedimiento TestTrap hace una prueba del comportamiento de la interrupción 1, que se debe producir cuando la Trap Flag (T) sea 1. Lo que debería ocurrir es que, inmediatamente despúes de la instrucción que hizo 1 a T, se ejecute una instrucción más del código y de inmediato se produzca la interrupción. Sin embargo, detectamos que en algunas computadoras esta interrupción se demora dos instrucciones, y no una, en ocurrir. Por éllo, es necesario detectar cómo se va a comportar la máquina antes de utilizar esta interrupción.

El procedimiento ExUniq ejecuta una instrucción del código del programa del usuario. Para éllo, carga los registros a los valores en que se quedaron tras la instrucción anterior, pone T en 1, y luego transfiere el control a dicha instrucción (si TestTrap detectó que la interrupción demora una instrucción de más, entonces se ejecuta un NOP antes de transferir el control). La interrupción 1 está interceptada, y devuelve el control al procedimiento.

Hemos detectado otro tipo de comportamiento anómalo en ciertas computadoras: algunas instrucciones involucradas con registros de segmento parecen ser ignoradas por la interrupción 1, así que cuando ExUniq se encuentra con una de estas instrucciones, transfiere el control a ExPaso, que es un procedimiento del que hablaremos más adelante.

El procedimiento ExHasta ejecuta el código del usuario desde una dirección inicial hasta otra de terminación. Para éllo, inserta un INT 3 en la

dirección hasta la que se quiere ejecutar, el cual devuelve el control.

El procedimiento ExPaso ejecuta desde una instrucción inicial del código del usuario hasta la instrucción consecutiva exclusive; es útil para, por ejemplo, ejecutar una subrutina sin tener que seguir su detalle. Opera calculando la longitud de la instrucción de inicio mediante el procedimiento Desens a fin de encontrar la dirección de la siguiente instrucción, para, con esta información, invocar al procedimiento ExHasta.

El procedimiento ExCond ejecuta el código del usuario hasta encontrar el cumplimiento de una condición determinada. Para hacer ésto alternativamente invoca a ExUniq y verifica la condición; cuando ésto ocurre, toma nuevamente el control.

El procedimiento ExTotal ejecuta el código del usuario hasta encontrar un punto de ruptura. Para éllo, lee de una tabla en que están estos puntos e inserta un INT 3 en la dirección de cada uno de

éllos, y luego transfiere el control. La interrupción está interceptada, y devuelve el control al procedimiento. Para definir la tabla hay tres procedimientos: AddBrkPt, que añade un punto de ruptura a la tabla; DelBrkPt, que borra un elemento de la tabla, y TogBrkPt, que activa o desactiva un punto de ruptura.

El procedimiento ExIns ejecuta una instrucción fuera del código del usuario, aceptando como parámetro de entrada su mnemotécnico. Para hacerlo, invoca a Assem, que genera el código de máquina, y luego pasa este código a ExUniq, al que invoca.

El procedimiento ReduceLevel invoca a la función EXEC del DOS a fin de ejecutar un subprograma, Child.Exe, que simplemente devuelve el control sin finalizar. De esta forma, conseguimos 'bajar de nivel', es decir, engañar al sistema operativo para que, si se llega a ejecutar la función EXIT no se retorne al DOS sino que se siga teniendo el control. La respuesta del programa a un EXIT es invocar nuevamente a ReduceLevel para estar preparado para un nuevo EXIT. Debe notarse que EXEC crea un PSP para el

subproceso a ejecutarse, y es este PSP el que asignamos al programa cargado mediante LoadExe.

El procedimiento SaveFile graba una porción de la memoria del usuario como un archivo. Para determinar el inicio del bloque que debe grabar examina el nombre del archivo, y da tratamientos diferentes si se trata de un .EXE, un .COM u otro cualquiera. Puesto que los archivos .EXE requieren de una cabecera con información normalmente generada por el linker, y no hemos desarrollado una rutina para crear esta cabecera, SaveFile no puede crear dicho tipo de archivos; sin embargo, es posible modificarlos, es decir, traerlos de disco, hacerles ciertos cambios y grabarlos después, siempre y cuando no contengan información de relocalización (es decir instrucciones tales como MOV AX,@DATA), pues ésta sería alterada. Nuestra recomendación es no grabar archivos de extensión .EXE con PRO. En cambio, sí es posible crear archivos nuevos .COM o de otros tipos, siempre que se desee que sean una simple imagen de lo que se tiene en memoria.

El procedimiento LoadExe carga en memoria un archivo desde disco, dado un nombre colocado en el PSP que fue creado por ReduceLevel; la lectura se hace de forma que ese PSP pasa a corresponder al archivo cargado, si este archivo es un programa .EXE o .COM. Este procedimiento hace la carga utilizando la función EXEC en el caso de archivos .EXE, a fin de permitir al DOS usar la información de relocalización proporcionada en la cabecera del archivo, y utilizando las funciones normales de archivos (3DH, 3EH, 3FH) para los demás casos.

Módulo Memory.Asm:

Contiene tres procedimientos de trabajo sobre zonas de memoria. El primero, Fill, permite llenar una zona con un valor de byte determinado. El segundo, Search, busca una tira de bytes en un área determinada. Y el último, Copy, copia un bloque de un lugar a otro en la memoria.

3.4 Integración de módulos.-

En este apartado daremos una breve descripción del programa, viéndolo como una entidad única e integrada por los módulos antes descritos (ver diagrama de bloques A.5).

El programa contiene una rutina inicial (rinic), un lazo principal y una rutina final (rfinal). En la rutina inicial, en primer lugar, se verifica la presencia de los archivos de ventanas requeridos para operar. Estos archivos fueron creados utilizando "vent.exe", que es un programa que permite definir recuadros en la pantalla con información fija y con campos que aceptan información dinámicamente del usuario o de un programa; llamamos "ventanas" a estos recuadros, y son grabados como registros en un archivo. Siguiendo la convención necesaria, nuestro programa invoca a las ventanas que necesita (mediante un nombre previamente asignado a cada una) para proporcionar información al usuario. Los archivos de ventanas y de macros son abiertos; de no existir un archivo de macros, uno es creado.

Si la búsqueda de los archivos de ventanas es exitosa, entonces se procede a reducir la memoria que se

tiene disponible, a fin de dejar espacio para el programa del usuario (que va a depurarse). Esto se debe hacer porque el DOS invoca a los programas mediante la función EXEC, que otorga toda la memoria libre al subproceso. Así pues, una vez reducida ésta a los requerimientos de nuestro programa (1300H párrafos, es decir, 77824 bytes), el resto puede ser utilizado por el programa del usuario.

De inmediato se instala en memoria un servicio para la interrupcion 2FH, que es la llamada 'multiplex'; esta interrupción recibe como parámetro un número de función en AH, y están libres todos los números desde 80H hasta F0H. Nosotros reconocemos como nuestra la función 80H, y el servicio responde a esta función restaurando el stack (que fue alterado por efecto de la interrupción) y devolviendo el control al programa, sin ejecutar jamás un IRET. Una vez instalado el servicio, invocamos a la función EXEC para correr el programa Child.Exe, que lo único que hace es invocar a la función 80H de la interrupción multiplex. Puesto que ya está instalado el servicio a esta interrupción, y no se llega a ejecutar un EXIT, el DOS "cree" que seguimos ejecutando un subproceso nuestro. El efecto de ésto es poder transferir temporalmente el control al programa del usuario sin temor de que éste

ejecute un EXIT, pues si lo hace, el DOS no toma el control, sino que lo retorna al proceso que invocó al último EXEC, es decir, nuestro propio programa, que entonces volverá a ejecutar Child.Exe para nuevamente bajar de nivel.

Una vez reducido el nivel del proceso al nivel del proceso hijo, se examina el PSP del proceso invocante (el PSP original proporcionado por el DOS a nuestro programa) para encontrar si el usuario ha pasado parámetros. De ser así, se carga el nombre del archivo cuyo nombre fue proporcionado; si se encuentran más parámetros, éstos son pasados al PSP del proceso hijo (que se creó cuando se corrió Child.Exe). La carga del archivo se hace de forma que el PSP del proceso hijo se convierte en el PSP del programa contenido en dicho archivo. Con éllo nos evitamos el tener que construir un PSP, ya que EXEC puede hacerlo por nosotros.

Entonces se pasa al ciclo principal, en que se interactúa con el usuario, aceptando los comandos que éste proporcione. Cada vez que se ejecuta cualquier opción se llama a uno de los módulos (por ejemplo, si el usuario desea ejecutar una línea de su código, se invocará ExUniq

o Expaso tras haber creado los parámetros de entrada que estas rutinas necesitan) y luego se actualiza la pantalla mediante la rutina pdebug, que pone en el video cada una de las ventanas con los últimos cambios. Así, cada vez que se actualiza la ventana de código es necesario invocar varias veces a DesensDet a fin de obtener los mnemotécnicos desensamblados correspondientes a la parte del código del usuario que se está analizando. Como se dijo antes, si uno de los procesos ExUniq, ExPaso, ExCond, ExHasta o ExTotal, en que se pasa temporalmente el control al código del usuario, este código ejecuta un EXIT, el control se toma e inmediatamente se vuelve a bajar de nivel. Hemos encontrado en ésto una ventaja inexistente en programas como Debug o Codeview, y es la posibilidad de volver a ejecutar un proceso de usuario que terminó sin necesidad de traerlo nuevamente de disco.

Debe notarse que antes de bajar de nivel se abren los archivos de ventanas y de macros requeridos por el programa con el bit de herencia (Inheritance bit) en 0, de suerte que los archivos son pasados abiertos al subproceso; así, desde el nivel bajo puede seguirse llamando a las ventanas y ejecutando macros. Por otro lado, los archivos que el proceso del usuario

IV. EJEMPLOS TUTORIALES: EXPERIMENTACION Y RESULTADOS

4.1 Una sesión típica de laboratorio

Esta herramienta está concebida para trabajar en el Laboratorio de Microcomputadores II. En general el trabajo empezará con el planteamiento de un problema o necesidad. La persona que desarrolla la solución deberá primero elaborar un diagrama de bloques sencillo, indicando las tareas que deben hacerse. Luego escribirá el código adecuado para cada bloque. Luego el programa debe ensamblarse con el utilitario MASM y hacerse ejecutable con el LINK. Casi siempre existirán errores de sintaxis que el MASM indicará. Estos deben corregirse, inclusive los "Warning" o avisos, pues son potencialmente errores que impedirán funcionar al programa. El LINK suele dar un "Warning" cuando no se ha definido el área de stack, pero éso no impide que un programa funcione, excepto en aplicaciones muy extensas como Pro.Exe. Esta parte del trabajo debe ser hecha sin otra ayuda que los mensajes del MASM, hasta salir sin errores.

El trabajo de Pro empieza entonces, cuando, a pesar de no haber errores de sintaxis, generalmente un programa no funciona. Pro ayudará a encontrar los errores, ahora sí

de lógica o flujo o incluso de cálculo, que subsisten en el código.

Las correcciones consiguientes que se hagan en el programa harán que cada vez la corrida avance más por buen camino, quizá hasta encontrar otro error. El usuario puede verificar el avance en la depuración con Pro, viendo si los errores anteriores han sido superados y seguir hasta que todo el programa esté bien.

El método a seguirse, en resumen, es el siguiente:

1.- Descripción del problema.

2.- Diagrama de bloques y solución inicial.

3.- Corrida inicial: resultados y síntomas.

4.- Depuración con Pro y correcciones sobre la marcha.

5.- Corrección de errores.

6.- Repetición de los pasos hasta funcionamiento completo.

7.- Solución definitiva.

4.2 Ejemplos de depuración de programas

En este punto vamos a presentar dos programas y llegar hasta su funcionamiento completo. Partimos de un punto en que ya se hayan superado los errores de ensamblado, pues en esa etapa solo interviene el usuario en la corrección.

Programa 1:

1.- Problema: Presentar en la pantalla un cronómetro que muestre una cuenta en segundos, hasta 60, con un error máximo aceptable de 1 segundo en total. Para sincronizar relojes, dar un BEEP al comenzar y al terminar el programa.

2.- Diagrama de bloques y solución inicial: constan en el apéndice A. El nombre del programa es Crono.EXE. El diagrama es el A.6.

3.- Corrida inicial: Aparece una larga serie de símbolos no esperados en la pantalla, una gran cantidad de BEEP´s y demora demasiado tiempo. Se "cuelga" al intentar detenerlo con Ctrl-Brk por lo que damos un Ctrl-Alt-Del.

4.- Depuración con Pro: Corremos Pro con la siguiente orden:

pro crono.exe ↵

Ingresamos al depurador y empezamos por buscar nuestra área de datos en la ventana principal de memoria para tener una mejor visión. Sin embargo buscamos los bytes que habíamos inicializado al escribir el programa (un 0dh y 24h seguidos) y no los encontramos. Para ver si están en alguna parte, entramos a la opción de búsqueda en memoria y elegimos dar la secuencia en hexadecimal. Ingresamos luego:

0d ↵ 24 ↵ ↵

El depurador encuentra la secuencia en el offset 16F del segmento de datos, sin embargo en el listado del programa vemos que la ubicación de los datos se supone en un área más baja.

Entonces notamos que lo que falta es inicializar el valor de DS con el segmento que el DOS le asigne. Esto

explica que al intentar el programa mostrar la secuencia de caracteres que forma el número, no encuentre el terminador (24H o "$"). Además el programa estaba trabajando sobre un área un tanto delicada: Vemos que CS es superior a DS con 10H. Quien conozca más en profundidad al DOS sabrá que el DS está apuntando al PSP (Program Segment Prefix) y que las alteraciones que se haga en esa zona son peligrosas. A ésto se debe que el programa se colgara luego del Ctrl-Brk.

5.- Corrección: Salimos de Pro y con un editor de texto añadimos las siguientes líneas justo al inicio del programa:

```
mov     ax,@data
mov     ds,ax
```

para inicializar el DS. Compilamos nuevamente el programa y lo corremos.

6.- Continuación de la depuración: Ahora el programa corre hasta una terminación, pero en la pantalla aparecen los gráficos correspondientes a los valores más bajos del

código ASCII, además suenan BEEP's continuamente. Para ver qué sucede volvemos a Pro:

pro crono.exe ↵

Lo primero que observamos es que en el código ya constan las dos líneas de inicialización de DS. Por medio de F1 ingresamos al editor de valores en pantalla y presionamos TAB hasta llegar a la ventana de memoria. Alli, con la flecha → pasamos al campo de segmento y ponemos el valor con el que DS se inicializa. De inmediato vemos ya un área reconocible (por los bytes 0DH y 24H que teniamos inicializados). Con ESC salimos del editor y empezamos a correr el programa paso a paso. Ingresamos a la subrutina que debe convertir el valor hexadecimal del contador en dos bytes en que el primero tenga las decenas y el segundo las unidades. Seguimos paso a paso esta subrutina y encontramos que la transformación ha sido correcta, pero que los valores almacenados no son ASCII. Por ejemplo, si el número que debía mostrarse era el 21, las decenas tienen el valor 02 y las unidades 01. Para convertir en ASCII se debe sumar 30H (el "0" en ASCII) luego de la transformación. Por ésto era que se

veían los gráficos de los valores más bajos del código ASCII y se escuchaban los BEEP's (07 en ASCII).

Salimos de Pro y volvemos al editor para poner dos líneas:

```
add     decena,30h
add     unidad,30h
```

Compilamos y ordenamos la corrida del programa. Se ve que los problemas de numeración han sido superados, pero el cronómetro es demasiado rápido. Toma aproximadamente 15 segundos en contar hasta 60. Entramos a Pro para calibrar la velocidad. Ubicamos fácilmente en el código el lazo de retardo y vemos que se pone el valor de AL en 2. Para cuadruplicar el retardo, entramos en la opción Ens, opción Ensambla. Ponemos como dirección de ensamblado la de la instrucción que queremos alterar (23) y presionamos ENTER. Escribimos:

```
mov     al,8 ↵ ↵
```

Vemos que el cambio consta ya en el código. Salimos otra vez al depurador y ordenamos la ejecución en tiempo

real (J). Tomamos el tiempo y vemos que el cronómetro está calibrado de acuerdo a la tolerancia indicada en el problema.

7.- Solución: Salimos de Pro, y con el editor corregimos el valor dado a AL y compilamos otra vez. Ahora la corrida es completamente satisfactoria. El listado definitivo consta en el apéndice A. El listado del programa no depurado es Crono1.Asm y el final es Crono.Asm. El programa Crono.Exe entregado es la versión depurada.

Programa 2:

1.- Problema: Pedir al usuario que dé el nombre de un archivo, recibirlo y mostrar sus 128 primeros bytes.

Nota: La solución inicial es una versión sin errores de sintaxis de un programa que empezaba a escribir un compañero nuestro.

2.- Diagrama de bloques y solución inicial: Constan en el apéndice A. El programa se llama Test.EXE. El diagrama es el A.7.

3.- Corrida inicial: Da el mensaje inicial pero luego aparecen más caracteres. Escribimos un nombre de archivo válido, pero el programa termina de inmediato sin mostrar los primeros bytes del archivo.

4.- Depuración con Pro: Corremos Pro con este comando:

pro test.exe ↵

Hacemos, tal como en el ejemplo anterior, que la ventana principal de memoria muestre el segmento con el que DS es inicializado. Luego con PageDown y PageUp buscamos el área en que se encuentra la pregunta al usuario. Si hace falta ajustar el área para ver mejor el mensaje, se lo puede hacer escribiendo un offset adecuado para la ventana.

En este ejemplo vamos a aplicar las capacidades de Pro para hacer muchas correcciones sobre la marcha, las mismas que serán anotadas para luego corregirlas de una sola vez con un editor.

Empezamos la ejecución paso a paso. Llegamos hasta antes de ejecutar la primera INT 21 que es la que muestra la pregunta y, al analizar las condiciones de entrada a la interrupción, encontramos que el buffer no termina como se indica en el manual de funciones del DOS, ésto es, con el signo "$". Esto es lo que provocó que aparecieran más caracteres que los esperados. Además, aunque ésto no es tan crítico, se ve que no existen caracteres para provocar el cambio de línea (CR y LF).

Entonces decidimos poner los caracteres CR, LF y "$" en vez de las últimas tres letras de la pregunta. Con ésto, aunque la pregunta no aparecerá completa, al menos terminará en el sitio correcto, pasando a la siguiente línea. No ponemos los caracteres después de la pregunta pues podemos estar alterando otros valores posteriores. Para hacer ésto entramos al editor con F1 y con TAB o SHIFT TAB llegamos a la opción de cambios hexadecimales a la memoria. Con las flechas vamos a la posición que queremos cambiar y escribimos:

0d ↵ 0a ↵ 24 ↵

Salimos con ESC y hacemos otro paso para que se ejecute la interrupción. Luego de ésto podemos ver la pregunta en la pantalla del usuario presionando F4. Para regresar al depurador, presionamos ESC.

Ahora pasamos a la recepción del nombre del archivo, revisando siempre que las condiciones de entrada a la interrupción sean correctas. Ejecutamos la INT 21 y se nos presenta la pantalla del usuario. Escribimos el nombre de un archivo que sí exista. Esta vez todo funciona bien, por lo que intentamos abrir el archivo cuyo nombre hemos recibido. Sin embargo al ejecutar paso a paso vemos que DX, que debía hacerse apuntar al nombre del archivo, se hace apuntar al inicio del buffer en que recibimos el nombre, sin tomar en cuenta que en ese buffer, antes del nombre, existen dos bytes usados por el DOS en la función de recepción de un string. Para continuar sin tener que salir de Pro, presionamos F1 y con las flechas nos ubicamos sobre DX. Allí alteramos su valor de 000E a 0010. Ahora sí DX apunta al nombre. Pero existe otro problema. En el nombre la función de lectura de teclado del DOS ha añadido al final un CR que hará que nuestro archivo no sea encontrado, provocando la terminación inmediata. Como seguimos en el editor, nos movemos con TAB o SHIFT TAB

hasta la opción que permite cambios de memorias en hexadecimal. Nos ubicamos sobre el 0D al final del nombre y escribimos 0 sobre él. Con ésto, además, aseguramos que el nombre recibido se ha convertido en un ASCIIZ. Salimos al depurador y seguimos paso a paso hasta ejecutar la INT 21. Vemos que la apertura es exitosa pues CF=0. Además vemos que en AX hemos recibido el File Handle.

Siguiendo paso a paso, llegamos a la lectura de los 128 bytes, que se hace correctamente. Esto lo podemos confirmar entrando con F1 al editor y llegando a alguna de las opciones de memoria con TAB o SHIFT TAB. Allí, con PageUp o PageDown buscamos el área en la que leímos y debemos encontrar allí los datos.

Seguimos con la ejecución paso a paso. El archivo se cierra ahora sin problemas, y entramos al lazo que muestra los 128 bytes. Podemos seguir paso a paso algunas iteraciones, pero si ya queremos ver terminada la etapa, entramos a la opción Corre, en la que indicamos la dirección en la que queremos parar: 6A. El programa se ejecuta hasta ese punto. Por un instante se ve la pantalla que el programa depurado elaboró. Si queremos ver la pantalla en detalle, pasamos con F4 a la página de

usuario. Con ESC regresamos al depurador. Finalmente se observa la terminación normal del programa al seguir la ejecución. En una sola corrida bajo Pro, el programa ha mostrado muchos errores, al parecer, todos.

5.- Corrección: Salimos de Pro y con un editor escribimos luego de la pregunta los bytes 0DH, 0AH y 24H (o "$").

Hacemos que, al abrir el archivo, DX apunte dos bytes arriba del inicio del buffer, es decir al nombre.

Finalmente escribimos una pequeña rutina que pone un cero en vez del CR al final del nombre del archivo, con lo que, de paso, convertimos al nombre en un ASCIIZ.

6.- Continuación de la depuración: El programa es compilado y enviado a ejecución. La prueba es por completo satisfactoria.

7.- Solución definitiva: El listado con el programa tal como quedó finalmente se presenta en el apéndice A. Test1.Asm es el listado corregido de Test.Asm. Test.Exe es la versión del programa que funciona.

tres salidas independientes de las cuales una es dedicada a realizar una cuenta descendente desde un valor programable que, al llegar a cero, envia una señal al controlador de DMA, el cual lee todas las localidades de memoria y las vuelve a escribir con el mismo valor (refrescamiento). El tiempo que se ocupa en el refrescamiento de memoria es "muerto" para el computador.

Por otro lado, los fabricantes de microcomputadores suelen diseñar sus equipos suponiendo las condiciones más adversas posibles. Así, por ejemplo, supondrán que la memoria va a tener la volatilidad más alta de las indicadas por el fabricante en sus hojas de datos y, por lo tanto, habrá que refrescarla muy a menudo. Por éso el tiempo entre una y otra operación de -refrescamiento de memoria será relativamente corto. Sin embargo casi nunca un chip llega a los limites dados por el fabricante.

En base a esta idea, podemos suponer que nada ocurrirá en el computador si aumentamos ligeramente el período de refrescamiento. Esto se refleja en un mayor porcentaje del tiempo dedicado a procesamiento y, en consecuencia, en un aumento de velocidad. De igual manera,

si reducimos ese periodo el computador se volverá más lento.

El limite estará en el punto en que los refrescamientos sean tan separados que la memoria en efecto se descargue, provocando que los chequeos de paridad que periódicamente hace el computador por medio de un Generador/Verificador de Paridad 74ALS280 fallen. Este provoca un NMI y un mensaje de "Parity Error" aparece, caso en el cual no hay más remedio que hacer un RESET.

El tiempo entre refrescamientos es programable, y se lo fija dando el valor deseado (en microsegundos) al registro correspondiente del Contador.

Hemos probado esta técnica con Pro, empleando como muestra dos programas. El primero es el Crono.Exe, desarrollado en la segunda parte de este capitulo, y el segundo fue un simple lazo en el que internamente pusimos dos lazos.

Primero entramos en Pro y, basándonos en el ejemplo que da Roemmele, ensamblamos un archivo Qfresh.Com desde 100h, con estas instrucciones (números en hexadecimal):

```
        mov     al,74     ;Seleccionamos Timer 1
        out     43,al     ;para contar hacia abajo.
bajo:   mov     al,15     ;Valor inicial para el
        out     41,al     ;byte menos significativo
alto:   mov     al,0      ;y para el byte más
        out     41,al     ;significativo.
fin:    mov     ah,4c
        int     21
```

El valor dado a AL en "bajo" y en "alto" define las partes menos y más significativa del valor con el que se inicializa el contador. Después de algunas pruebas hemos visto que aparentemente esos valores son los que tiene normalmente una de las dos computadoras en que desarrollamos esta Tesis, pues no producen cambios de velocidad en las pruebas. Equivalen a un refrescamiento cada 21 microsegundos.

Grabamos este archivo en disco con el nombre Qfresh.Com y el tamaño de 10 (hexadecimal).

Definimos un macro que cargue Qfresh.Com y nos deje listos para ensamblar una instrucción en la dirección que arriba marcamos como "bajo". Con ésto podemos cambiar

Luego hicimos otra prueba. Creamos un programa con el nombre de Veloz.Com. Su listado es éste:

```
lazo:   mov     al,FF       ;inicio del lazo
        mov     cx,FFFF
lazo1:  loop    lazo1       ;1er lazo interior
        mov     cx,FFFF
lazo2:  loop    lazo2       ;2do lazo interior
        dec     al
        jnz     lazo
        mov     ah,2        ;BEEP
        mov     dl,7
        int     21
fin:    mov     ah,4C
        int     21
```

Las etiquetas que se ven aquí obviamente no se escriben en el ensamblador sino que sirven para saber a dónde debe ser algún salto. Los valores están en hexadecimal.

Salvamos el programa con nombre Veloz.Com y tamaño 1A. Creamos un macro similar a CRONO con nombre VELOZ. Probamos el tiempo que toma este macro sin alteraciones y es de unos 62 segundos. Hacemos las mismas pruebas que para CRONO y anotamos los resultados.

Tabla 10: Resultados del experimento

Periodo de refrescamiento(us)	Tiempo de ejecución(s) Crono.Exe	Veloz.Exe
02 (02h)	107	113
03 (03h)	79	85
04 (04h)	73	77
08 (08h)	65	68
15 (0fh)	61	64
21 (15h)..normal.	.....60.....	62
48 (30h)	58	61
96 (60h)	57	60
511 (1ffh)	57	60

Un gráfico muestra los resultados en el apéndice A.

Las conclusiones son claras y demuestran la verdad de las hipótesis, aunque las medidas no han sido exactas, pues no era el objetivo del ejemplo. Se puede ver que del valor inicial obtenido no se obtiene una ganancia mayor al 5 % . Sin embargo las diferencias son dramáticas cuando reducimos el período de refrescamiento. La relación inversa que existe entre el período de refrescamiento y el tiempo total de ejecución es evidente.

El gráfico muestra que, en estado normal, los valores del período de refrescamiento nos ubican en la parte plana de la curva. Por éllo podemos ganar muy poco en velocidad, pero perder mucho. Este resultado hace evidente el criterio que tuvo el fabricante para elegir el período.

V. CONCLUSIONES Y RECOMENDACIONES

El programa PRO.EXE es una herramienta de depuración bastante útil, pues presenta una gran cantidad de información simultáneamente en la pantalla, y actualiza esta información automáticamente. Se compara muy favorablemente con otros programas similares existentes en el mercado, y creemos que es el primer paquete de este tipo desarrollado en el Ecuador; existen buenas posibilidades de comercializalo, con un consecuente beneficio para la Facultad.

PRO.EXE es un programa muy probado y útil como herramienta de depuración y de investigación; el apartado de ejemplos de depuración (4.2) de esta Tesis tiene dos ejemplos muy claros que muestran su utilidad para corregir programas escritos en lenguaje Assembly del 8086.

Sin embargo, hasta el momento de la culminación de esta Tesis de Grado, no podemos considerarlo como un paquete acabado, pues son innumerables las mejoras que tenemos pensado procurarle, además de corregir algunos pequeños defectos que hemos detectado y que puedan seguirse descubriendo; para éllo, contamos con el gran aporte que sería la utilización del paquete en Laboratorio de Microcomputadoras II, pues el uso intensivo del

programa por parte de los estudiantes producirá sin duda muchas sugerencias.

En la etapa de desarrollo de la Tesis se suscitaron algunas situaciones y problemas cuya solución son el aporte de este trabajo. Hablemos de ellas brevemente.

Se encontraron diferencias importantes en la forma como el DOS maneja los archivos ejecutables de extensiones .EXE y .COM: el primero tiene una cabecera de longitud variable con información de ítemes de relocalización, valores iniciales de registros y otros; el segundo es una imagen de cómo queda el programa en la memoria al ser cargado, y no tiene ninguna cabecera; por ello, fue necesario el desarrollo de rutinas diferentes de manipulación de ambos tipos de archivos. Si bien se logró evitar todos los problemas concernientes a la carga, no fue posible regenerar los ítemes de relocalización en la cabecera de los programas .EXE; ello será adicionado al programa en el futuro.

Se detectó un comportamiento anómalo no documentado en algunos microprocesadores de la familia del 8086 en cuanto al número de instrucciones que se retarda la

interrupción 1 en producirse una vez que la bandera de trampa T se ha hecho 1 (debería ser una sola instrucción, pero a veces son más). Esto hizo necesario el desarrollo de una rutina de reconocimiento de la forma cómo el microprocesador se comporta, a fin de tomar medidas que eviten los problemas que la anomalía produciría. Debemos señalar que en ésto, PRO es superior a otros depuradores (como DEBUG o Codeview), que eventualmente fallan en algunas computadoras.

Puesto que la ejecución de un programa de usuario por porciones implica la intervención de su código (restaurándolo cada vez que el depurador recupera el control), no fue posible permitir al usuario seguir el flujo de programas en ROM (como el BIOS), puesto que no pueden ser alterados; sin embargo, otros depuradores (como DEBUG) sí son capaces de hacer ésto, e ignoramos la forma en que proceden (como es obvio, ésto no está documentado). Sugerimos este asunto como tema de investigación, pues un resultado concreto en este aspecto implicaría una mejora importante a nuestro paquete.

PRO.EXE tiene una habilidad que ningún otro depurador que conozcamos posee, y es el poder conservar la pantalla

del usuario a tal punto que la presencia del depurador es totalmente transparente a cualquier aplicación de usuario, incluso aunque maneje directamente la Video RAM (VRAM). Conseguimos ésto manteniendo aquella pantalla siempre preservada, y restaurándola cada vez que el programa del usuario o una porción de éste (aunque se trate de una sola instrucción) toma el control, y sin utilizar las funciones del DOS ni del BIOS, sino directamente la memoria de video, que tiene un formato muy sencillo ya explicado anteriormente. Esto evita el problema que se presenta cuando el programa depurado no trabaja con las funciones del DOS y del BIOS sino sobre la VRAM. Pero tiene su inconveniente cuando el programa puesto a prueba cambia el modo de video, pues este parámetro no ha sido considerado. Esta podría ser una de las mejoras a incluirse, pero no es imperativa pues será bastante raro que programas no comerciales usen estas características.

El módulo ensamblador representó un paso muy interesante en la elaboración de PRO, pues implicó el desarrollo de un analizador de sintaxis de mnemotécnicos. La dificultad que implicaba el generar directamente código de máquina a partir de un mnemotécnico nos indujo a trabajar en dos etapas. En la primera chequeamos la

validez de los elementos sintácticos del mnemotécnico y lo reducimos a una secuencia de bytes que representa a dichos elementos en forma condensada, y llamamos a esta secuencia "pseudocódigo". En la segunda, generamos el código de máquina a partir del pseudocódigo. Creemos que este módulo puede ser tomado como ejemplo para futuras aplicaciones que requieran de análisis de sintaxis de instrucciones y de generación de código.

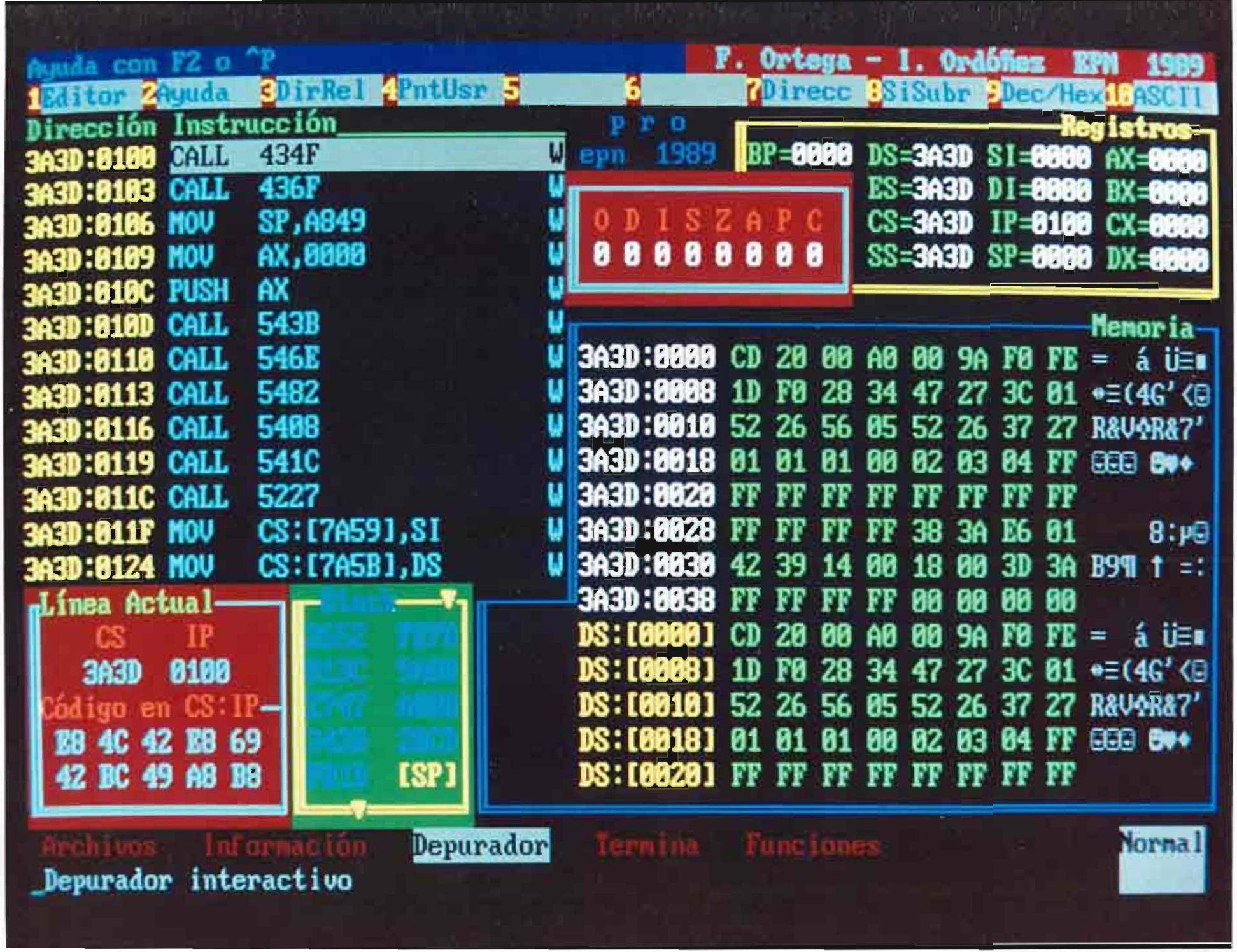
El desarrollo de PRO no se hizo tratándolo como un solo programa, sino como un conjunto de módulos que se integraron posteriormente; esta técnica de programación modular demostró ser eficiente y cómoda, pues inclusive permite a varias personas el trabajar en un solo proyecto. Por éllo, recomendamos este estilo de programación a cualquier persona o grupo de personas que planeen desarrollar un paquete relativamente extenso.

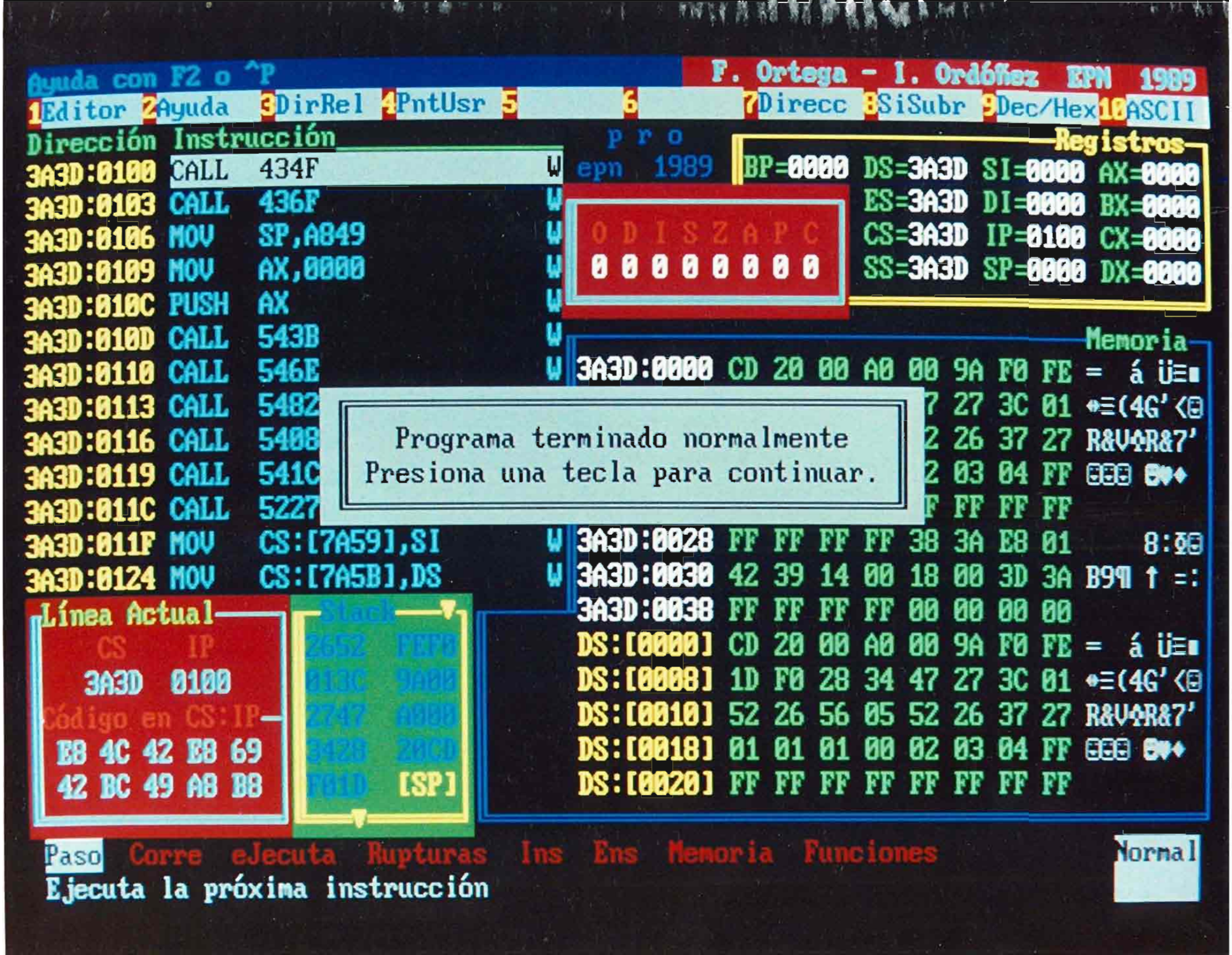
Complementando este criterio, hemos de decir que los módulos de esta Tesis están ampliamente documentados y pueden ser utilizados por otras aplicaciones, lo que significa ahorro de tiempo y esfuerzo. En general, todo desarrollo de Software debiera ser orientado, no estrictamente al programa que se desarrolla actualmente,

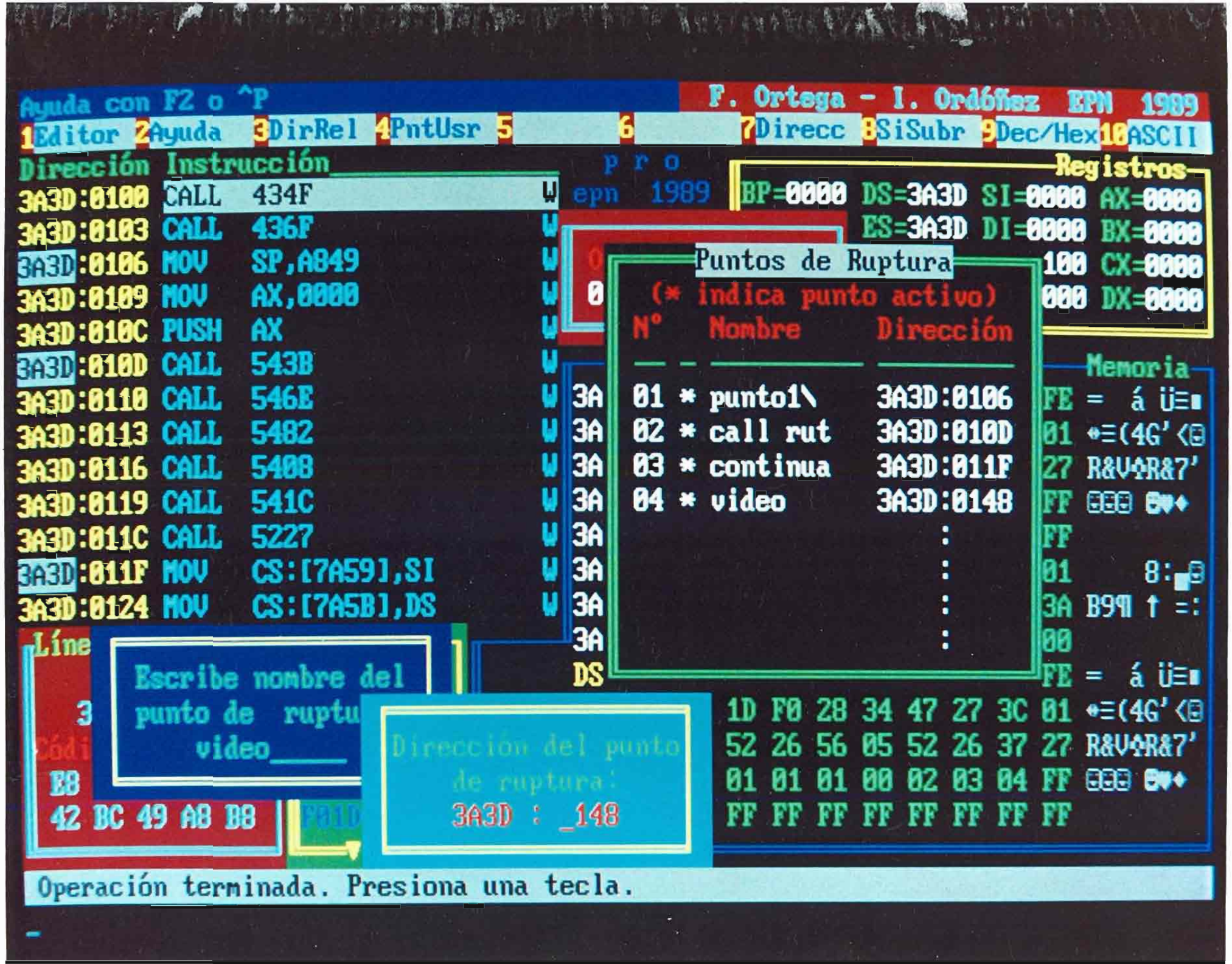
sino a su integración a cualquier otro. Por otro lado, de quienes dirigen el desarrollo de Tesis o trabajos que involucren a la programación, se requiere que destinen un gran esfuerzo a la planificación de éllos, para lograr un avance coordinado y no disperso, sin duplicación de trabajo, y, sobre todo, con conocimiento de causa y objetivo.

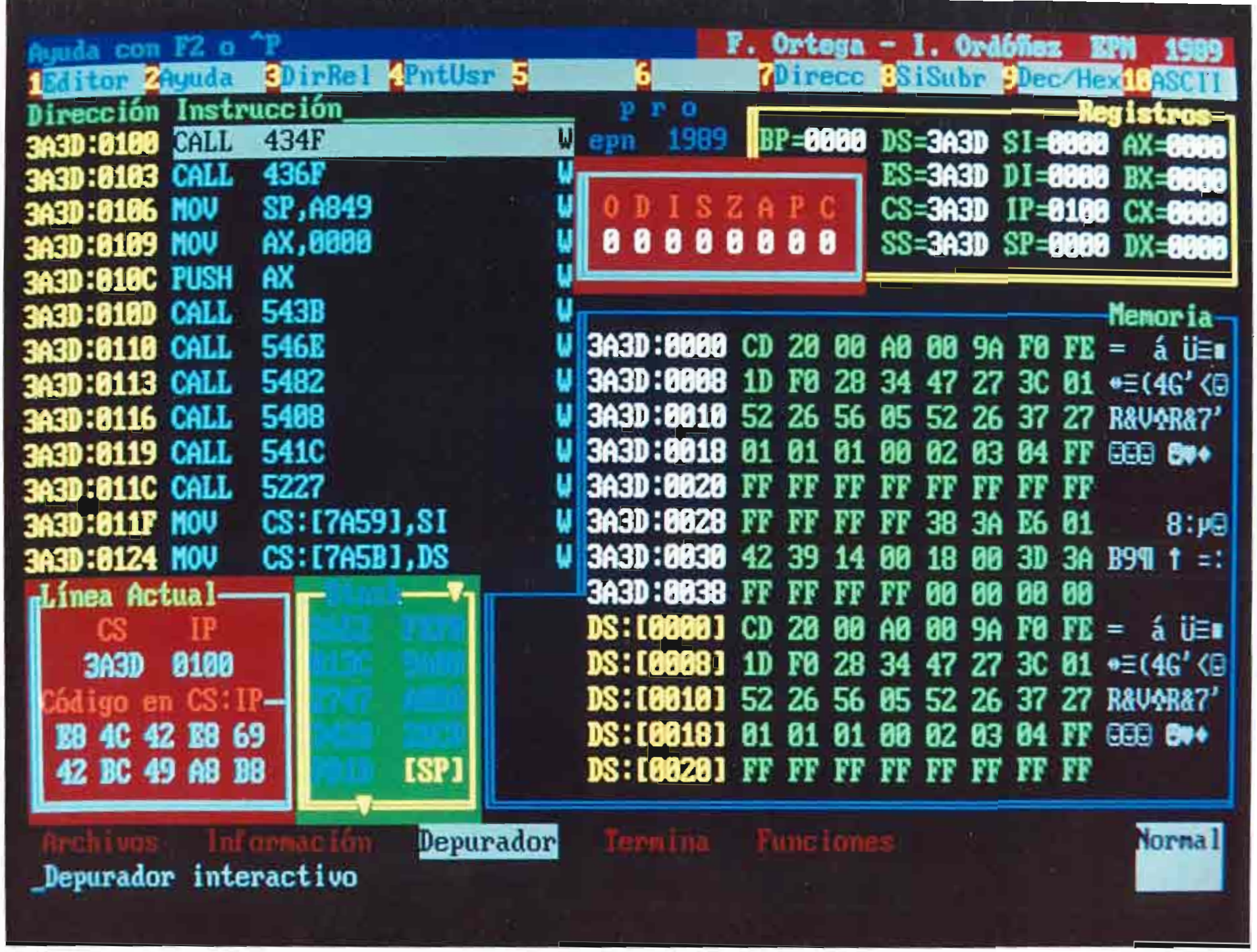
Aunque el paquete está diseñado para fácil manejo y aprendizaje, debe notarse que el usuario ha de tener un conocimiento relativamente bueno del microprocesador 8086 y de sus técnicas de programación, el modo de acción del DOS y la arquitectura de las computadoras de la familia de la IBM PC para obtener el máximo provecho del depurador.

Sugerimos que, en vista de que esta Tesis provee a la Facultad de un paquete de depuración más adecuado a sus necesidades que los existentes, se reoriente el contenido de algunas materias que actualmente se dictan, a fin de dar mayor importancia a la comunicación de datos entre microcomputadores, campo al que muchos futuros profesionales se van a encontrar ligados y que hasta ahora no ha sido abordado con suficiente ímpetu.


Ayuda con F2 o ^P
F. Ortega - I. Ordóñez EPN 1989
1Editor 2Ayuda 3DirRel 4PntUsr 5 6 7Direcc 8SiSubr 9Dec/Hex 10ASCII
Dirección Instrucción
3A3D:0100 CALL 434F
3A3D:0103 CALL 436F
3A3D:0106 MOV SP,A849
3A3D:0109 MOV AX,0000
3A3D:010C PUSH AX
3A3D:010D CALL 543B
3A3D:0110 CALL 546E
3A3D:0113 CALL 5482
3A3D:0116 CALL 5408
3A3D:0119 CALL 541C
3A3D:011C CALL 5227
3A3D:011F MOV CS:[7A59],SI
3A3D:0124 MOV CS:[7A5B],DS
pro
epn 1989
O D I S Z A P C
0 0 0 0 0 0 0 0
Registros
BP=0000 DS=3A3D SI=0000 AX=0000
ES=3A3D DI=0000 BX=0000
CS=3A3D IP=0100 CX=0000
SS=3A3D SP=0000 DX=0000
Memoria
3A3D:0000 CD 20 00 A0 00 9A F0 FE
3A3D:0008 1D F0 28 34 47 27 3C 01
3A3D:0010 52 26 56 05 52 26 37 27
3A3D:0018 01 01 01 00 02 03 04 FF
3A3D:0020 FF FF FF FF FF FF FF FF
3A3D:0028 FF FF FF FF 38 3A E6 01
3A3D:0030 42 39 14 00 18 00 3D 3A
3A3D:0038 FF FF FF FF 00 00 00 00
DS:[0000] CD 20 00 A0 00 9A F0 FE
DS:[0008] 1D F0 28 34 47 27 3C 01
DS:[0010] 52 26 56 05 52 26 37 27
DS:[0018] 01 01 01 00 02 03 04 FF
DS:[0020] FF FF FF FF FF FF FF FF
Línea Actual
CS IP
3A3D 0100
Código en CS:IP
E8 4C 42 E8 69
42 DC 49 A8 B8
[SP]
Archivos Información Depurador Termina Funciones
Normal
Depurador interactivo


Ayuda con F2 o ^P
F. Ortega - I. Ordóñez EPN 1989
1Editor 2Ayuda 3DirRel 4PntUsr 5 6 7Direcc 8SiSubr 9Dec/Hex 10ASCII
Dirección Instrucción
3A3D:0100 CALL 434F W
3A3D:0103 CALL 436F W
3A3D:0106 MOV SP,A849 W
3A3D:0109 MOV AX,0000 W
3A3D:010C PUSH AX W
3A3D:010D CALL 543B W
3A3D:0110 CALL 546E W
3A3D:0113 CALL 5482
3A3D:0116 CALL 5408
3A3D:0119 CALL 541C
3A3D:011C CALL 5227
3A3D:011F MOV CS:[7A59],SI W
3A3D:0124 MOV CS:[7A5B],DS W
pro
epn 1989
O D I S Z A P C
0 0 0 0 0 0 0 0
Registros
BP=0000 DS=3A3D SI=0000 AX=0000
ES=3A3D DI=0000 BX=0000
CS=3A3D IP=0100 CX=0000
SS=3A3D SP=0000 DX=0000
Memoria
3A3D:0000 CD 20 00 A0 00 9A F0 FE
3A3D:0028 FF FF FF FF 38 3A E8 01
3A3D:0030 42 39 14 00 18 00 3D 3A
3A3D:0038 FF FF FF FF 00 00 00 00
DS:[0000] CD 20 00 A0 00 9A F0 FE
DS:[0008] 1D F0 28 34 47 27 3C 01
DS:[0010] 52 26 56 05 52 26 37 27
DS:[0018] 01 01 01 00 02 03 04 FF
DS:[0020] FF FF FF FF FF FF FF FF
Programa terminado normalmente
Presiona una tecla para continuar.
Línea Actual
CS IP
3A3D 0100
Código en CS:IP
E8 4C 42 E8 69
42 BC 49 A8 B8
Stack
2652 FEF8
013C 9A00
2747 A000
3428 20CD
F810 [SP]
Paso Corre eJecuta Rupturas Ins Ens Memoria Funciones
Normal
Ejecuta la próxima instrucción


Ayuda con F2 o ^P
F. Ortega - I. Ordóñez EPN 1989
1Editor 2Ayuda 3DirRel 4PntUsr 5 6 7Direcc 8SiSubr 9Dec/Hex 10ASCII
Dirección Instrucción
3A3D:0100 CALL 434F
3A3D:0103 CALL 436F
3A3D:0106 MOV SP,A849
3A3D:0109 MOV AX,0000
3A3D:010C PUSH AX
3A3D:010D CALL 543B
3A3D:0110 CALL 546E
3A3D:0113 CALL 5482
3A3D:0116 CALL 5408
3A3D:0119 CALL 541C
3A3D:011C CALL 5227
3A3D:011F MOV CS:[7A59],SI
3A3D:0124 MOV CS:[7A5B],DS
p r o
epn 1989
Registros
BP=0000 DS=3A3D SI=0000 AX=0000
ES=3A3D DI=0000 BX=0000
CX=0000
DX=0000
Puntos de Ruptura
(* indica punto activo)
N° Nombre Dirección
01 * punto1 3A3D:0106
02 * call rut 3A3D:010D
03 * continua 3A3D:011F
04 * video 3A3D:0148
Memoria
Escribe nombre del punto de ruptu
video
Dirección del punto de ruptura:
3A3D : _148
1D F0 28 34 47 27 3C 01
52 26 56 05 52 26 37 27
01 01 01 00 02 03 04 FF
FF FF FF FF FF FF FF FF
Operación terminada. Presiona una tecla.


Ayuda con F2 o ^P
F. Ortega - I. Ordóñez EPN 1989
1Editor 2Ayuda 3DirRel 4PntUsr 5 6 7Direcc 8SiSubr 9Dec/Hex 10ASCII
Dirección Instrucción
3A3D:0100 CALL 434F W
3A3D:0103 CALL 436F W
3A3D:0106 MOV SP,A849 W
3A3D:0109 MOV AX,0000 W
3A3D:010C PUSH AX W
3A3D:010D CALL 543B W
3A3D:0110 CALL 546E W
3A3D:0113 CALL 5482 W
3A3D:0116 CALL 5408 W
3A3D:0119 CALL 541C W
3A3D:011C CALL 5227 W
3A3D:011F MOV CS:[7A59],SI W
3A3D:0124 MOV CS:[7A5B],DS W
pro
epn 1989
Registros
BP=0000 DS=3A3D SI=0000 AX=0000
ES=3A3D DI=0000 BX=0000
CS=3A3D IP=0100 CX=0000
SS=3A3D SP=0000 DX=0000
O D I S Z A P C
0 0 0 0 0 0 0 0
Memoria
3A3D:0000 CD 20 00 A0 00 9A F0 FE
3A3D:0008 1D F0 28 34 47 27 3C 01
3A3D:0010 52 26 56 05 52 26 37 27
3A3D:0018 01 01 01 00 02 03 04 FF
3A3D:0020 FF FF FF FF FF FF FF FF
3A3D:0028 FF FF FF FF 38 3A E6 01
3A3D:0030 42 39 14 00 18 00 3D 3A
3A3D:0038 FF FF FF FF 00 00 00 00
DS:[0000] CD 20 00 A0 00 9A F0 FE
DS:[0008] 1D F0 28 34 47 27 3C 01
DS:[0010] 52 26 56 05 52 26 37 27
DS:[0018] 01 01 01 00 02 03 04 FF
DS:[0020] FF FF FF FF FF FF FF FF
Línea Actual
CS IP
3A3D 0100
Código en CS:IP
E8 4C 42 E8 69
42 BC 49 A8 B8
[SP]
Archivos Información Depurador Termina Funciones
Normal
_Depurador interactivo

```
;ÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍ
;ÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍ          FMAC.ASM          ÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍ
;ÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍ
;ÍÍÍÍÍ     Macros usados en el archivo flujo.asm.     ÍÍÍÍÍ
;ÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍ
;
;-------------------------------------------------------------------------
;Macro          Parametros      Funcion
;-------------------------------------------------------------------------
;actuar                         salta a una de las dir. en una tabla
;avuda          dir,nlin,color  presenta ventana con instrucciones
;bufcla         pnclave,pbuf    encuentra la posicion de un buffer
;curs           x               pone un "cursor" en x,24
;explica        pnclave         muestra la explicacion(PNCLAVE)
;getkey                         recibe dato del teclado (BIOS:
;getmac                         da el # del macro con nombre en NOMBUF
;getxor                         obtiene ultimo nibble <>0 en BUFFER
;guarpantalla                   guarda una pantalla
;iluminaon                      resalta la opcion actual
;inbuf          dir             recibe string desde teclado (BIOS)
;indmac                         da una lista de los macros en el indice
;limpant                        limpia la pantalla
;menaba                         cursor abajo en ventana de subopc.
;menarr                         cursor arriba
;mender                         cursor a la derecha
;menizq                         cursor a la izquierda
;menulinea      pnclave         muestra menu(PNCLAVE)
;menuvent                       muestra menu en forma de ventana
;mostrar        pdir,plin,pcol  muestra pregunta y espera respuesta
;movbuf         pbuf1,pbuf2     PBUF1 <-- PBUF2
;nocurs         x               pone atributo normal en x,24
;nunivel        pbuf,pniv,pnum  pone PNUM en PBUF(PNIV)
;opcion?                        espera que se elija una opcion
;pchar          car             escribe CAR (DOS)
;pmenu          pnlin,pcolor    muestra menu string apuntado por AX
;posdir         pvarx,pvary,dir convierte pos. de pantalla en direccion
;restpantalla                   restaura ultima pantalla salvada
;rutina                         pasa control al usuario y lo recibe
;salvareg                       push a todo
;status1                        muestra fase del programa
;statusv                        muestra datos importante del usuario
;traereg                        pop a todo
;valida?                        ve si una opcion es del menu del usuario
;subrut                         subrutinas de todos los macros
;-------------------------------------------------------------------------

;ÍÍÍÍÍ ACTUAR ÍÍÍÍÍÍÍÍÍÍÍÍÍÍ
;Busca el AL-esimo elemento de la tabla de direcciones
;que ira despues de MTABLA para saltar a esa direccion.
actuar  macro
        local   ctabla
        push    bx
```

```
        mov     bx,offset mtabla
        call    macl
        dec     bx
        jmp     dword ptr direct
mtabla:
        endm

;;;;;; AYUDA  ;;;;;;;;;;;;;;;; 16/02/89
;Escribe NLIN lineas separadas por ceros desde DIR en COLOR.
;Ocupa columnas 1 a 78 y las filas del centro de la pantalla.
ayuda   macro   dir,nlin,color
        push    ax
        push    cx
        mov     cl,nlin
        mov     ch,color
        mov     ax,offset dir
        call    may
        pop     cx
        pop     ax
        endm

;;;;;; BUFCLA  ;;;;;;;;;;;;;;;;
;Busca PBUF en TABBUF y pone su ubicacion en PNCLAVE
bufcla  macro   pnclave,pbuf
        push    si
        push    ax
        mov     si,offset pbuf
        call    mbc
        mov     al,mbcbyte
        mov     pnclave,al
        pop     ax
        pop     si
        endm

;;;;;; CURS    ;;;;;;;;;;;;;;;; 17/02/89
;pone el atributo de (x,24) en inverso.
curs    macro   x
        push    ax
        mov     al,x
        mov     ah,bnsubb
        call    mcu
        pop     ax
        endm

;;;;;; EXPLICA ;;;;;;;;;;;;;;;;
;presenta explicacion de opcion cuyo # es pnclave
explica macro   pnclave
        push    cx
        mov     cl,pnclave
        call    mex
        pop     cx
        endm
```

```
;;;;;;; GETKEY ;;;;;;;;;;;;;;;
;Espera hasta recibir un dato del teclado y lo manda en GETWORD
getkey  macro
        call    mgk
        mov     ax,getword
        endm

;;;;;;; GETNMAC ;;;;;;;;;;;;;;; 16/02/89
;Obtiene NUMMAC (numero del macro en el indice)
;buscando su nombre NOMBUF en ese indice.
getnmac macro
        call    mgn
        endm

;;;;;;; GETXPR ;;;;;;;;;;;;;;;; 10/02/89
;Obtiene de BUFFER el ultimo nibble distinto de 0, en XPREV.
;Sirve para regresar (con ^L o rutina inversa) a
;la opcion que dio origen a cierto menu.
getxpr  macro
        call    mgx
        endm

;;;;;;; GUARPANTALLA ;;;;;;;;;;
;guarda la pantalla en PANBUF
guarpantalla    macro
        call    mgp
        endm

;;;;;;; ILUMINAOP ;;;;;;;;;;;;;
;Resalta la opcion actual (indicada por la variable X)
iluminaop       macro
        call    mio
        endm

;;;;;;; INBUFF ;;;;;;;;;;;;;;;; 17/02/89
;Recibe desde el teclado el buffer DIR cuya longitud se indica en
;el primer byte del buffer. En el segundo byte se obtendra el
;numero de bytes recibidos. Solo se aceptan letras o numeros.
;Para edicion, solo Backspace. Pita si se ha llegado al limite.
inbuff  macro   dir
        push    si
        mov     si,offset dir
        call    mib
        pop     si
        endm

;;;;;;; INDMAC ;;;;;;;;;;;;;;;; 16/02/89
;Llena la lista LISMAC en base al indice de macros
;en AREAMAC. Reemplaza los ceros por espacios (20H).
indmac  macro
        call    mim
```

```
        endm

;;;;;; LMPPNT  ;;;;;;;;;;;;;;;; 15/04/89
;Borra la pantalla
lmppnt  macro
        push    di
        push    cx
        push    ax
        mov     cx,2000
        mov     di,0
        mov     ax,0720h
        rep     stosw
        pop     ax
        pop     cx
        pop     di
        endm

;;;;;; MENABA  ;;;;;;;;;;;;;;;;
;Mueve cursor abajo en la ventana de subopciones.
menaba  macro
        call    mbab
        endm

;;;;;; MENARR  ;;;;;;;;;;;;;;;;
;Similar a MENABA.
menarr  macro
        call    mba
        endm

;;;;;; MENDER  ;;;;;;;;;;;;;;;;
;Mueve cursor de opcion actual a la derecha.
mender  macro
        call    mbd
        endm

;;;;;; MENIZQ  ;;;;;;;;;;;;;;;;
;Similar a MENDER.
menizq  macro
        call    mbi
        endm

;;;;;; MENULINEA ;;;;;;;;;;;;;;
;Muestra menu (PNCLAVE)
menulinea       macro   pnclave
        push    cx
        mov     cl,pnclave
        call    mml
        pop     cx
        endm

;;;;;; MENUVENT ;;;;;;;;;;;;;;;
;Muestra menu(NCLAVE1) en forma de ventana.
```

```
@menuvert       macro
        call    @mv
        endm

;;;;;;; MOSTRAR ;;;;;;;;;;;;;;;;
;Muestra pregunta apuntada por PDIR y espera respuesta. La pregunta
;debe tener esta forma: Mensaje,0,# de respuestas, lista de
;respuestas. En AL sale el numero de la respuesta elegida.
@mostrar macro  pdir,plin,pcol
        push    ax
        mov     ah,plin
        mov     al,pcol
        mov     @oslin,ah
        mov     @oscol,al
        mov     ax,offset pdir
        call    @@@s
        pop     ax
        mov     al,@@@s
        endm

;;;;;;; MOVBUF  ;;;;;;;;;;;;;;;;
;Mueve PBUF1 a PBUF2.
@movbuf macro   pbuf2,pbuf1
        push    si
        push    di
        mov     si,offset pbuf1
        mov     di,offset pbuf2
        call    @@b
        pop     di
        pop     si
        endm

;;;;;;; NOCURS  ;;;;;;;;;;;;;;;; 17/02/89
;pone el atributo de (x,24) en normal.
nocurs  macro   x
        push    ax
        mov     al,x
        mov     ah,bnorm
        call    @cu
        pop     ax
        endm

;;;;;;; NUNIVEL ;;;;;;;;;;;;;;;;
;Pone PNUM en el buffer PBUF en el nivel PNIV.
nunivel macro   pbuf,pniv,pnum
        push    cx
        push    dx
        mov     cl,pniv
        mov     ch,pnum
        mov     dx,offset pbuf
        call    @nn
        pop     dx
```

```
        pop     cx
        endm

;;;;;;; OPCION? ;;;;;;;;;;;;;;;;
;Espera que se digite una opcion. Saca en MOPACT el numero de
;opcion. Las opciones son: Flechas,ENTER,^D,^K,^L,^P y otros.
;Ademas es aqui donde se almacenan los datos al grabar macros
;y se leen del macro en vez del teclado al correr o probarlos.
opcion? macro
        call    mop
        endm

;;;;;;; PCHAR  ;;;;;;;;;;;;;;;;
;Usa funcion del DOS para escribir CAR en pantalla.
pchar   macro   car
        push    dx
        mov     dl,car
        call    mpch
        pop     dx
        endm

;;;;;;; PMENU  ;;;;;;;;;;;;;;;;
;Muestra menu apuntado por AX. Al salir MPMPRA
;da la direccion del primer byte luego del 0.
pmenu   macro   pnlin,pcolor
        push    cx
        push    cx
        mov     cl,pnlin
        mov     dh,pcolor
        call    mnm
        pop     dx
        pop     cx
        endm

;;;;;;; POSDIR  ;;;;;;;;;;;;;;;;
;Da la direccion PDIR correspondiente a (PVARX,PVARY)
;para acceso directo a la memoria de video.
posdir  macro   pvarx,pvary,pdir
        push    ax
        push    bx
        push    cx
        mov     cl,pvary
        mov     bl,pvarx
        call    mpd
        mov     bx,mpddir
        mov     pdir,bx
        pop     cx
        pop     bx
        pop     ax
        endm

;;;;;;; RESTPANTALLA ;;;;;;;;;;
```

```
;Trae de PANBUF la ventana que fue salvada en GUARPANTALLA.
restpantalla    macro
        call    mrp
        endm

;;;;;;; RUTINA  ;;;;;;;;;;;;;;;
;Pone la direccion de la rutina(nCLAVE) de usuario en DIRSAL y la de
;regreso en MRAUX2 y MRAUX3. Pasa control al usuario, que debe regresar
;poniendo RETORNO al fin de su rutina para regresar a MRUT1.
rutina  macro
        mov     cl,nclave
        mov     al,num_but
        mul     cl
        add     ax,offset tabin         ;busco dir. de rutina(nclave)
        add     ax,4
        mov     si,ax
        mov     bx,[si]
        mov     al,[si+2]               ;y tipo de rutina.
        mov     tiprut,al               ;Preparo el salto.
        mov     mraux1,al               ;guardo tipo de rutina.
        mov     bp,offset dirsal
        mov     ds:[bp],bx
        mov     ds:[bp+2],cs
        mov     bx,offset mrut1         ;Preparo dir. de regreso.
        mov     mraux2,cs
        mov     mraux3,bx
        jmp     dword ptr dirsal        ;Control al usuario.
mrut1:  mov     al,tiprut               ;Regreso a FLUJO.
        cmp     al,mraux1               ;Ha cambiado TIPRUT?
        je      mrut2
        mov     contmcr,0ffh            ;si, cod. de error de macros,
        mov     x,1                     ;ademas X,Y previos pueden
        mov     y,0                     ;estar errados.
mrut2:
        endm

;;;;;;; SALVAREG ;;;;;;;;;;;;;;;
;Push a todo.
salvareg        macro
        push    ax
        push    bx
        push    cx
        push    dx
        push    bp
        push    si
        push    di
        push    ds
        push    es
        push    ss
        endm

;;;;;;; STATUSF ;;;;;;;;;;;;;;;; 16/02/89
```

```
;Presenta 6 bytes apuntados por FASEPTR en (72,23)
;Se usa para indicar la fase actual del programa.
status1 macro
        call    msf
        endm

;;;;;;; STATUSv ;;;;;;;;;;;;;;;; 16/02/89
;Presenta 6 bytes apuntados por VALPTR en (73,24)
;Se usa para mostrar importantes datos del usuario.
statusv macro
        call    msv
        endm

;;;;;;; TRAEREG ;;;;;;;;;;;;;;
;Pop a todo.
traereg macro
        pop     es
        pop     ds
        pop     di
        pop     si
        pop     bp
        pop     dx
        pop     cx
        pop     bx
        pop     ax
        endm

;;;;;;; VALIDA? ;;;;;;;;;;;;;;;
;Si en OPCION? se digito una opcion distinta de las que maneja
;normalmente, ACTUAR manda el flujo aca para ver si es una opcion
;puesta por el usuario en el menu. Si es asi, AL=numero de opcion.
;Si el primer byte despues de la ultima opcion del menu es FF se
;supone que la ultima es la "opcion de escape" para todas las
;posibilidades que no consten en el menu.
valida? macro
        call    mv?
        mov     al,mv?aux
        endm

;;;;;;; SUBRUT ;;;;;;;;;
;Este es un macro que contiene todas las subrutinas llamadas por los
;otros macros de este archivo. Se hizo de esta manera para evitar el
;uso de demasiadas banderas, y que su ausencia no produzca "Phase
;errors" al momento del ensamblado.
subrut  macro

mact:
        salvareg
        dec     al              ;pongo el AL-esimo elemento de la
        mov     cl,2            ;tabla en DIRACT y CS en DIRACT+2
        mul     cl              ;para hacer el JMP al regresar.
```

```
        add     ax,5
        mov     bp,offset direct
        push    si
        mov     si,ax
        mov     bx,cs:[si]
        pop     si
        mov     ds:[bp],bx
        mov     ds:[bp+2],cs
        traereg
        retn

may:
        push    dx
        push    bx
        mov     bx,ax
        mov     dl,25           ;centro el texto en pantalla:
        sub     dl,cl           ;linea inicial= (25-#lineas)/2 -1
        shr     dl,1
        dec     dl
        mov     maylin,dl
        dec     maylin
        mov     maycol,ch
        mov     ax,offset blanco
may1:   pmenu   maylin,maycol   ;primero una linea en blanco.
        inc     maylin
        mov     mpmprx,bx
may2:   mov     ax,mpmprx       ;Ahora el resto de lineas. MPMPRX
        pmenu   maylin,maycol   ;da la direccion del primer byte
        inc     maylin          ;despues del 0.
        dec     cl
        jnz     may2
        mov     ax,offset blanco
        pmenu   maylin,maycol   ;Linea en blanco al final.
        pop     bx
        pop     dx
        retn

mbc:
        salvareg
        mov     al,nmaxbuf      ;Busca un buffer en la tabla TABBUF
        cmp     al,0            ;y entrega el numero clave
        jne     mbc0            ;que servira para obtener toda la
        jmp     mbc4            ;informacion necesaria acerca de la
mbc0:   mov     mbcmenu,0       ;opcion representada por el buffer.
        mov     mbcbuf,si       ;Si el buffer no existe, clave=FF.
        mov     mbcpos,offset tabbuf
mbc1:   mov     mbcbyte,0
        mov     bx,mbcbuf
mbc3:   mov     ah,[bx]
        inc     bx
        mov     bp,mbcpos
        xor     ch,ch
```

```
        mov     cl,mocbyte
        inc     mbcbyte
        mov     di,cx
        cmp     al,es:[bp+di]
        jnz     moc2
        cmp     mocbyte,0
        jnz     moc1
        mov     al,mocmenu
        jmp     moc3
moc2:   inc     mocmenu
        mov     al,nmaxbuf
        cmp     al,mocmenu
        je      moc4
        add     mocpos,8
        jmp     moc1
moc4:   mov     al,0ffh
moc3:   mov     mocbyte,al
        traereg
        retn

mcu:
        salvareg                ;Pone el atributo dado por AH en
        posdir  0,24,mcudir     ;la posicion (AL,24)
        mov     si,mcudir
        mov     bh,ah
        mov     cl,2
        mul     cl
        add     si,ax
        inc     si
        mov     es:[si],bh
        traereg
        retn

mex:
        salvareg                ;Busca en TABINF la direccion de la
        mov     al,ninfbuf      ;explicacion y la pone en linea 23.
        mul     cl
        add     ax,offset tabinf
        mov     bx,ax
        mov     ax,[bx]
        pmenu   24,colexp
        traereg
        retn

mgk:
        salvareg                ;Uso funcion del BIOS que espera
        mov     ah,0            ;que se pulse una tecla.
        int     16h
        mov     getword,ax
        traereg
        ret
```

```
rgn:
        salvareg                ;Busca si el nombre en NOMBUF esta
        push    es              ;en la lista de macros y da su numero
        push    ds              ;+80h en NUMMAC si asi ocurre. Si es
        pop     es              ;nuevo, da el # del primero libre.
        mov     dh,0ffh         ;Si ya no hay espacio, da FF.
mgn1:   inc     dh
        cmp     dh,8
        je      mgn2
        mov     si,offset nombuf
        add     si,2            ;bytes 0 y 1 son de uso de otro macro.
        mov     di,offset areamac
        mov     al,8
        mul     dh
        add     di,ax
        inc     di              ;byte 0 es la longitud del macro.
        mov     cx,7
        repe    cmpsb           ;comparo.
        jne     mgn1            ;si no es igual, comparo otro.
        add     dh,80h          ;macro viejo.
        mov     nummac,dh
        jmp     mgn4
mgn2:   mov     dh,0ffh         ;Busco primero libre.
mgn21:  inc     dh
        cmp     dh,8
        je      mgn3
        mov     di,offset areamac
        mov     al,8
        mul     dh
        add     di,ax
        mov     al,0
        mov     cx,8
        repe    scasb
        jne     mgn21
        mov     nummac,dh       ;macro nuevo.
        jmp     mgn4
mgn3:   mov     nummac,0ffh     ;macro nuevo sin sitio.
mgn4:   pop     es
        traereg
        retn

mgx:
        salvareg                ;se divide NIVEL/2, se suma a direccion
        mov     cl,nivel        ;inicial de BUFFER, y se obtiene el msn
        shr     cl,1            ;distinto de 0 en ese byte.
        xor     ch,ch           ;Ese es el ultimo nibble<>0 en BUFFER
        mov     si,offset buffer
        add     si,cx
        mov     al,[si]
        mov     ah,al
        shr     ah,1
        shr     ah,1
```

```
        sar     ax,1
        sar     ax,1
        cmp     ax,0
        jz      mgx1
        add     al,ah
mgx1:   mov     xorev,al
        traereg
        retn

mgp:
        salvareg                ;paso pantalla a PANBUF.
        xor     si,si
        mov     di,offset panbuf
        mov     bx,ds
        mov     es,bx
        mov     ds,videoseg
        mov     cx,3680
        rep     movsb
        traereg
        retn

mio:
        salvareg                ;Resalta la opcion actual, dada por X.
        mov     es,videoseg
        mov     dl,collart
        mov     al,x
mio13:  posdir  1,23,miodir     ;trabajo directo en pantalla.
        mov     si,miodir       ;busco la X-esima palabra en la
        mov     di,si           ;linea 23 y cambio su atributo.
        push    es
        pop     ds
        mov     cl,al
mio131: lodsw
        cmp     al,20h          ;salto espacios iniciales.
        je      mio131
        sub     si,2
        dec     cl
        jz      mio16
mio14:  lodsw
        cmp     al,20h          ;salto palabras.
        jne     mio14
mio15:  lodsw
        cmp     al,20h
        je      mio15
        dec     si
        dec     si
        dec     cl
        jnz     mio14
mio16:  mov     di,si           ;resalto palabra.
        lodsw
        mov     ah,dl
        cmp     al,20h
```

```
        je      mio17
        cmp     al,0
        je      mic17
        stosw
        jmp     mio1o
mio17:  traereg
        retn

mib:
        salvareg                ;obtengo string desde teclado.
        mov     cl,[si]         ;maximo # de caracteres
        mov     ch,0            ;contador
        mov     dx,si           ;puntero auxiliar
        add     si,2            ;bytes 0 y 1 de uso del macro
        posdir  1,24,mibdir     ;direccion de primer caracter
        mov     bx,mibdir
mib1:   mov     mibx,ch
        inc     mibx
        curs    mibx
mib2:   getkey
        cmp     al,08h          ;Backspace?
        jne     mib21
        jmp     mib4
mib21:  cmp     al,0dh          ;Enter?
        jne     mib22
        jmp     mib5
mib22:  cmp     al,30h          ;<"0"?
        jc      mib2
        cmp     al,7bh          ;>"z"?
        jnb     mib2
        cmp     al,3ah          ;es un numero?
        jb      mib3
        cmp     al,41h          ;"9"<al<"A"?
        jb      mib2
        cmp     al,5bh          ;es una mayuscula?
        jb      mib3
        cmp     al,61h          ;"Z"<al<"a"?
        jb      mib2
        sub     al,20h          ;es minuscula, hacer mayuscula
mib3:   nocurs  mibx
        cmp     ch,cl           ;contador=maximo?
        jne     mib31
        pchar   07h             ;si, beep.
        jmp     mib1
mib31:  mov     [si],al         ;AL al buffer
        inc     si              ;apunto prox. direccion
        mov     es:[bx],al      ;AL a pantalla
        add     bx,2            ;apunto prox. direccion
        inc     ch              ;contador+1
        jmp     mib1
mib4:   nocurs  mibx
        cmp     ch,0            ;buffer vacio?
```

```
        jne     mib41
        cmpar   07h             ;si, beep.
        jmp     mib1
mib41:  dec     si              ;borro ultimo caracter
        mov     byte ptr[si],0  ;en el buffer
        sub     bx,2            ;y en pantalla
        mov     byte ptr es:[bx],20h
        dec     ch              ;contador-1
        jmp     mib1
mib5:   nocurs  mibx
        mov     si,dx           ;fin de rutina: byte 1
        inc     si              ;indica # de bytes recibidos.
        mov     [si],ch
        traereg
        retn

mia:
        salvareg                ;pone la lista de macros actual
        push    es              ;en LISMAC. Reemplaza los 00 con
        push    ds              ;2E (puntos).
        pop     es
        mov     si,offset areamac
        mov     di,offset lismac
        mov     dl,8
mia1:   inc     si
        mov     al,20h
        stosb
        mov     cl,7
mia2:   lodsb
        cmp     al,0
        jne     mia3
        mov     al,2eh
mia3:   stosb
        dec     cl
        jnz     mia2
        dec     dl
        jnz     mia1
        pop     es
        traereg
        retn

mmab:
        salvareg
        mov     cl,nclave1      ;Busca el # de subopciones que tiene
        mov     al,ninfbuf      ;el menu de la opcion actual dada por
        mul     cl              ;NCLAVE1. Si Y supera ese #, Y=0.
        add     ax,offset tabinf
        inc     ax
        inc     ax
        push    bx
        mov     bx,ax
        mov     si,[bx]
```

```
        pop     bx
mcab1:  lodsb
        cmp     al,0
        jne     mcab1
        lodsb
        inc     al
        inc     x
        cmp     al,x
        jne     mmab2
        mov     y,0
mmab2:  traereg
        retn

mma:
        salvareg
        mov     cl,nclave1      ;Busca # de subopc. de menu(NCLAVE1)
        mov     al,ninfbuf      ;Si Y=FF, Y igual a ese #.
        mul     cl
        add     ax,offset tabinf
        inc     ax
        inc     ax
        push    bx
        mov     bx,ax
        mov     si,[bx]
        pop     bx
mma1:   lodsb
        cmp     al,0
        jne     mma1
        lodsb
        dec     y
        cmp     y,0ffh
        jne     mma2
        mov     y,al
mma2:   traereg
        retn

mmd:
        salvareg
        mov     cl,nclave       ;Busca # de opciones de menu actual.
        mov     al,ninfbuf      ;Si X supera ese valor, X=1.
        mul     cl              ;Siempre Y=0.
        add     ax,offset tabinf
        inc     ax
        inc     ax
        push    bx
        mov     bx,ax
        mov     si,[bx]
        pop     bx
mmd1:   lodsb
        cmp     al,0
        jne     mmd1
        lodsb
```

```
        inc     al
        inc     x               ;a la derecha.
        cmp     al,7
        jne     mm02
        mov     x,1
mm02:   mov     y,0
        traereg
        retn

mmi:
        salvareg
        mov     cl,nclave       ;Busca # de opciones de menu actual.
        mov     al,ninfbuf      ;Si X=0, X igual a ese #. Siempre Y=0.
        mul     cl
        add     ax,offset tabinf
        inc     ax
        inc     ax
        push    bx
        mov     dx,ax
        mov     si,[bx]
        pop     bx
mmi1:   lodsb
        cmp     al,0
        jne     mmi1
        lodsb
        dec     x
        cmp     x,0
        jne     mmi2
        mov     x,al
mmi2:   mov     y,0
        traereg
        retn

mmi:
        salvareg
        mov     al,ninfbuf      ;Muestra el menu(CL) y resalta la
        mul     cl              ;opcion actual.
        add     ax,offset tabinf
        inc     ax
        inc     ax
        mov     bx,ax
        mov     ax,[bx]
        pmenu   23,colfin
        iluminaop
        traereg
        retn

mmv:
        salvareg                        ;Muestra ventana se subopc.
        mov     cl,nclave1              ;primero veo si la opcion
        mov     al,ninfbuf              ;tiene algun submenu.
        mul     cl                      ;Si es asi, la rutina
```

```
        add     ax,offset tabinf        ;debe ser de tipo 3.
        adc     ax,6
        mov     bx,ax
        cmp     byte ptr[bx],3
        jz      mnv3
        jmp     mnv7
mnv3:   mov     cl,nclave               ;Ahora busco la direccion
        mov     al,ninfbuf              ;del menu al que pertenece
        mul     cl                      ;esta opcion en TABINF.
        add     ax,offset tabinf
        inc     ax
        inc     ax
        push    bx
        mov     bx,ax
        mov     si,[bx]                 ;SI apunta al inicio del menu.
        pop     bx
        mov     cl,0                    ;la opcion actual (x) para saber
        mov     dl,x                    ;columna inicial de ventana.
mnv30:  lodsb                           ;Salto espacios iniciales
        inc     cl                      ;para los cuanto.
        cmp     al,20h
        je      mnv30
        cmp     x,1                     ;si x=1 ya tengo la columna
        jnz     mnv31
        dec     cl
        mov     mnvx1,cl
        jmp     mnv4
mnv31:  dec     dl                      ;DL: cuantas opc mas debo saltar.
mnv32:  lodsb                           ;Cuento bytes hasta encontrar
        inc     cl                      ;un espacio (CL contador).
        cmp     al,20h
        jne     mnv32
mnv33:  lodsb                           ;sumo espacios entre opciones
        inc     cl
        cmp     al,20h
        je      mnv33
        dec     dl                      ;una opc menos que saltar
        jnz     mnv32
        dec     cl                      ;Termino. Ventana empieza en
        mov     mnvx1,cl                ;columna anterior.
mnv4:   mov     mnvcar,0
        mov     cl,nclave1              ;Ahora busco direccion de menu
        mov     al,ninfbuf              ;asociado a la opcion actual
        mul     cl                      ;en TABINF
        add     ax,offset tabinf
        inc     ax
        inc     ax
        push    bx
        mov     bx,ax                   ;SI apunta a primer byte
        mov     si,[bx]                 ;de ese menu
        pop     bx
        push    si
```

```
mmv40:  lodsb                           ;no toco en cuenta los
        cmp     al,20h                  ;espacios iniciales.
        je      mmv40
        dec     si
        mov     cx,1                    ;Veo cuantas filas y columnas
mmv41:  lodsb                           ;ocupa la ventana
        cmp     al,20h                  ;Los espacios no cuentan
        jz      mmv42
        cmp     al,0                    ;termina si AL=0
        jne     mmv410
        mov     mmvlin,cl               ;# de subopciones del menu
        jmp     mmv5
mmv410: inc     ch                      ;CH: # de car de subopcion
        cmp     ch,mmvcar               ;MMVCAR: maximo CH logrado
        jb      mmv41
        mov     mmvcar,ch
        jmp     mmv41
mmv42:  lodsb                           ;saltar espacios
        cmp     al,20h
        je      mmv42
        cmp     al,0                    ;termina si AL=0
        jne     mmv43
        mov     mmvlin,cl               ;# de subopciones del menu
        jmp     mmv5
mmv43:  inc     cl                      ;otra subopcion
        mov     ch,0                    ;# de car = 0
        dec     si
        jmp     mmv41
mmv5:   pop     si
        mov     al,mmvlin               ;Busco la linea inicial
        inc     al                      ;de la ventana incluidas
        inc     al                      ;dos lineas de recuadro.
        mov     ah,23
        sub     ah,al
        mov     mmvlin,ah
        mov     mmvlin1,ah              ;servira despues
        inc     mmvcar
        posdir  mmvx1,mmvlin,mmvdir     ;dir. de esquina superior
        mov     di,mmvdir               ;izquierda en pantalla
        mov     es,videoseg
        mov     al,esqsi                ;dibujar esa esquina
        mov     ah,colven               ;en color de ventana
        stosw
        mov     al,linsup               ;dibuja linea superior
        mov     cl,mmvcar
        xor     ch,ch
        rep     stosw
        mov     al,esqsd                ;dibuja esq. sup. der.
        stosw
mmv50:  inc     mmvlin                  ;proxima linea
        mov     bl,mmvlin
        cmp     bl,22                   ;linea inferior si BL=22
```

```
        je      mmv6
        posdir  mmvx1,mmvlin,mmvdir     ;escribir las suboopciones
        mov     di,mmvdir
        mov     al,linlat               ;pero antes el recuadro.
        mov     cl,mmvcar               ;long de la linea
        stosw
mmv501: lodsb                           ;no toco en cuenta los
        cmp     al,20h                  ;espacios iniciales
        je      mmv501
        dec     si
mmv51:  lodsb                           ;leo byte del menu
        stosw                           ;lo escribo en pantalla
        dec     bl
        cmp     al,20h                  ;He terminado la opcion?
        je      mmv52
        cmp     al,0                    ;He terminado el menu?
        jne     mmv51
        dec     di                      ;en ese caso borro el 00H
        dec     di                      ;que escribi
        mov     al,20h                  ; y pongo un ' '
        stosw
mmv52:  cmp     bl,0                    ;lleno el resto de la linea
        je      mmv53                   ;con espacios
        mov     al,20h
mmv521: stosw
        dec     bl
        jnz     mmv521
mmv53:  mov     al,linlat               ;y termino con recuadro.
        stosw
mmv54:  lodsb                           ;salto espacios
        cmp     al,20h
        jz      mmv54
        dec     si
        jmp     mmv50
mmv6:   posdir  mmvx1,mmvlin,mmvdir     ;dibuja linea inferior
        mov     di,mmvdir
        mov     al,esqii
        mov     ah,colven
        stosw
        mov     al,linsup
        mov     cl,mmvcar
        xor     ch,ch
        rep     stosw
        mov     al,esqid
        stosw                           ;fin de dibujo de ventana.
        cmp     y,0                     ;existe suboopcion seleccionada?
        jz      mmv7                    ;no, termina trabajo.
        mov     al,y                    ;si, resaltarla cambiando
mmv63:  mov     bl,mmvlin1              ;el atributo de la linea.
        add     al,bl
        mov     mmvlin,al
        posdir  mmvx1,mmvlin,mmvdir
```

```
        mov     di,mmvdir
        mov     ax,3
        add     di,ax
        mov     bl,mmvcar
        mov     al,colact
mmvb4:  stosb
        inc     di
        dec     bl
        jnz     mmvb4
mmv7:   traereg
        retn

mmos:
        salvareg
        pmenu   moslin,moscol   ;Presenta pregunta y espera respuesta.
        mov     si,ax
mmos0:  lodsb
        cmp     al,0
        jne     mmos0
        mov     mdirop,si
mmos1:  lodsb                   ;Obtengo el # de opciones.
        mov     cl,al
        mov     dl,al
        getkey                  ;obtengo respuesta.
        cmp     al,'a'          ;minusculas --> mayusculas.
        jc      mmos11
        cmp     al,'z'+1
        jnc     mmos11
        sub     al,20h
mmos11: mov     bx,ax
mmos2:  mov     al,[si]         ;comparo y si no es ninguna de las
        inc     si              ;posibles, vuelve a pedir respuesta.
        sub     cl,1
        jc      mmos4
        cmp     al,0
        je      mmos3
mmos3:  cmp     al,bl
        jne     mmos2
        jmp     mmos5
mmos4:  mov     si,mdirop
        jmp     mmos1
mmos5:  sub     dl,cl
        mov     mnmos,dl        ;# de respuesta en MNMOS.
        traereg
        retn

mmb:
        salvareg                ;mueve 8 bytes.
        push    ds
        pop     es
        mov     cx,8
        rep     movsb
```

```
        traereg
        retn

mnn:
        salvareg
        test    cl,1            ;pongo CH en el CL-esimo nibble
        jz      mnn1            ;del buffer apuntado por DX.
        mov     al,16
        mul     ch
        cmp     al,0
        jnz     mnn4
        mov     ch,0ffh
        mov     al,0fffh
        jmp     mnn1
mnn4:   mov     ch,al
mnn1:   shr     cl,1
        mov     bl,cl
        xor     bh,bh
        add     bx,dx
        cmp     ch,0
        jz      mnn5
        cmp     al,0fffh
        jnz     mnn2
mnn5:   and     [bx],ch
        jmp     mnn3
mnn2:   or      [bx],ch
mnn3:   traereg
        retn

mop:
        salvareg                        ;maneja opciones.
        mov     cl,nclave1              ;Primero veo de que tipo es la
        mov     al,ninfou1              ;opcion actual. (Esto es usado
        mul     cl                      ;al salir de este macro).
        add     ax,offset tabinf
        add     ax,6
        mov     si,ax
        mov     al,[si]
        mov     tipruti,al
        cmp     bmac,1                  ;Corriendo/probando macro?
        je      mop1
        jmp     mop2
mop1:   mov     al,longmac              ;si.
        inc     al
        cmp     al,contmac              ;termino?
        jne     mop10
        mov     bmac,0                  ;si,
        mov     fasepir,offset fnorm    ;fase normal
        status1
        jmp     mop2
mop10:  mov     si,offset buffmac       ;estoy en macro,
        push    ax                      ;leo dato de BUFFMAC
```

```
        mov     al,contmac
        xor     ah,ah
        mov     si,ax
        add     si,si
        mov     ax
        mov     bx,[si]
        inc     contmac
        cmp     contmac,0ffh            ;fin de ejecucion por
        je      mop12                   ;flujo ambiguo?
        cmp     over,0                  ;probando macro?
        je      mop13
mop11:  getkey                          ;si, espero ya sea
        cmp     al,cr                   ;ENTER para seguir
        je      mop13
        cmp     al,ctrlk                ;o ^K para terminar.
        jne     mop11
mop12:  mov     bmac,0                  ;termino prueba,
        mov     over,0
        mov     bx,0ff00h               ;no ejecuto ultima opcion
        mov     faseptr,offset fnorm    ;fase normal.
        mov
        status
mop13:  mov     ax,bx                   ;tomo el valor del buffer
        jmp     mop3                    ;en vez del teclado.
mop2:   getkey                          ;no es macro,dato de teclado.
        cmp     gmac,0                  ;grabando macro?
        jne     mop21
        jmp     mop3
mop21:  mov     bx,0ff00h
        cmp     contmcr,0ffh            ;si, fin macro por razones de
        je      mop22                   ;rutina de macros de usuario?
        cmp     al,ctrlk                ;si,termino de grabar?
        je      mop22
        mov     bl,contmac              ;grabo dato en BUFFMAC
        xor     bh,bh
        add     bx,bx
        add     bx,offset buffmac
        mov     ds:[bx],ax
        inc     contmac                 ;y cuento # de datos. No eje-
        cmp     contmac,maxmac          ;cuto la primera en exceso.
        je      mop23
        jmp     mop3
mop23:  mov     bx,ax
        dec     contmac                 ;excedi el limite.
        mov     ax,offset blanco
        pmenu   24,colexp
        mov     ax,offset escmcr
        pmenu   23,collact
mop231: getkey
        mov     ax,getword
        cmp     al,1bh                  ;ESC?
        jne     mop231
mop22:  mov     al,nummac               ;paso datos al area de macros.
```

```
        mov     cl,6
        mul     ci
        add     ax,offset areamac
        push    es
        push    ds
        pop     es
        mov     di,ax
        mov     al,contmac              ;long. del macro,
        mov     [di],al
        inc     di
        mov     si,offset nombmac       ;y nombre
        mov     cx,7
        rep     movsb
        mov     al,nummac               ;ahora paso BUFFMAC
        mov     cl,2*maxmac             ;al area que empieza
        mul     cl                      ;con MAC1. Cada macro
        add     ax,offset mac1          ;ocupa 2*MAXMAC bytes
        mov     di,ax                   ;o MAXMAC words
        mov     si,offset buffmac
        mov     cx,maxmac
        rep     movsw
        pop     es
        mov     opmac,0                 ;termina grabacion
        mov     fasepir,offset fnorm    ;fase normal.
        statusf
        mov     ax,bx                   ;ejecuto la ultima opcion
mop3:   cmp     al,0                    ;es un car.especial?
        je      mop32
mop31:  cmp     al,cr                   ;no, es Enter?
        jne     mop311
        mov     al,5
        jmp     mop4
mop311: cmp     al,ctrlo                ;es ^O?
        jne     mop312
        mov     al,7
        jmp     mop4
mop312: cmp     al,ctrlp                ;es ^P?
        jne     mop313
        mov     al,8
        jmp     mop4
mop313: cmp     al,ctrlk                ;es ^K?
        jne     mop314
        mov     al,9
        jmp     mop4
mop314: cmp     al,1bh                  ;es ESC?
        jne     mop315
        mov     al,10
        jmp     mop4
mop315: mov     mopaux,al               ;es otro?
        mov     al,6
        jmp     mop4
mop32:  cmp     ah,4bizq                ;es <--?
```

```
        jne     mop321
        mov     al,1
        jmp     mop4
mop321: cmp     ah,flecder              ;es --->
        jne     mop322
        mov     al,2
        jmp     mop4
mop322: cmp     ah,flecarr              ;es flecha arriba?
        jne     mop323
        mov     al,3
        jmp     mop4
mop323: cmp     ah,flecaba              ;es flecha abajo?
        jne     mop324
        mov     al,4
        jmp     mop4
mop324: mov     mopauxl,ah              ;es otro.
        mov     mopaux,0
        mov     al,6
mop4:   mov     mopact,al
        cmp     pcoc,1                  ;tengo ventana de opciones?
        jne     mop5
        restpantalla                    ;si,recupero pantalla.
mop5:   traereg
        retn

mpch:
        salvareg
        mov     ah,2            ;Uso rutina del DOS para escribir DL.
        int     21h
        traereg
        retn

mps:
        salvareg
        push    dx              ;pone un string terminado en 0
        xor     ch,ch           ;en linea CL con atributo DH.
        mov     si,ax           ;EL string es apuntado por AX.
        mov     al,longlin
        mul     cl
        mov     es,videoseg
        mov     di,ax
        pop     dx
        mov     ah,dh
        push    di
        mov     al,20h
        mov     cx,80           ;borro la linea.
        rep     stosw
        pop     di
        inc     di
        inc     di
mps0:   lodsb                   ;pongo el string.
        cmp     al,0
```

```
        je      @@n1
        stosw
        jmp     @@g0
@@n1:   mov     @@mprx,si       ;MPMPRX apunta a proximo string.
        traereg
        retn

@@d:
        salvareg
        mov     al,160          ;Direccion = CL*160 + BL.
        mul     cl
        xor     bh,bh
        add     ax,dx
        add     ax,bx
        mov     @@ddir,ax
        traereg
        retn

@@p:
        salvareg                ;Paso PANBUF a pantalla.
        xor     di,di
        mov     es,videoseg
        mov     si,offset panbuf
        mov     cx,3680
        rep     movsb
        traereg
        retn

@@f:
        salvareg
        mov     si,faseptr      ;FASEPTR tiene la direccion de un
        posdir  73,23,@@fdir    ;string de 6 bytes que debe ser
        mov     di,@@fdir       ;mostrado en la posicion (73,23).
        mov     cx,6
        mov     ah,collact
@@f1:   lodsb
        stosw
        dec     cx
        jnz     @@f1
        traereg
        retn

@@v:
        salvareg
        mov     si,valptr       ;VALPTR tiene la direccion de un string
        posdir  73,24,@@fdir    ;de 6 bytes en el que el usuario puede
        mov     di,@@fdir       ;poner algo para aparecer en (73,24).
        mov     cx,6
        mov     ah,collact
@@v1:   lodsb
        stosw
        dec     cx
```

```
        jnz     mnv1
        traereg
        retn

mv?:
        salvareg                        ;Busca si la eleccion esta
        mov     cl,nclave               ;en el menu del usuario.
        mov     al,ninfou?
        mul     cl
        add     ax,offset tabinf
        inc     ax
        inc     ax
        push    cx
        mov     bx,ax
        mov     si,[bx]                 ;SI apunta al menu
        pop     bx
mv?1:   lodsb
        cmp     al,0
        jne     mv?1
        lodsb                           ;SI apunta a primera opcion
        mov     cl,al                   ;AL tiene el # de opciones
        mov     dl,al
        pushn   si                      ;si el primer byte despues
        push    ax                      ;de las opciones validas
        mov     mv?aux1,0
        xor     ah,ah                   ;es FF, entonces evacuar
        add     si,ax                   ;las teclas no validas
        cmp     byte ptr[si],0ffh       ;por la ultima opcion.
        jne     mv?11
        mov     mv?aux1,al              ;dejo MV?AUX1 como bandera
mv?11:  pop     ax
        pop     si
        mov     al,mopaux               ;A las minusculas
        cmp     al,'a'
        jb      mv?2
        cmp     al,'z'
        ja      mv?2
        sub     al,20h                  ;las hago mayusculas.
mv?2:   mov     bl,al
mv?21:  lodsb
        cmp     al,bl                   ;veo si opcion es valida
        je      mv?3
        dec     cl
        jnz     mv?21
        cmp     mv?aux1,0               ;no fue valida, veo si hay
        je      mv?22                   ;rutina de escape
        mov     al,mv?aux1              ;si hay esa rutina
        mov     mv?aux,al               ;salgo con respuesta
        xor     ah,ah                   ;y ZF=1
        jmp     mv?4
mv?22:  or      al,1                    ;no hay esa rutina,ZF=0
        jmp     mv?4
```

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;;;;;;;;;  MACMCR.ASM  ;;;;;;;;;;;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;Macro-instrucciones para macro-comandos
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;-----------------------------------------------------------------------
;Macro  Parametros               Funcion
;-----------------------------------------------------------------------
;cerrmcr                         cierra archivo de macros
;creamcr dirnom                  crea archivo de macros
;grabmcr                         graba y cierra archivo de macros
;getkey                          lee teclado o buffer de macros
;inimcr dirnom                   abre y lee (o crea) archivo de macros
;limpmcr                         pone 0 en las areas de macros
;rtodo                           pop todo
;stodo                           push todo
;sbrmcr                          subrutinas de los otros macros
;-----------------------------------------------------------------------

;;;;;; CERRMCR ;;;;;;;;;;;;;; 17/04/89
;Cierra el archivo cuyo handle es MCRHAND.
cerrmcr macro
        call    rcmcr
        endm

;;;;;; CREAMCR ;;;;;;;;;;;;;; 10/03/89
;Crea el archivo cuyo nombre esta dado por el ASCIIZ
;que empieza en DIRNOM. Pone el handle en MCRHAND.
creamcr macro   dirnom
        push    dx
        mov     dx,offset dirnom
        call    rcmc
        pop     dx
        endm

;;;;;; GRABMCR ;;;;;;;;;;;;;; 10/03/89
;Graba en el archivo cuyo handle es MCRHAND los 1088 bytes que
;empiezan en AREAMAC y contienen el indice y datos del macro manejados
;por FLUJO; y los 2048 bytes que empiezan en MCRDAT y que componen
;el area de datos de los macros que el usuario ha dado en respuesta
;a rutinas del programa usuario de FLUJO.
;Codigos de error en AX: 0, todo bien; 1, no hay archivo abierto.
grabmcr macro
        call    rgm
        endm

;;;;;; GETKEY  ;;;;;;;;;;;;;; 10/03/89
;Es la funcion de teclado elemental intervenida para poder
;leer un dato del teclado o del buffer de macros, y poder
;grabar un dato en ese buffer si es necesario. Su salida
;es GETWORD que tiene el codigo ASCII extendido de una tecla.
;Si se estaba grabando macro y se termino el espacio para el
```

```
;macro, se pone el byte de interface en FF y MCRERR en 1.
getkey  macro
        call    rgk
        endm

;!!!!!! INIMCR  !!!!!!!!!!!!!!!!! 10/03/89
;Recibe a partir de DIRNOM un ASCIIZ con el nombre del archivo
;de macros. Trata de abrir el archivo. Si existe, lo lee en
;AREAMAC (1085 bytes) y en MCRDAT (siguientes 2048 bytes).
;Guarda el handle en MCRHAND.Codigos de error extendidos en AX.
;Si detecta que un archivo no es de macros, sale AX = FF.
;Pone valor AREAMAC-1085 en DIRNMAC. En [DIRNMAC] esta el # del
;macro actual y el siguiente byte es el de interface, que debe
;hacerse FF cuando deba terminarse la grabacion de un macro.
inimcr  macro   dirnom
        push    dx
        mov     dx,offset dirnom
        call    rim
        pop     dx
        endm

;!!!!!! LMPMCR  !!!!!!!!!!!!!!!!! 17/04/89
;Limpia las areas de macros
lmpmcr  macro
        call    rlmcr
        endm

;!!!!!! RTODO   !!!!!!!!!!!!!!!!! 10/03/89
rtodo   macro
        pop     ax
        pop     bx
        pop     cx
        pop     dx
        pop     si
        pop     di
        pop     bp
        pop     ds
        pop     es
        endm

;!!!!!! STODO   !!!!!!!!!!!!!!!!! 10/03/89
stodo   macro
        push    es
        push    ds
        push    bp
        push    di
        push    si
        push    dx
        push    cx
        push    bx
        push    ax
        endm
```

```
;!!!!!! SBRMCR !!!!!!!!!!!!!!! 10/03/89
sbrmcr  macro
        .data
mcrdat  db      2048 dup(0)     ;datos manejados por programa usuario
control db      0,0             ;Para ver si archivo es de macros (FC
dirmacr dw      0               ;direccion # de macro actual
mcrhand dw      0               ;handle de archivo de macros
getword dw      0               ;salida de GETWEW
mcrerr  db      0               ;para sacar codigo de error.

        .code
rmmcr:  stodo
        mov     bx,mcrhand      ;obtengo el handle y
        mov     ah,3eh          ;cierro el archivo
        int     21h
        rtodo
        retn

rmmc:   stodo
        mov     ah,3ch          ;creo archivo de macros.
        mov     cx,0
        int     21h
        mov     mcrhand,ax
        rtodo
        retn

rgm:    stodo
        cmp     mcrhand,0       ;habia archivo de macros abierto?
        jne     rgm1            ;si, continuo.
        mov     mcrerr,1        ;no salgo con codigo de error
        jmp     rgmf
rgm1:   mov     control,'F'
        mov     control+1,'0'
        mov     ah,42h          ;fijo puntero de R/W
        mov     cx,0            ;al inicio del archivo.
        mov     dx,0
        mov     bx,mcrhand
        mov     al,0
        int     21h
        mov     cx,1088         ;escribo primera parte del archivo
        mov     dx,offset areamac
        mov     ah,40h
        int     21h
        mov     cx,2050         ;escribo segunda parte
        mov     dx,offset mcrdat
        mov     ah,40h
        int     21h
        mov     mcrerr,0
rgmf:   rtodo
        mov     al,mcrerr
        mov     ah,0
```

```
        retn

rgk:    stodo
        mov     mcrerr,0
        mov     bx,dirnmac      ;veo si quiero entrar a macros
        cmp     byte ptr[bx+1],0ffh
        je      rgkn
        cmp     bgmac,0         ;grabando macros?
        jne     rgk2            ;si.
        cmp     bmac,0          ;no, corriendo o probando macros?
        jne     rgk1            ;si.
rgkn:   call    rgk0            ;no, rutina normal de teclado.
        jmp     rgkf
rgk1:   mov     si,offset mcrdat        ;leo dato de buffer
        mov     bx,dirnmac
        mov     dh,[bx]
        mov     dl,0            ;DX=# de macro * 256
        add     si,dx           ;SI apunta a macro actual
        mov     dl,[bx+1]
        mov     di,dl
        mov     bh,0
        add     ax,bx
        mov     ax,[si+bx]
        mov     getworo,ax      ;paso dato como leido de teclado.
        mov     bx,dirnmac
        inc     byte ptr[bx+1]
        cmp     byte ptr[bx+1],80h      ;fin?
        je      rgk3
        cmp     bver,0          ;probando?
        je      rgkf            ;no, salgo.
        push    getword         ;si,espero ENTER para confirmar.
rgk11:  call    rgk0
        mov     ax,getword      ;es ENTER?
        cmp     al,0dh
        jne     rgk11           ;no, espero un ENTER.
        pop     getword         ;si, salgo.
        jmp     rgkf
rgk2:   mov     si,offset mcrdat        ;leo dato de teclado
        mov     bx,dirnmac
        mov     dh,[bx]
        mov     dl,0            ;DX=# de macro * 256
        add     si,dx           ;SI apunta a macro actual
        mov     dl,[bx+1]
        mov     bl,dl
        mov     bh,0
        add     bx,bx
        call    rgk0            ;guardo dato en el buffer
        mov     ax,getword
        mov     [si+bx],ax
        mov     bx,dirnmac
        inc     byte ptr[bx+1]
        cmp     byte ptr[bx+1],80h      ;termino area de macros?
```

```
        jb      rgr1                    ;no.
rgr0:   mov     byte ptr[bx+1],0ffh     ;si, menor estos valores como
        mov     mcrerr,1                ;indicacion de fin de macro
rgr1:   rtodo
        retn

rgr0:   stodo
        mov     ah,0
        int     16h
        mov     getword,ax
        rtodo
        retn

rim:    stodo                   ;intento abrir el archivo
        mov     bx,offset areamac
        add     bx,1088         ;inicializo DIRNMAC
        mov     dirnmac,bx
        mov     ah,3dh
        mov     al,02h
        int     21h
        jnc     rim1            ;todo bien
        mov     ar,89h          ;error, veo de que tipo es.
        mov     bx,0
        int     21h
        mov     mcrerr,al       ;salgo con codigo de error extendido
        jmp     rim1
rim1:   mov     mcrhand,ax
        mov     bx,ax
        mov     ah,42h          ;veo si es de macros
        mov     cx,0
        mov     dx,3136
        mov     al,0
        int     21h
        mov     cx,2
        mov     bx,mcrhand
        mov     dx,offset control
        mov     ah,3fh          ;READ
        int     21h
        cmp     control,'F'
        jne     rim11
        cmp     control+1,'0'
        je      rim12
rim11:  mov     ah,3eh          ;no paso chequeo
        mov     bx,mcrhand      ;cierro archivo
        int     21h
        mov     mcrerr,0ffh
        jmp     rim1
rim12:  mov     bx,mcrhand      ;es de macros
        mov     ah,42h          ;LSEEK
        mov     cx,0
        mov     dx,0
        mov     al,0
```

```
;!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
;!!!!!!!!!!     RUTVNT.ASM      !!!!!!!!!!
;!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
;!!  Rutinas para manejo de ventanas  !!
;!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
;-------------------------------------------------------------------------
;Macro  Parametros              Funcion
;-------------------------------------------------------------------------
;hexasc                         hex en AL ==  ascii en BX
;idvnt  dirasc,dirres,longres   identifica archivo y area de ventanas
;popall                         pop a todo
;posdir x,y                     calcula direccion en memoria de video
;pushall                        push a todo
;vnt    nombre,tipo             muestra o actualiza una ventana
;sbrvnt                         subrutinas de los otros macros
;-------------------------------------------------------------------------

;!!!!!! HEXASC  !!!!!!!!!!!!!!! 10/03/89
;El byte en AL se convierte en dos bytes ASCII en BX.
hexasc  macro
        call    rha
        endm

;!!!!!! IDVNT   !!!!!!!!!!!!!!! 08/03/89
;Este macro recibe el nombre del archivo de ventanas en un ASCIIZ que
;inicia en DIRASC. Abre el archivo, trae todos los grupos principales
;al area que empieza en DIRRES y guarda DIRRES y LONGRES en el indice.
;En AX salen codigos de error: 00 todo bien, 01 error al abrir archivo,
;02 si se encuentra que no es un archivo de ventanas, y
;03 error por falta de espacio (LONGRES= 6 grupos para indice + # de
;grupos principales + 1 grupo para una tabla de residencia actual + al
;menos 4 grupos para secundarios, a menos que todas las ventanas sean
;principales). Nota: Un grupo=256 bytes.
idvnt   macro   dirasc,dirres,longres
        push    dx
        push    di
        push    bx
        mov     dx,offset dirasc
        mov     di,offset dirres
        mov     bl,longres
        call    riv
        pop     bx
        pop     di
        pop     dx
        endm

;!!!!!! POPALL  !!!!!!!!!!!!!!! 10/03/89
popall  macro
        pop     ax
        pop     bx
        pop     cx
        pop     dx
```

```
        pop     si
        pop     di
        pop     bp
        pop     ds
        pop     es
        endm

;;;;;; POSDIR  ;;;;;;;;;;;;;;;; 09/07/89
;Convierte la posicion (X,Y) en una direccion en
;PUNTER para acceso directo a la memoria de video.
posdir  macro   x,y
        push    cx
        mov     cl,x
        mov     ch,y
        call    rpd
        pop     cx
        endm

;;;;;; PUSHALL ;;;;;;;;;;;;;;;; 10/03/89
pushall macro
        push    es
        push    ds
        push    dx
        push    di
        push    si
        push    dx
        push    cx
        push    bx
        push    ax
        endm

;;;;;; VNT     ;;;;;;;;;;;;;;;; 08/03/89
;BX apunta al nombre de la ventana. Si existe y es principal, sus
;grupos en el area de residentes son los mismos que tenia en el
;archivo. Si es secundaria ve si es residente, caso en que sus grupos
;estan en la entrada en el indice. Si no es residente, se la trae al
;area de residentes y se la marca como tal, borrando la residencia
;de las ventanas sobre las que esta ventana se coloque. Muestra la
;ventana y donde se encuentren variables se las leen del area apuntada
;por AX:SI. Existen dos TIPOs de llenado: 'h', hexadecimal, c/byte
;se muestra en dos variables de la ventana y 'a', ASCII, c/byte se
;muestra como tal en pantalla.
;En AX sale un codigo de error: 0, todo bien; 1, nombre no existe.
vnt     macro   tipo
        mov     tipvnt,tipo
        call    rvnt
        endm

;;;;;; SBRVNT  ;;;;;;;;;;;;;;;; ../03/89
sbrvnt  macro

tabha   db      '0','1','2','3','4','5','6','7','8','9'
```

```
        jnz     riv11
        cmp     al,[di+bx]
        je      riv12
        mov     ax,2            ;paridad de ventanas no se cumple
        jmp     riv4
riv12:  mov     bx,3e
        mov     cl,22
        mov     al,0
riv13:  add     al,[di+bx]
        inc     bx
        dec     cl
        jnz     riv13
        cmp     al,[di+bx]
        je      riv2
        mov     ax,2            ;paridad de grupos no se cumple
        jmp     riv4
riv2:   mov     al,[di+9]       ;# de grupos principales
        cmp     al,[di+1]       ;= total de grupos?
        je      riv21
        add     al,4            ;+4 grupos minimo de secundarias
riv21:  mov     al,6            ;+6 de indice
        inc     al              ;+1 para tabla de residencia
        cmp     al,[di+12]
        jle     riv3
        mov     ax,3            ;muy poco espacio
        jmp     riv4
riv3:   mov     dx,di           ;todo bien, leo resto del indice
        add     dx,72
        mov     cx,1536-72
        mov     bx,[di+13]      ;handle
        mov     ah,3fh
        int     21h
riv4:   mov     dx,di           ;leo todos los principales
        add     dx,1536
        mov     ch,[di+9]       ;CX=# de grupos principales*256
        mov     cl,0            ;= # de bytes por leer.
        mov     ah,3fh
        int     21h
        mov     ch,[di+9]       ;preparo numero de proximo grupo
        add     ch,6            ;a usarse de area de residentes
        mov     [di+15],ch
        mov     ch,[di+8]       ;copio los # de grupos en el archivo
        add     di,48           ;a los grupos en memoria para
        mov     bp,di           ;las CH ventanas principales
riv5:   add     bp,24           ;y las marco como residentes
        mov     di,bp
        mov     byte ptr[di+8],3
        mov     bx,[di+16]
        mov     [di+20],bx
        mov     bx,[di+18]
        mov     [di+22],bx
        dec     ch
```

```
        jnz     rtv2
        mov     ax,0
rtv1:   pop     cx
        pop     si
        retn

pddir   dw      0
rdd:    push    ax
        push    cx
        mov     al,160          ;80i2 bytes por linea
        mul     ch
        add     cl,cl           ;+ 2 veces el # de columna
        mov     ch,0
        add     ax,cx
        mov     pddir,ax
        pop     cx
        pop     ax
        retn

segvar  dw      0               ;segmento donde estan las variables
offvar  dw      0               ;offset de la variable actual
tipvnt  db      0               ;tipo: 'h', 'a' o 't'
dirent  dw      0               ;direccion de entrada de vent. actual
numgrp  db      0               ;# de grupos que faltan por traer
columna db      0               ;columna escrita actualmente
linea   db      0               ;linea escrita actualmente
numbyte dw      0               ;# de byte mostrado del grupo actual

rvnt:   pushall
        mov     segvar,ax
        mov     offvar,si
        mov     di,dirind       ;direccion del indice
        mov     dl,[di+2]       ;ultima ventana valida
        add     di,48           ;voy a buscar el nombre en las entradas
        push    es              ;del indice.
        push    ds
        pop     es
        mov     dh,2
        mov     bp,di
rvnt1:  add     bp,24
        mov     di,bp           ;ubico entrada indicada por
        inc     dh              ;DH
        mov     si,bx
        mov     cx,9            ;nombre de 8 bytes
rvnt10: mov     ah,es:[di]      ;leo 1 byte del nombre de una ventana
        inc     di
        lodsb                   ;leo 1 byte del nombre buscado
        dec     cx
        jz      rvnt11          ;nombres iguales?
        cmp     al,ah
        jne     rvn10           ;nombres diferentes
        cmp     al,0            ;iguales?
```

```
        je      rvnt11
        jmp     rvnt10
rvnt10: cmp     dh,dl           ;busco en todas las ventanas?
        jne     rvnt1           ;no, sigo.
        pop     es              ;si, nombre no existe,
        mov     numbyte,1       ;error = 1
        jmp     rvnt5
rvnt11: pop     es
        mov     si,dirind
        mov     [di+6],dh       ;guardo # ventana actual
        mov     al,24
        mul     dh              ;ubico entrada actual y
        add     di,ax           ;DS:DI apunta a ventana actual
        mov     dirent,di
        mov     dh,[di+8]       ;veo si es P o S.
        cmp     dh,3
        jne     rvnt2           ;S.
        jmp     rvnt4           ;P.
rvnt2:  cmp     dh,2            ;secundaria. Reside?
        jne     rvnt20
        jmp     rvnt4           ;si.
rvnt20: mov     dl,[di+9]       ;no, ubicar la ventana.
        mov     byte ptr[di+8],2        ;mov residencia a la ventana
        mov     si,dirind       ;DS:SI apunta al indice
        mov     ah,[si+12]      ;ubico tabla de residencia
        dec     ah              ;AX tiene el offset de la tabla
        mov     al,0            ;respecto del inicio del indice
        mov     bp,si
        add     bp,ax           ;DS:BP apunta a la tabla
        mov     cl,[si+15]      ;CL = # grupo que va a ser dado
        mov     bx,16           ;a la ventana actual
rvnt21: cmp     byte ptr[di+bx],0       ;termino ventana?
        je      rvnt3           ;si.
        mov     [di+bx+4],cl
        push    bp              ;no, leer valor en tabla
        add     bp,cx
        mov     al,ds:[bp]
        cmp     al,0            ;grupo estaba libre?
        je      rvnt22          ;si.
        push    si              ;no, quito residencia a la ventana
        mov     ah,24
        mul     ah
        add     si,ax
        mov     byte ptr[si+8],0
        pop     si
rvnt22: mov     al,[si+6]       ;ventana actual
        mov     ds:[bp],al      ;a la tabla de residencia
        pop     bp
        inc     cl              ;preparo proximo grupo a usarse
        mov     ah,[si+12]
        sub     ah,2
        cmp     cl,ah
```

```
        jle     rvnt23
        mov     ah,[si+9]       ;termino area de residentes
        add     ah,6            ;==> voy a inicio de area de
        mov     cl,ah           ;secundarias
rvnt23: mov     [si+15],cl
        inc     bl              ;ya revise cuatro grupos?
        cmp     bl,20
        jne     rvnt2           ;no, volver.
rvnt3:  mov     bx,ie
rvnt31: mov     dh,[di+bx]      ;leo # de grupo
        cmp     dh,0            ;termino?
        je      rvnt4           ;si.
        mov     di,0            ;no. DX:DI=offset desde
        mov     cx,0            ;inicio de archivo
        mov     al,0
        mov     ah,42h          ;LSEEK
        push    bx
        mov     bx,[si+13]
        int     21h
        pop     bx
        mov     dh,[di+bx+4]    ;DS:DX apunta a buffer de lectura
        mov     di,0            ;Leo un grupo del disco
        add     bx,si
        mov     cx,256
        mov     ah,3fh          ;READ
        push    bx
        mov     bx,[si+13]
        int     21h
        pop     bx
        inc     bl
        cmp     bl,20           ;ya leyo 4 grupos?
        jne     rvnt31          ;no, sigo leyendo.
rvnt4:  mov     ax,[di+12]      ;esq. sup. izq.
        mov     linea,ah
        mov     columna,al
        mov     al,[di+9]       ;# de grupos
        mov     numgrp,al
        mov     numbyte,0       ;# de byte actual
        mov     si,dirind
        mov     ah,[di+20]      ;AX posicion primer grupo
        mov     bp,di
        mov     al,0
        add     si,ax           ;SI apunta al grupo
        posdir  columna,linea
        mov     di,pddir
rvnt41: lodsw
        cmp     al,1            ;variable?
        je      rvnt42          ;si.
        stosw                   ;no, mostrar caracter.
        jmp     rvnt44
rvnt42: push    ds
        mov     bx,offvar       ;leo una variable
```

```
        mov     ds,segvar
        mov     al,[bx]
        pop     ds
        inc     offvar
        cmp     tipvnt,'h'      ;ventana hexadecimal?
        je      rvnt43          ;si.
        stosw                   ;no, ventana ascii.
        jmp     rvnt44
rvnt43: hexasc                  ;AL se convierte en ascii en BX
        mov     al,bh
        stosw                   ;muestro mitad mas significativa.
        loosw                   ;Leo atributo siguiente.Supongo
        mov     al,bl           ;que AL=1 (otra variable) y
        stosw                   ;muestro la otra mitad.
        inc     columna
        mov     dx,si           ;Con DX vere si hubo cambio de grupo.
        call    rvnt0           ;ve si grupo actual ya termino.
        cmp     dx,si           ;Hubo cambio de grupo?
        je      rvn431          ;no, SI apunta correctamente.
        add     si,2            ;si, hago que SI apunte bien.
rvn431: cmp     al,1            ;termino la ventana?
        je      rvnt5           ;si, salir.
rvnt44: call    rvnt0           ;no, seguir.
        cmp     al,1            ;termino ventana?
        je      rvnt5           ;si.
        inc     columna
        mov     al,ds:[bp+14]   ;termino linea?
        cmp     al,columna
        jge     rvnt41          ;no, continuar.
        inc     linea           ;si, nueva linea.
        mov     al,ds:[bp+12]   ;primera columna.
        mov     columna,al
        posdir  columna,linea
        mov     di,pddir
        mov     al,ds:[bp+15]   ;termino ventana?
        cmp     al,linea
        jb      rvnt5
        jmp     rvnt41          ;no, continuar.
rvnt5:  mov     numbyte,0       ;si, salir.
rvntf:  popall
        mov     ax,numbyte
        retn
rvnt0:  push    bp              ;varia contadores y ve si es necesario
        add     numbyte,2       ;empezar otro grupo. Altera AX.
        cmp     numbyte,100h    ;termino un grupo?
        jb      rvnt02          ;no.
        dec     numgrp          ;si, un grupo menos por mostrar
        cmp     numgrp,0        ;termino?
        jne     rvnt01          ;no.
        mov     al,1            ;si, salgo con AL=1.
        jmp     rvnt03
rvnt01: mov     ah,numgrp       ;leo un # de grupo
```

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;      MACPRG.ASM        ;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;-----------------------------------------------------------------------
;Macro  Parametros                Funcion
;-----------------------------------------------------------------------
;actuar                           salta a la AL-esima dir. de una tabla
;finmacro                         termina macro y da un mensaje
;getchar                          lee un ASCII
;getdata x,y,buff,tipo            lee un bin, hex o dec
;getline buff                     lee un string del teclado
;limpln pos,atrib                 borra una linea
;limppnt                          borra pantalla
;print  x,y,dir,atrib             pone un texto en pantalla
;rest23                           ;restaura linea 23
;salva23                          ;salva linea 23
;usrpro                           intercambia pantallas de usuario y PRO
;sbrpro                           subrutinas de los otros macros
;-----------------------------------------------------------------------

;;;;;;; ACTUAR  ;;;;;;;;;;;;;;;; 30/03/89
;Lee el AL-esimo elemento de la tabla TABACT y salta a la direccion
;dada por ese elemento. AL=0 para el primer elemento de la tabla.
actuar  macro
        local   tabact
        push    si
        mov     ah,0
        add     ax,ax
        mov     si,offset tabact
        add     si,ax
        mov     ax,cs:[si]
        pop     si
        jmp     ax
tabact:
        endm

;;;;;;; FINMACRO ;;;;;;;;;;;;;;;; 20/04/89
;Da un mensaje de fin de macro y espera un ESC para seguir
finmacro macro
        call    rfinm
        endm

;;;;;;; GETCHAR ;;;;;;;;;;;;;;;;; 23/03/89
;Lee en GCASCII un ASCII del teclado o buffer de macros. En GCCOD
;sale el siguiente codigo: 0, hay dato valido en GCASCII. 1, ENTER. 2,
;TAB. 3, SHIFT TAB. 4, ESC. 5, alguna funcion o tecla especial, cuyo
;ASCII extendido sale en GCASCII. Para que ENTER, TAB, SHIFT TAB y
;ESC sean aceptados como datos normales, se debe presionar antes F10.
getchar macro
        call    rgch
        endm
```

```
;;;;;;; GETDATA ;;;;;;;;;;;;;;;; 23/03/89
;Lee datos del teclado o buffer de macros en BUFF. En BUFF se tiene:
;# de bytes pedidos, # de datos leidos, datos. # convertido en hex
;(4 bytes). Hace eco en pantalla en (X,Y). De acuerdo al TIPO
;recibe datos en bin (0 a 1), hex (0 a F), dec (0 a 9). La longitud de
;BUFF debe ser adecuada: bin, cualquiera; hex, 2 o 4 bytes; dec, 2
;o 5 bytes. Si se presiona DEL o BS se borra el ultimo dato ingresado.
;ENTER es esperado para confirmar un dato y convertirlo en hex. En
;GCCOD sale un codigo si ocurre lo siguiente: 0, hay un dato correcto al
;final de BUFF. 1, flecha arriba. 2, flecha abajo. 3, flecha derecha.
;4, flecha izquierda. 5, TAB. 6, SHIFT TAB. 7, ESC. 8, PGUP. 9 PGDN.
;Cualquier otro dato es procesado para ver si es valido incluirlo en el
;buffer. Se produce un BEEP si un dato es invalido o al llegar a los
;limites. Si al ingreso a la subrutina GDRESET es 1, buffer y variables
;son inicializadas. Si es 0, no lo son, y se puede seguir recibiendo
;el mismo dato. Los numeros binarios no son convertidos en hex.
getdata macro   x,y,buff,tipo
        push    dx
        push    cx
        push    cx
        mov     dx,offset buff
        mov     cl,x
        mov     ch,y
        mov     dl,tipo
        call    rgd
        pop     dx
        pop     cx
        pop     bx
        endm
        -

;;;;;;; GETLINE ;;;;;;;;;;;;;;;; 10/04/89
;Lee del teclado un buffer BUFF (ASCII), que tiene este formato: Numero
;de bytes pedidos, numero de bytes realmente ingresados, texto. Para
;edicion se acepta Backspace o Delete. ENTER para terminar el texto.
;En GCCOD sale un codigo: 1, se digito ENTER (buffer valido). 2, TAB.
;3, SHIFT TAB. 4, ESC y 5, especial. Este macro usa GETCHAR para
;recibir datos. En  X, Y recibe la posicion en que debe hacer eco.
getline macro   x,y,buff
        push    si
        push    cx
        mov     cl,x
        mov     ch,y
        mov     si,offset buff
        call    rgl
        pop     cx
        pop     si
        endm

;;;;;;; LIMPLN  ;;;;;;;;;;;;;;;; 28/03/89
;Borra la linea NUM y pone en ella el atributo ATRIB
limpln  macro   num,atrib
        push    ax
```

```
        mov     al,num
        mov     ah,atrib
        call    rll
        pop     ax
        endm

;;;;;;; LMPPNT ;;;;;;;;;;;;;;; 15/04/89
;borra la pantalla
lmpant  macro
        push    cx
        push    di
        push    ax
        mov     cx,2000
        mov     di,0
        mov     ax,0720h
        rep     stosw
        pop     ax
        pop     di
        pop     cx
        endm

;;;;;;; PRINT  ;;;;;;;;;;;;;;;; 28/03/89
;Escribe el texto apuntado por DIR en la pantalla a partir de
;(X,Y) con atributo ATRIB. El texto debe terminar con un 0.
print   macro   x,y,dir,atrib
        posdir  x,y
        push    si
        push    ax
        mov     si,offset dir
        mov     ah,atrib
        call    rprt
        pop     ax
        pop     si
        endm

;;;;;;; SALVA23 ;;;;;;;;;;;;;;; 21/04/89
salva23 macro
        call    rsal23
        endm

;;;;;;; REST23  ;;;;;;;;;;;;;;;; 21/04/89
rest23  macro
        call    rres23
        endm

;;;;;;; USRPRO  ;;;;;;;;;;;;;;;; 09/04/89
;Intercambia las pantallas de usuario y del programa
usrpro  macro
        call    rusrp
        endm

;;;;;;; SBRPRO  ;;;;;;;;;;;;;;;; ../03/89
```

```
sbrpro  macro
        .data
qccod   db      0               ;codigo de tecla especial oprimida
qcreset db      1               ;GETDATA inicializa variables?
qcascii db      0               ;resultado de GETCHAR
qctab   db      '0123456789ABCDEFabcdef'
pant    dw      2000 dup(0720h) ;buffers para guardar pantallas (USRPRO)
pantr   dw      2000 dup(0720h)
pant?   db      1               ;PANT tiene pantalla si PANT?=1
rglx    db      0               ;x,y para getline
rgly    db      0
finbufmcr db    'Fin del buffer y fin del macro. Presiona ESC.',0
buf23   dw      80 dup(0)       ;para guardar la linea 23

        .code

rfinm:  stodo
        salva23                 ;salvo linea 23
        limpln  23,revers
        print   1,23,finbufmcr,revers
rfinm1: getkey                  ;recibo ESC para salir
        mov     ax,getword
        cmp     al,1bh
        jne     rfinm1
        rest23                  ;restauro linea 23
        rtodo
        retn

rgch:   push    ax
        push    dx
rgch1:  mov     dl,1
        getkey                  ;leo dato de teclado o buffer
        cmp     mcrerr,1        ;error en macro en GETKEY?
        jne     rgch11
        finmacro
        jmp     rgch1           ;si.
rgch11: mov     ax,getword
        cmp     al,0            ;es tecla especial?
        je      rgch3           ;si.
rgch2:  cmp     al,0dh          ;ENTER?
        je      rgch4
        inc     dl
        cmp     al,9            ;TAB?
        je      rgch4
        inc     dl
        inc     dl
        cmp     al,1bh          ;ESC?
        je      rgch4
        mov     dl,0            ;es un ASCII valido.
        mov     dh,al
        jmp     rgch4
rgch3:  mov     dl,5
```

```
        mov     dh,al
        cmp     ah,68           ;F10? 44h
        je      rgch31          ;si.
        cmp     ah,15           ;SHIFT TAB? 0Fh
        jne     rgch4
        mov     dl,3
        jmp     rgch4
rgch31: mov     ci,0
        getkey
        cmp     mcrerr,1        ;error?
        jne     rgch32
        finmacro
        jmp     rgch21
rgch32: mov     ax,getword
        mov     dh,al
        cmp     al,1bh          ;ESC? 1bh
        je      rgch4
        cmp     al,0dh          ;ENTER? 0dh
        je      rgch4
        cmp     al,9            ;TAB?  9h
        je      rgch4
        jmp     rgch1
rgch4:  mov     gccod,dl
        mov     gcascii,dh
        pop     dx
        pop     ax
        retn

rgd:    stodo
        cmp     gdreset,1       ;inicializo variables?
        jne     rgd2            ;no.
        mov     dh,[bx]         ;# de datos pedidos.
        mov     al,0
        mov     [bx+1],al
        mov     si,2
        mov     al,'0'
rgd1:   mov     [si+bx],al      ;inicializo buffer
        inc     si
        dec     dh
        jnz     rgd1
rgd2:   getkey
        mov     gccod,0
        cmp     mcrerr,1        ;error en macros?
        jne     rgd21
        finmacro
        jmp     rgd2
rgd21:  mov     ax,getword
        cmp     al,0            ;tecla especial?
        jne     rgd4
rgd3:   mov     gccod,1
        cmp     ah,72           ;flecha arriba? 48h
        je      rgd34
```

```
        inc     gccod
        cmp     ah,80           ;flecha abajo?
        je      rgd3f
        inc     gccod
        cmp     ah,77           ;->?
        je      rgd3f
        inc     gccod
        cmp     ah,75           ;<-?
        je      rgd3f
        add     gccod,2
        cmp     ah,15           ;SHIFT TAB?
        je      rgd3f
        add     gccod,2
        cmp     ah,73           ;PGUP?
        je      rgd3f
        inc     gccod
        cmp     ah,81           ;PGDN?
        je      rgd3f
        cmp     ah,83           ;DEL?
        jne     rgd2
        jmp     rgdb
rgd3f:  jmp     rgdf
rgd4:   cmp     al,8            ;BS?
        jne     rgd411
        jmp     rgdd
rgd411: cmp     al,0dh          ;ENTER?
        jne     rgd412
        jmp     rgde
rgd412: cmp     al,9            ;TAB?
        jne     rgd413
        mov     gccod,5
        jmp     rgdf
rgd413: cmp     al,1bh          ;ESC?
        jne     rgd42
        mov     gccod,7
        jmp     rgdf
rgd42:  cmp     dl,'b'          ;binario?
        jne     rgd44
rgd43:  cmp     al,'0'          ;es '0' o '1'?
        je      rgd431
        cmp     al,'1'
        je      rgd431
        jmp     rgdb            ;dato no valido, beep.
rgd431: jmp     rgd46
rgd44:  cmp     dl,'h'          ;hex?
        jne     rgd45
        mov     dh,0            ;veo si dato es correcto.
        mov     si,offset gdtab
rgd441: cmp     al,[si]         ;consta en la tabla?
        je      rgd442
        inc     si
        inc     dh
```

```
        cmp     dh,22           ;acabo la tabla?
        jne     rgd441
        jmp     rgdb            ;no fue valido.
rgd442: jmp     rgd46           ; access = ABCDEF
rgd45:  mov     dh,0            ;decimal. Veo si dato es correcto.
        mov     si,offset gdtab
rgd451: cmp     al,[si]         ;consta en la tabla?
        je      rgd452
        inc     si
        inc     dh
        cmp     dh,10           ;acabo la tabla?
        jne     rgd451
        jmp     rgdb            ;no fue valido.
rgd452: mov     al,[bx]
        cmp     al,[bx+1]       ;buffer lleno?
        jne     rgd4521
        jmp     rgdb
rgd4521: push   dx
        push    cx
        push    dx
        mov     si,dx
        add     si,3
        cmp     byte ptr[bx],3  ;3 bytes (0-255)?
        je      rgd455
rgd453: mov     ax,10000
        mov     cl,[si]
        mov     ch,0
        sub     cl,30h
        mul     cx
        cmp     dx,0            ;<FFFF?
        je      rgd454
        jmp     rgd45o          ;overflow, beep.
rgd454: mov     di,ax
        inc     si
        mov     ax,1000
        mov     cl,[si]
        sub     cl,30h
        mul     cx
        add     di,ax
        jnc     rgd455
        jmp     rgd45o          ;overflow
rgd455: inc     si
        mov     ax,100
        mov     cl,[si]
        sub     cl,30h
        mul     cx
        add     di,ax
        jnc     rgd4551
        jmp     rgd45o          ;overflow
rgd4551: cmp    byte ptr[bx],3
        jne     rgd456
        cmp     ah,0
```

```
        je      rgd45a
        jmp     rgd45b          ;overflow
rgd456: inc     si
        mov     ax,10
        mov     cl,[si]
        sub     cl,30h
        mul     cx
        add     di,ax
        jnc     rgd4561
        jmp     rgd45b          ;overflow
rgd4561: cmp    byte ptr[bx],2
        jne     rgd457
        cmp     ah,0
        je      rgd457
        jmp     rgd45b          ;overflow
rgd457: pop     dx
        mov     cl,dh
        push    dx
        add     di,cx
        jnc     rgd4571
        jmp     rgd45b          ;overflow
rgd4571: cmp    byte ptr[bx],3
        jne     rgd45f
        cmp     ah,0
        je      rgd45f
rgd45b: pop     dx              ;overflow
        pop     cx
        pop     cx
        jmp     rgdb
rgd45f: inc     si
        mov     [si],di
        pop     dx
        pop     cx
        pop     dx
rgd46:  mov     ax,getword
        mov     dh,al
        call    rgd0            ;DH=#, BX-->inicio buffer
        cmp     al,1            ;exceso de datos?
        jne     rgd5
rgdb:   push    dx              ;BEEP
        mov     ah,2
        mov     dl,7
        int     21h
        pop     dx
        jmp     rgd2
rgd5:   call    rgd01
        jmp     rgd2
rgdd:   cmp     byte ptr[bx+1],0        ;DELETE. Buffer vacio?
        je      rgdb
        call    rgd02
        call    rgd01
        jmp     rgd2
```

```
rgd2:   cmp     d.,h            ;hex?
        je      rgd0            ;si los binarios no los transformo y
        jmp     rgd1            ;los dec. llegan ya transformados.
rgd2:   mov     si,dx           ;convierto en hex.
        mov     d.,[bx]
        mov     dh,0
        add     si,2
        add     si,dx
        cmp     dl,2
        je      rgd62
rgd61:  mov     al,[bx+2]       ;convierto MSB
        call    rgd03
        mov     cl,4
        shl     dh,cl
        mov     [si+1],dh
        mov     al,[bx+3]
        call    rgd03
        add     [si+1],dh       ;MSB transformado
        add     bx,2
rgd62:  mov     al,[bx+2]       ;convierto LSB
        call    rgd03
        mov     cl,4
        shl     dr,cl
        mov     [si],dh
        mov     al,[bx+3]
        call    rgd03
        add     [si],dh
rgd1:   rtodo
        retn

rgd0:   push    bx
        push    cx
        mov     al,[bx]         ;pone DH en el buffer apunt. por BX
        cmp     al,[bx+1]       ;ya no hay espacio?
        mov     al,1
        je      rgd0f
        inc     byte ptr[bx+1]  ;incremento contador
        mov     si,bx
        mov     bx,1
        cmp     byte ptr[si],1
        je      rgd0n
rgd01:  mov     al,[bx+si+2]    ;rota a la izquierda n-1 veces
        mov     [bx+si+1],al    ;donde n=# de datos pedidos.
        inc     bl
        cmp     bl,[si]
        jb      rgd01
rgd0n:  mov     [bx+si+1],dh    ;incluyo nuevo dato.
        mov     al,0
rgd0f:  pop     cx
        pop     bx
        retn
rgd01:  push    cx              ;presento buffer en pantalla
```

```
        push    dx
        posdir  cl,cn
        mov     di,pddir
        rcv     bp,di
        mov     cl,[bx]
        mov     ch,0
        rcv     al, _'          ;primero borro
rgd011: stosb
        inc     di
        loop    rgd011
        mov     al,[bx+1]
        mov     cl,[bx]
        sub     cl,al
        mov     si,bx
        add     si,2
        add     si,cx
        mov     di,bp
        add     cx,cx
        add     di,cx
        mov     cl,[bx+1]
        cmp     cl,0
        je      rgd013
rgd012: lodsb                   ;muestro el numero
        stosb
        inc     di
        loop    rgd012
rgd013: pop     dx
        pop     cx
        retn
rgd02:  push    bx              ;rota a la derecha
        dec     byte ptr[bx+1]  ;disminuyo contador
        mov     si,bx
        mov     bl,[si]
        mov     bh,0
        cmp     bl,1
        je      rgd022
rgd021: mov     al,[bx+si]
        mov     [bx+si+1],al
        dec     bx
        cmp     bl,1
        jne     rgd021
rgd022: mov     byte ptr[bx+si+1],'0'
        pop     bx
        retn
rgd03:  mov     di,offset pdtab ;Busco AL en EDTAB
        mov     dh,0            ;y traigo ubicacion en DX.
rgd031: cmp     al,[di]
        je      rgd032
        inc     di
        inc     dh
        cmp     dh,22
        jne     rgd031
```

```
rgd032: cmp     dh,16
        jb      rgd034
        sub     dh,6
rgd034: retn

rgl:    push    di              ;recibo un string desde teclado
        push    cx              ;El puerto al inicio del buffer
        push    dx
        mov     rglx,cl
        mov     rgly,ch
        cmp     gdreset,1       ;inicializo el buffer?
        jne     rgla
        mov     di,si           ;limpia el buffer
        inc     di
        mov     cl,[si]
        mov     ch,0            ;CX=# car.
        inc     cx
        mov     ax,ds
        mov     es,ax
        mov     al,0
        rep     stosb
        mov     es,videoseg
rgla:   mov     dh,y
rgl1:   mov     cl,[si+1]
        call    curs24          ;pongo cursor
        getchar                 ;recibo un dato del teclado
        mov     al,atrprev      ;quito cursor
        mov     di,dirprev
        stosb
        mov     cl,[si]         ;num. maximo de caracteres
        mov     bl,[si+1]       ;num. actual de caracteres
        cmp     gccod,5         ;car. especial?
        je      rgl2
        cmp     gccod,1         ;ENTER?
        je      rgl3
        cmp     gccod,0         ;ascii normal?
        je      rgl11
        jmp     rgl3            ;TAB, SHIFT TAB y ESC: salir.
rgl11:  mov     al,gcascii
        cmp     al,8            ;Backspace?
        je      rgld
        cmp     bl,cl           ;fin del buffer?
        je      rglb
        mov     [bx+si+2],al    ;no, guardo dato
        mov     cl,bl           ;eco a pantalla
        add     ci,rglx
        mov     ch,rgly
        posdir  cl,ch
        mov     di,pddir
        mov     al,gcascii
        stosb
        inc     byte ptr[si+1]
```

```
        jmp     rgl1
rglb:   mov     dl,7            ;BEEP
        mov     ah,2
        int     21h
        jmp     rgl1
rgl2:   cmp     gcascii,83      ;DELETE?
        je      rgld
        jmp     rgl3
rgld:   cmp     bl,0            ;puedo seguir borrando?
        je      rglb            ;no.beep
        dec     byte ptr[si+1]
        mov     byte ptr[bx+si+1],0
        mov     cl,bl           ;borro en pantalla
        add     cl,rglx
        dec     cl
        mov     ch,rgly
        posdir  cl,ch
        mov     di,pddir
        mov     al,' '
        stosb
        jmp     rgl1
rgl3:   pop     bx
        pop     cx
        pop     di
        retn

curs24: push    cx
        push    di
        add     cl,rglx
        mov     ch,rgly         ;encuentro posicion del cursor
        posdir  cl,ch           ;altero AX,DI
        mov     di,pddir
        inc     di
        mov     ah,es:[di]
        mov     atrprev,ah
        mov     dirprev,di
        mov     al,revers
        stosb
        pop     di
        pop     cx
        retn

rll:    push    cx
        push    di
        posdir  0,al
        mov     di,pddir        ;dir de inicio de linea
        mov     al,' '          ;AL tiene el ' ' y AH el atributo
        mov     cx,80
        rep     stosw           ;borra linea
        pop     di
        pop     cx
        retn
```

```
rlmo:   push    di
        push    ax
        push    cx
        mov     di,0
        mov     cx,2000
        mov     ax,0720h        ;car: espacio, atributo: normal.
        rep     stosw           ;borra pantalla.
        pop     cx
        pop     ax
        pop     di
        retn

rprt:   push    di
        mov     di,pddir        ;direccion de inicio
rprt1:  lodsb
        cmp     al,0            ;termino?
        je      rprt2
        stosw                   ;AL tiene el ASCII, AX el atributo
        jmp     rprt1
rprt2:  pop     di
        retn

real23: push    es
        push    ds
        pop     es
        mov     ds,videoseg
        mov     si,23*160
        mov     di,offset buf23
        mov     cx,80
        rep     movsw
        mov     ax,@data
        mov     ds,ax
        mov     es,videoseg
        pop     es
        retn

rres23: mov     si,offset buf23
        mov     di,23*160
        mov     cx,80
        rep     movsw
        retn

rusrp:  push    si              ;Si PANT?=1, la pantalla a mostrarse esta en
        push    di              ;PANT, sino en PANTR. Uno de estos buffers
        push    cx              ;es cambiado con la memoria de video.
        mov     si,0
        cmp     pant?,1         ;datos actuales en PANT  ?
        jne     rusrp1
        mov     di,offset pantr ;si
        jmp     rusrp2
rusrp1: mov     di,offset pant  ;no, datos actuales en PANTR
```

```
        cosseg

        .model  small

        .stack  200h

        .data

        extrn   tabinf:near,tabbuf:near
        public  dirret,mraux2,mraux3,videoseg,valptr
        public  mopaux,mopaux1,tiorut,otropux,colven
        public  areamac,bmac,bgmac,over


bopc    db      0               ;bandera para mostrar suboopciones
bmac    db      0               ;bandera de ejec. de macros
bver    db      0               ;bandera de comprob. de macros
bgmac   db      0               ;bandera de grab. de macros
nivel   db      0               ;nivel del menu actual
nivel1  db      0               ;nivel de la opcion actual
nivel2  db      0               ;nivel de la subopcion actual
resnivel db     0
buffer  db      9 dup(0)        ;indica camino hacia menu actual
buffer1 db      8 dup(0)        ;camino hacia opcion actual
buffer2 db      8 dup(0)        ;camino hacia subopcion actual
respbuf db      8 dup(0)
nclave  db      0               ;ubicacion de BUFFER's en TABBUF
nclave1 db      0
nclave2 db      0
x       db      0               ;X de opcion actual
y       db      0               ;Y de subopcion actual
xprev   db      0               ;X que origino menu actual
acabar? db      'Quieres terminar (S/N)?',0,2,'SN'
mibolr  dw      0               ;Variables de macros del MASM
mopaux  db      0               ;sitios para paso de dato de
mopaux1 db      0               ;teclado si hay rutina de escape
mopact  db      0
mbcbyte db      0
mbcmenu db      0
mbcoos  dw      0
mbcbuf  dw      0
videoseg dw     0               ;segmento de video
panbuf  db      3680 dup(0)     ;sitio para salvar pantalla
mmvx1   db      0
mmvlin  db      0
mmvlin1 db      0
mmvcar  db      0
mmvdir  dw      0
mv?aux  db      0
mv?aux1 db      0
mmmprx  dw      0
mmylin  db      0
```

```
savsp   dw      0
dirsal  dw      2 dup(0)        ;sitio para dir. de rutinas de usuario
tiprut  db      0               ;tipo de rutina: 1, 2 o 3
tiprutl db      0               ;tipo de rutina de opcion actual
graux1  db      0
grauX1  dw      0               ;Aqui se guardan datos mientras se
grauX3  dw      0               ;corre la rutina del usuario.
dirret  dw      2 dup(0)        ;Direccion de retorno
diract  dw      2 dup(0)
inword  dw      0
getword dw      0               ;salida de GETKEY
anmos   db      0
mdirod  dw      0
mosiin  db      0
mosrol  db      0
mpgdir  dw      0
mcudir  dw      0
mibdir  dw      0
mibx    db      0
fasepir dw      0               ;puntero de fase de programa (73,23)
valptr  dw      0               ;puntero de valor que va en (73,24)
mefdir  dw      0
fnorm   db      "Normal"        ;fases del programa.
fgrab   db      "Grab-M"
fprob   db      "Prob-M"
fcorr   db      "Corr-M"
fmac    db      "Macros"
otroaux db      0               ;bandera que indica paso por "sotr".
blanco  db      78 dup (20h),0
insprog db      "                        OPCIONES VALIDAS EN LOS MENUES.",0
        db      "  Estas son las opciones válides para cualquier menú :",0
        db      "  Se puede elegir una opción por su 'letra clave' o "
        db      "con las flechas y ENTER.",0
        db      "  ^O  presenta/quita las subopciones de una "
        db      "opción en una ventana.",0
        db      "  ESC permite salir al nivel superior de menúes.",0
        db      "  ^P  presenta esta pantalla de ayuda.",0
        db      "  ^K  lleva a las opciones de macros del programa.",0
        db      "                    ...Presiona cualquier tecla...",0
escrma  db      "  Nombre del macro (7 car. máx.) o ENTER para salir...",0
nombuf  db      7,9 dup (0)     ;buffer para recibir el nombre del macro
areamac db      64 dup (0)      ;indice de macros
macl    dw      8192 dup (0)    ;area de macros
nummac  db      0               ;numero del macro actual en el indice
contmcr db      0               ;interface de macros de FLUJO y usuario
longmac db      0               ;longitud del macro actual
contmac db      0               ;contador de posicion en el macro actual
buffmac dw      64 dup (0)      ;macro actual
nombmac db      7 dup (0)       ;nombre del macro actual
nomac   db      "Macros solo desde menú principal. Presiona una tecla.",0
nomoem  db      "No hay sitio para nuevo macro. Presiona una tecla.",0
grmac   db      "Inicio grabación de macro. Terminar: ^K. "
```

```
        db      'Presiona una tecla. ..
borrmac db      'Borro este macro (S/N)?',0,2,'SN'
promac  db      'Inicio prueba de macro. Seguir: ENTER. Terminar: ESC.',0
ejecmac db      'Inicio corrida de macro. Presiona una tecla.',0
escmsr  db      'Fin del buffer y fin del macro. Presiona ESC.',0
insmac  db      33 dup (20h),'MACROS.',0
        db      '       Se pueden usar hasta 8 macros de 64 '
        db      'opciones máximo.',0
        db      '       Ingresa desde menú principal con ^K.',0
        db      '       Escribe el nombre del macro. Termina con ENTER.',0
        db      '       Si es nuevo, definelo.',0
        db      '       Si es viejo lo puedes probar, correr o borrar.',0
        db      '       Este es el indice actual de macros:',0
        db      6 dup (20h)
lismac  db      64 dup (20h),0
pcbmac  db      'Probar...P  Correr...C  Borrar...B',0,3,'PCB'
colven  db      bnnorm          ;atributos: color de ventana
collin  db      bnnorm          ;color de linea
colexp  db      bnnorm          ;color de explicacion
colact  db      bnsubb          ;color de subopcion actual
collact db      bninv           ;color de opcion actual

        .code

        extrn   rinic:near,rfinal:near
        extrn   maxbuf:abs

minfbuf equ     7               ;informaciones asociadas a buffer
longlin equ     160             ;80*2 (incluyendo atributos)
fleder  equ     77              ;teclas pulsadas:
fleizq  equ     75
flearr  equ     72
flecba  equ     80
cr      equ     13
ctrlo   equ     15
ctrlp   equ     16
ctrli   equ     9
ctrlk   equ     11
esqsi   equ     0dah            ;codigos para dibujo de cuadro
linsup  equ     0c4h
esqsd   equ     0bfh
linlat  equ     0b3h
esqii   equ     0c0h
esqid   equ     0d9h
bnnorm  equ     7               ;video normal (B/N)
bnsubb  equ     09h             ;subrayado intenso
bninv   equ     79h             ;inverso
maxmac  equ     64              ;macros de 64 words maximo

        include fmac.asm

flujo   proc    near
```

```
        subrut                  ;subrutinas de macros del MASM.
inicio:
        mov     ax,@data        ;inicializo DS y ES
        mov     ds,ax
        mov     ax,40h
        mov     es,ax
        mov     ax,es:[63h]
        mov     dx,0b000h
        cmp     ax,3b4h
        je      vidaddr
        add     dx,800h
vidaddr:
        mov     videoseg,dx
        mov     es,dx
        mov     ax,0720h        ;20h = " ", 07h = video normal.
        mov     cx,2000
        cld
        xor     di,di
        rep     stosw           ;borro pantalla
        mov     cl,25
i_1:    penar   0ah             ;escribo 25 LF para poner
        dec     cl              ;cursor al fin de pantalla
        jnz     i_1
        mov     faseptr,offset fnorm
        mov     valptr,offset blanco
        call    rinic           ;rutina inicial de usuario.
        guarpantalla            ;muestro instrucciones
        lmppnt
        ayuda   insprog,6,collect
        getkey
        restpantalla
        mov     nivel,0         ;inicializo variables que
        mov     buffer,1        ;manejan flujo del programa.
        mov     buffer1,1h
        mov     bopc,0          ;no subopciones.
        mov     bmac,0          ;no macros.
        mov     bver,0          ;no verificar macros.
        mov     bgmac,0         ;no grabar macros.
        mov     x,1             ;opcion actual
a_nivel:
        mov     y,0             ;subopcion actual
b_subnivel:
        buficla nclave,buffer   ;Variables del menu actual
        movbuf  resbuf,buffer
        mov     al,nivel
        mov     resnivel,al
        movbuf  buffer1,buffer
        mov     al,nivel
        mov     nivel1,a.       ;Variables de opcion actual
        inc     nivel1
        menuvel buffer1,nivel1,x
```

```
        oufcla  nclave1,buffer1
        movbuf  buffer2,buffer1
        mov     al,nivel1
        mov     nivel2,al       ;Variables de subopcion actual
        inc     nivel2
        nunivel buffer2,nivel2,v
        oufcla  nclave2,buffer2
        explica nclave2         ;Explica que hace la subopcion
e_ventana:
        cmp     bopc,1          ;Nuestro ventana de subopciones?
        jne     b_s5
        guarpantalla            ;Si. pero salvo pantalla.
        menuvent nclave1
b_s5:
        menulinea nclave        ;Nuestro menu.
        status1                 ;Fase del programa y
        statusv                 ;valor dado por usuario.
        mov     otroaux,0
c_opcion:
        opcion?                 ;recibe datos de teclado.
        mov     al,mopact       ;# de la opcion elegida.
c_ob1:  actuar                  ;salta a la rutina correcta.
        dw      offset dizq,offset dder,offset darr
        dw      offset daba,offset dret,offset dotr
        dw      offset dcto,offset dcto,offset dctl
        dw      offset dctl
dizq:
        menizq  nclave          ;pasar a opcion a la izquierda
        jmp     b_subnivel
dder:
        mender  nclave          ;opcion a la derecha
        jmp     b_subnivel
darr:
        cmp     bopc,1          ;subopcion superior si
        je      c_oa1           ;la hay y si BOPC=1
        jmp     c_opcion
c_oa1:
        cmp     tipruti,3       ;opcion tiene subopciones?
        jz      c_oa2
        jmp     c_opcion        ;no.
c_oa2:  menarr  nclave1         ;si.
        jmp     b_subnivel
daba:
        cmp     bopc,1          ;subopcion inferior si
        je      c_ob1           ;la hay y si BOPC=1
        jmp     c_opcion
c_ob1:
        cmp     tipruti,3
        jz      c_ob2
        jmp     c_opcion
c_ob2:  menaba  nclave1
        jmp     b_subnivel
```

```
oret:
        inc     nivel           ;se presiono ENTER, correr
        nunivel buffer,nivel,x  ;rutina de opcion actual.
c_or2:  cmp     y,0
        jne     c_or1
        cmp     tiprut1,3
        je      c_or3
        jmp     c_oo2
c_or3:  mov     x,1
        jmp     c_oo2
c_or1:  mov     al,x
        cc.     x,al
        jmp     c_oo2
ootr:
        valida? nclave          ;se digito una opcion desconocida
        je      c_oo1           ;esta entre las validas del menu?
        jmp     e_ventana       ;no.
c_oo1:
        mov     y,0             ;si, correr rutina de esa opcion.
        mov     x,al
        inc     nivel
        nunivel buffer,nivel,x
        mov     otroaux,1
c_oo2:
        oufcla  nclave,buffer   ;aqui se transfiere el control al
        rutina  nclave          ;usuario y este lo devuelve.
c_oo3:  actuar                  ;actuar de acuerdo a tiprut.
        DW      offset rnorm,offset rinv,offset rrama
rnorm:
        movbuf  buffer,respbuf  ;normal, quedarse en nivel actual.
        mov     al,respnivel
        mov     nivel,al
        jmp     b_subnivel
rinv:
        movbuf  buffer,respbuf  ;inversa, salir a nivel superior.
        mov     al,respnivel
        mov     nivel,al
        jmp     d_supernivel
rrama:
        cmp     otroaux,1       ;rama, profundizar.
        jne     rrama1
        mov     x,1
rrama1: jmp     a_nivel
octo:
        cmp     bopc,1                  ;si BOPC=1 hacer 0 y viceversa.
        je      c_oco1
        mov     bopc,1
        jmp     c_oco2
c_oco1: mov     bopc,0
        mov     y,0
c_oco2: jmp     b_subnivel
octp:
```

```
        guarpantalla                    ;mostrar instrucciones.
        lmoont
        ayuda   insprog,8,collact
        getkey
        restoantalla
        jmp     b_submivel
pctm:                                   ;Rutinas de macros.
        mov     contmcr,0               ;reset a byte de interface.
        guarpantalla
        mov     ax,offset blanco
        pmenu   24,colexp
        cmp     nivel,0                 ;nivel actual era 0?
        je      c_ok0
        mov     ax,offset nomac         ;no. salir.
        pmenu   23,collact
        getkey
        jmp     c_ok3
c_ok0:
        mov     x,1                     ;los macros se deben iniciar
        mov     y,0                     ;desde el mismo punto de partida.
        mov     pospc,0
        mov     ax,0                    ;borrar area de nombre de macro.
        push    es
        push    ds
        pop     es
        mov     di,offset nombuf
        inc     di
        mov     cx,9
        rep     stosb
        mov     cx,maxmac               ;borro buffer de macros.
        mov     di,offset buffmac
        rep     stosw
        mov     contmac,0
        mov     cx,7                    ;borro nombre actual.
        mov     di,offset nombmac
        rep     stosb
        pop     es
        indmac                          ;prepara un indice de macros.
        lmppnt
        ayuda   insmac,8,collact        ;instrucciones de macros.
        mov     ax,offset escrma
        pmenu   23,collact
        mov     faseptr,offset fmac
        statusf                         ;muestra fase del programa.
        inbuff  nombuf                  ;recibe un nombre.
        mov     si,offset nombuf
        inc     si
        mov     al,[si]
        cmp     al,0                    ;se dio nombre?
        jne     c_ok01                  ;si.
        mov     faseptr,offset fnorm    ;no, salir.
        statusf
```

```
        jmp     c_ok5
c_ok01: getnmac                         ;obtengo numero de macro.
        cmp     nummac,0ffh             ;macro nuevo sin sitio?
        jne     c_ok1                   ;no
        mov     ax,offset noroom
        pmenu   23,collact
        getkey
        mov     faseptr,offset fnorm
        jmp     c_ok5
c_ok1:  cmp     nummac,80h              ;macro antiguo?
        jb      c_ok2
        jmp     c_ok3                   ;si.
c_ok2:  mov     ax,offset grmac         ;no, grabar macro nuevo.
        pmenu   23,collact
        mov     faseptr,offset fgrab
        statusf
        getkey
        push    es
        push    ds
        pop     es
        mov     si,offset nombuf
        add     si,2
        mov     di,offset nombmac
        mov     cx,7
        rep     movsb
        pop     es
        mov     bgmac,1                 ;bandera de grabacion.
        jmp     c_ok5
c_ok3:  sub     nummac,80h              ;macro viejo.
        mostrar pcbmac,23,collact
        actuar
        dw      offset c_ok31,offset c_ok32,offset c_ok33
c_ok31: mov     ax,offset promac        ;probar macro.
        pmenu   23,collact
        mov     faseptr,offset fprob    ;mostrar fase.
        statusf
        getkey
        call    inimac
        mov     bver,1                  ;banderas para probar macro.
        mov     bmac,1
        jmp     c_ok5
inimac:
        push    es                      ;paso macro al buffer Buffmac
        push    ds
        pop     es
        mov     si,offset mac1
        mov     al,nummac
        mov     cl,2*maxmac
        mul     cl
        add     si,ax
        mov     di,offset buffmac
        mov     cx,maxmac
```

```
        rep     movsw
        mov     si,offset areamac
        mov     al,nummac
        mov     cl,8
        mul     cl
        add     si,ax
        mov     al,byte ptr[si]         ;obtengo datos de macro actual.
        mov     longmac,al
        inc     si
        mov     di,offset nombmac
        mov     cx,7
        rep     movsb
        mov     contmac,0
        pop     es
        retn
c_ok32: mov     ax,offset ejemac        ;correr macro.
        omenu   23,collact
        mov     faseptr,offset fcorr
        status?
        getkey
        call    inimac
        mov     over,0                  ;banderas para correr macro
        mov     bmac,1
        jmp     c_ok5
c_ok33: mov     faseptr,offset inore
        status?
        mostrar borrmac,23,collact      ;borrar macro?
        cmp     al,1
        jne     c_ok5                   ;no
        mov     ax,8                    ;si
        push    es
        push    ds
        pop     es
        mov     cl,nummac
        mul     cl
        add     ax,offset areamac
        mov     di,ax
        mov     cx,8
        xor     ax,ax
        rep     stosb
        mov     ax,2*maxmac
        mov     cl,nummac
        mul     cl
        add     ax,offset macl
        mov     di,ax
        mov     cx,maxmac
        xor     ax,ax
        rep     stosw
        pop     es
c_ok5:  restpantalla
        jmp     b_subnivel
```

```
;;;;;;;;;;;;;;;;;;;;;;;;;;DESENS.ASM;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;Programa desensamblador.
;Por IVAN ORDOÑEZ REINOSO.
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
        .model  small

        .data

segreg  db      'E,C,S,D,'
oper1   db      'ADD,OR,ADC,SBB,AND,SUB,XOR,CMP,'
oper2   db      'DAA,DAS,AAA,AAS,'
oper3   db      'INC,DEC,PUSH,POP,'
oper4   db      'O,NO,C,NC,Z,NZ,BE,A,S,NS,P,NP,L,GE,LE,G,'
oper5   db      'PUSH,POP,SAH,LAH,'
oper6   db      'ROL,ROR,RCL,RCR,SHL,SHR,???,SAR,'
oper7   db      'TEST,???,NOT,NEG,MUL,IMUL,DIV,IDIV,'
oper8   db      'CLC,STC,CLI,STI,CLD,STD,'
oper9   db      'INC,DEC,CALL,CALL FAR,JMP,JMP FAR,PUSH,???,'
operpt  db      'PUSH ,POP ,'
opertx  db      'TEST,XCHG,'
operm?  db      'MOV,???,'
operlea db      'LEA,'
operxa  db      'XCHG AX,'
opercc  db      'CWD,CBW,'
operjc  db      'JMP,CALL,'
operwai db      'WAIT,'
operca  db      'CMPS,MOVS,'
opersto db      'STOS,'
opersl  db      'SCAS,LODS,'
operrn  db      'RETN,'
operrf  db      'RETF,'
operll  db      'LDS,LES,'
operi3  db      'INT 03,'
operint db      'INT ,'
operio  db      'INTO,'
operir  db      'IRET,'
operaa  db      'AAD,AAM,'
operxla db      'XLAT,'
operesc db      'ESC ,'
operjl  db      'JCXZ,LOOP,'
operaio db      'OUT ,IN ,OUT DX,'
operlk  db      'LOCK,'
operrrr db      'REPE,REP,REPNE,'
operhlt db      'HLT,'
opercmc db      'CMC,'
opernop db      'NOP,'
reg1    db      'AL,CL,DL,BL,AH,CH,DH,BH,'
reg2    db      'AX,CX,DX,BX,SP,BP,SI,DI,'
modos   db      'SI+BX,DI+BX,SI+BP,DI+BP,SI,DI,BP,BX,'

casos   db      40h,60h,70h,80h,84h,88h,8ch,90h
        db      91h,98h,9ah,9bh,9ch,0a0h,0a4h,0a8h
```

```
        db      [illegible]
        db      [illegible]
        db      [illegible]
        db      [illegible]

direct  dw      offset case40to5f,      offset case60to7f
        dw      offset case80to9f,      offset case70to7f
        dw      offset case80to8f,      offset case90to97
        dw      offset case98to9b,      offset case9cto9f
        dw      offset case90,          offset case91to97
        dw      offset case98to99,      offset case9a
        dw      offset case9a,          offset case9cto9f
        dw      offset caseA0toA3,      offset caseA4toA7
        dw      offset caseA8toA9,      offset caseAatoAb
        dw      offset caseactoaf,      offset caseb0tobf
        dw      offset casec0toc1,      offset casec2toc3
        dw      offset casec4toc5,      offset casec6toc7
        dw      offset casec8toc9,      offset casecatocb
        dw      offset casecc,          offset casecc
        dw      offset casece,          offset casecf
        dw      offset casec0tod1,      offset casec4tod5
        dw      offset casec8,          offset casec
        dw      offset cased8tod9,      offset casee0toe7
        dw      offset casee4toe7,      offset casee8toe9
        dw      offset caseea,          offset caseeb
        dw      offset caseactoef,      offset casef0
        dw      offset casef1,          offset casef2tof3
        dw      offset casef4,          offset casef5
        dw      offset casef6tof7,      offset casef8toid

aaaaa   db      0
ll      db      0
addr    dw      ?
one     db      ?
opcode  db      7 dup (?)
astring dw      ?
bstring dw      ?
atastr  db      20 dup (?)
atbstr  db      20 dup (?)

        .code

readarra\       proc    near
;-----------------------------------------------------------------------
;Este procedimiento lee un elemento de un arreglo alfanumérico, de dirección
;especificada por DS:SI. El elemento se coloca desde ES:DI, y DI se acomoda
;para que apunte a la dirección siguiente al ultimo carácter del elemento.
;Los separadores entre elementos deben ser comas (,).
;
;Entrada:        CX = número del elemento en el arreglo.
;
;Salida:         DI = dirección del ultimo carácter transferido + 1
```

```
;-----------------------------------------------------------------------
        push    ax
        push    cx
        push    si
        or      cl,ch           ;primer elemento del arreglo?
        jz      encont          ;si
buscoma:                        ;buscar el elemento
        inc     si
        cmp     byte ptr[si],','  ;separador?
        jne     buscoma         ;no
        loop    buscoma         ;si, decrementar numero de elemento
        inc     si              ;acuntar al elemento encontrado
encont:
        lodsb                   ;carácter en AL
        cmp     al,','          ;separador?
        je      finread         ;si, finalizar transferencia
        stosb                   ;no, transferir
        jmp     short encont
finread:
        pop     si
        pop     cx
        pop     ax
        retn
readarray       endp

        public convhex
convhex proc    near
;-----------------------------------------------------------------------
;Este procedimiento genera un string de 2 caracteres desde ES:DI que
;representan hexadecimalmente al número en AL. DI se acomoda como en el
;procedimiento readarray.
;
;Entrada:       AL = número por convertir.
;
;Salida:        DI = dirección del ultimo carácter generado + 1
;-----------------------------------------------------------------------
        push    ax
        push    cx
        mov     ah,al
        and     ah,0fh          ;4 lsb en AH
        mov     cl,4
        shr     al,cl           ;4 msb en AL
        or      ax,3030h        ;conv. a ascii
        cmp     al,'9'
        jbe     numal
        add     al,'A'-'9'-1
numal:  cmp     ah,'9'
        jbe     numah
        add     ah,'A'-'9'-1
numah:  stosw
        pop     cx
        pop     ax
```

```
        retn
convhex endp

conv2hex        proc    near
;---------------------------------------------------------------------------
;Este procedimiento genera un string de 4 caracteres desde ES:DI que
;representan hexadecimalmente a la palabra apuntada por DS: SI+BX. DI se
;acomoda como en reasarray.
;
;Salida:        DI = dirección del ultimo caracter generado + 1
;
;Usa:           convhex
;---------------------------------------------------------------------------
        push    ax
        mov     ax,[si+bx]
        push    ax
        mov     al,ah
        call    convhex
        pop     ax
        call    convhex
        pop     ax
        retn
conv2hex        endp

cod3bit proc    near
;---------------------------------------------------------------------------
;Este procedimiento obtiene el número de 3 bits de los bits 3, 4 y 5 de
;BL, y lo coloca en BL. Es de utilidad para, por ejemplo, obtener reg
;en el byte de modo de direccionamiento.
;
;Entrada:       BL
;
;Salida:        BL = (BL and 38H)/8
;---------------------------------------------------------------------------
        push    cx
        and     bl,38h
        mov     cl,3
        shr     bl,cl
        pop     cx
        retn
cod3bit endp

modrm   proc    near
;---------------------------------------------------------------------------
;Este procedimiento genera el string de modo de direccionamiento [...]
;desde DS:bstring, y lo termina con 0.
;
;Entrada:       AL = tipo de operador (w)
;
;Salida:        CX = longitud presumida de instrucción
;               memop = 1 si se opera con memoria
;
```

```
;Usa:          convhex, conv2hex, readarray
;-----------------------------------------------------------------------

        push    cx
        push    bx
        push    si
        push    di
        push    es

        push    cs
        pop     es
        mov     di,bstring
        mov     bx,11
        mov     bl,[si+bx+1]    ;segundo byte del opcode
        mov     bh,bl
        and     bh,0c0h
        mov     cl,6
        shr     bh,cl           ;2 msb de dicho byte en bh (mod)
        mov     dl,bl
        and     dl,7            ;3 lsb de dicho byte en dl (r/m)
        cmp     dl,6
        jne     nodir           ;si no fuera modo directo
        or      bh,bh
        jnz     nodir           ;idem

        mov     memop,1         ;modo directo: mod=0, r/m=6
        mov     al,'['
        stosb                   ;poner la dirección entre corchetes
        mov     bx,11           ;y terminar string con 0.
        add     bx,2
        call    conv2hex
        mov     al,']'
        xor     ah,ah
        stosw
        mov     cx,4            ;long. 4 bytes
        jmp     short finmodrm

nodir:  cmp     bh,3            ;registro a registro?
        jne     no3             ;no
        mov     cl,dl           ;si,leer de la tabla de registros
        xor     ch,ch
        push    si
        mov     si,offset reg1  ;poner nombre de registro en string
        or      al,al           ;instrucción de word?
        jz      primfila        ;no
        mov     si,offset reg2  ;si
primfila:
        call    readarray
        xor     al,al
        stosb                   ;terminar con 0
        mov     cx,2
        pop     si
        jmp     short finmodrm
```

```
no3:    mov     memop,1         ;operacion con memoria
        push    bx
        mov     al,'['
        stosb
        push    si
        mov     si,offset modos
        mov     cl,dl           ;leer un modo de direc. de la tabla
        xor     ch,ch
        call    readarray
        pop     si
        cmp     bh,1            ;modo=1?
        jne     modno1          ;no
        mov     bx,11           ;si, offset de un byte
        add     bx,2
        mov     bl,[si+bx]      ;obtener offset
        cmp     bl,127          ;hallar su signo
        jbe     positoffs
        mov     al,'-'
        neg     bl
        jmp     short addsgn
positoffs:
        mov     al,'+'
addsgn: stosb
        mov     al,bl
        call    convhex         ;incluir offset en el string
        jmp     short endselmodrm

modno1: cmp     bh,2            ;hay offset?
        jne     short endselmodrm
        mov     al,'+'          ;si, de 2 bytes
        stosb
        mov     bx,11
        add     bx,2
        call    convZhex
endselmodrm:
        mov     al,']'
        xor     ah,ah
        stosw                   ;acabar con 0
        pop     bx
        mov     cl,bh
        xor     ch,ch
        add     cx,2            ;long. = mod + 2
finmodrm:
        pop     es
        pop     di
        pop     ax
        pop     bx
        pop     dx
        retn
modrm   endp
```

```
modregrm        proc    near
;-----------------------------------------------------------------------
;Este procedimiento genera el string de modo de direccionamiento y su
;registro operador en el orden apropiado a partir de ES:DI y acomoda DI
;como en readarray. Utiliza DS:astring y DS:bstring
;
;Entrada:       Am = ok (oar de bits de dirección y tipo de operador
;
;Salida:        CX = longitud de instruccion
;               AL = w (tipo de operador,
;               membp = 1 si se ha operado con memoria.
;
;Usa:           modrm, cod3bit, readarray, order
;-----------------------------------------------------------------------
        push    bx
        push    dx
        push    si
        push    es

        push    cs
        pop     es
        mov     al,am
        and     al,1            ;obtener w
        call    modrm           ;obtener [...]
        call    cod3bitinc      ;obtener cod del byte de modo de dir.
        mov     si,offset reg1  ;leer reg de la tabla segun w
        or      al,al
        jz      bringreg
        mov     si,offset reg2
bringreg:
        push    cx
        push    di
        mov     di,astring
        mov     cl,bl
        xor     ch,ch
        call    readarray
        mov     dh,ah           ;obtener d en DH
        and     dh,2
        pop     di
        pop     cx
        pop     es
        call    order           ;poner en orden [...] y su operador

        pop     si
        pop     dx
        pop     bx
        retn
modregrm        endp

order   proc    near
;-----------------------------------------------------------------------
;Este procedimiento establece el orden en que deben ir astring y bstring en
```

```
;el mnemotecnico (a partir de ES:DI) segun el bit de dirección (DH) y los
;coloca, acomodando despues DI como en readarray.
;
;Entrada:       DH = bit de direccion
;
;usa:           addastr, addbstr
;-----------------------------------------------------------------------
        push    ax
        mov     al,
        stosb                   ;poner un espacio
        or      dh,dh
        jz      bfirst

        call    addastr
        mov     al,','
        stosb
        call    addbstr
        jmp     short finorder

bfirst: call    addbstr
        mov     al,','
        stosb
        call    addastr
finorder:
        pop     ax
        retn
order   endp

addstr  proc    near
;-----------------------------------------------------------------------
;Este procedimiento coloca un string apuntado por DS:SI terminado en 0
;desde ES:DI, acomodando DI como en readarray.
;-----------------------------------------------------------------------
        push    si
        push    ax
xfer:   lodsb
        or      al,al           ;fin de string?
        jz      finaddstr       ;si
        stosb                   ;no, continuar
        jmp     short xfer
finaddstr:
        pop     ax
        pop     si
        retn
addstr  endp

;-----------------------------------------------------------------------
;Los siguientes dos procedimientos colocan los strings apuntados por
;astring o bstring desde ES:DI, acomodando DI como en readarray.
;
;Usan:  addstr
;-----------------------------------------------------------------------
```

```
accastr proc    near
        push    si
        mov     si,astring
        call    accstr
        pop     si
        retn
accastr endp

addbstr proc    near
        push    si
        mov     si,bstring
        call    addstr
        pop     si
        retn
addbstr endp

accdata proc    near
;-----------------------------------------------------------------------
;Este procedimiento genera un string de operación de acumulador con datos
;inmediatos, y lo coloca a partir de ES:DI, acomodando DI como en readarray.
;
;Entrada:       AL = w (bit de tipo de operador)
;
;Salida:        CX = longitud de instrucción
;
;Usa:           readarray, convhex, conv2hex
;-----------------------------------------------------------------------
        push    bx
        push    dx
        push    si
        mov     dl,al           ;preservar w
        mov     al,' '
        stosb                   ;poner un espacio
        xor     cx,cx           ;leer string de acumulador (AL o AX)
        mov     si,offset reg1
        or      dl,dl           ;segun w
        jz      data1
        mov     si,offset reg2
data1:  call    readarray
        pop     si
        mov     al,','
        stosb                   ;poner una coma despues del acumulador
        mov     bx,11
        inc     bx              ;SI+BX apunta al dato inmediato
        or      dl,dl           ;que es de 1 o 2 bytes segun w
        jz      dat1
        call    conv2hex
        jmp     short finaccdata
dat1:   mov     al,[si+bx]
        call    convhex
finaccdata:
        mov     al,dl           ;recuperar w
```

```
        mov     cl,2
        add     cl,al           ;longitud de instrucción
        pop     si
        pop     dx
        retn
accesta endp

realadr proc    near
;-----------------------------------------------------------------
;Este procedimiento calcula la dirección real de un salto relativo y la
;convierte en un string a partir de ES:DI, con DI como en entrada.
;
;Entrada:       AL = w
;               CX = longitud de instruccion
;
;Usa:           convhex
;-----------------------------------------------------------------
        push    ax
        push    bx
        mov     bx,11           ;apuntar al opcode
        or      al,al           ;desplazamiento de un byte?
        mov     al,[si+bx+1]
        cbw
        jz      dsplb           ;si
        mov     ah,[si+bx+2]    ;no
dsplb:  add     ax,ador
        add     ax,cx           ;direccion real en ax
        push    ax
        mov     al,' '
        stosb                   ;poner un espacio
        mov     al,ar           ;poner el string
        call    convhex
        pop     ax
        call    convhex
        pop     bx
        pop     ax
        retn
realadr endp

wob     proc    near
;-----------------------------------------------------------------
;Este procedimiento coloca "W" o "B" en ES:DI e incrementa DI, segun w. Es
;util para completar instrucciones de string.
;
;Entrada:       AH = opcode
;
;Salida:        AL = w
;               CX = longitud de instrucción
;-----------------------------------------------------------------
        mov     al,ah
        and     al,1            ;AL = w
        push    ax
```

```
        jnz     instw
        mov     al,'B'
        jmp     short addar
instw:  mov     al,'W'
addar:  stosb
        mov     cx,1            ;long = 1
        pop     si
        retn
mod     endp

datadic proc    near
;---------------------------------------------------------------------
;Este procedimiento genera el string de modo de direccionamiento [...] y
;un operador inmediato desde ES:DI. Acomoda DI como readarray .
;
;Entrada:       AH = opcode
;               DL = sw (bits de extension de signo y tipo de operador)
;
;Salida:        AL = w
;               CX = longitud de instruccion
;
;Usa:           modrm, addbstr, convhex
;---------------------------------------------------------------------
        push    bx
        mov     al,ah
        and     al,1            ;obtener w
        mov     bx,offset atbstr
        mov     bstring,bx
        call    modrm           ;generar [...]
        push    ax
        mov     al,' '
        stosb                   ;poner un espacio
        call    addbstr         ;poner [...]
        mov     al,','          ;poner una coma
        stosb
        mov     bx,11           ;apuntar al opcode
        add     bx,cx           ;apuntar a los datos
        mov     ax,[si+bx]
        inc     cx
        cmp     dl,1
        jne     unbyte          ;s=0 y w=1 implica dato de 2 bytes
        inc     cx
        mov     bl,al
        mov     al,ah
        call    convhex
        mov     al,bl
unbyte: call    convhex
        pop     ax
        pop     bx
        retn
datadic endp
```

```
datret  proc    near
;-------------------------------------------------------------------------
;Este procedimiento decide si ciertas instrucciones requieren datos, y, de
;ser así, los incluye como string desde ES:DI, acomoda DI como readarray.
;
;Entrada:       AH = opcode
;
;Salida:        AL = w
;               CX = long. de instruccion
;
;Usa:           conv2hex
;-------------------------------------------------------------------------
        push    bx
        test    ah,1
        jz      naydat          ;si el lsb del opcode es 0, nay datos
        mov     cx,1
        jmp     short findatret
naydat: mov     al,' '          ;poner un espacio
        stosb
        mov     bx,11
        inc     bx
        call    conv2hex        ;poner dato
        mov     cx,3
findatret:
        mov     al,1            ;w=1
        pop     bx
        retn
datret  endp

absoluto        proc    near
;-------------------------------------------------------------------
;Este procedimiento pone un string de dirección absoluta desde ES:DI,
;acomodando DI como readarray.
;
;Salida:        AL = w
;               CX = long. de instrucción
;
;Usa:           conv2hex
;-------------------------------------------------------------------
        push    bx
        mov     al,' '
        stosb                   ;poner un espacio
        mov     bx,11
        add     bx,3            ;apuntar al segundo par de bytes
        call    conv2hex
        mov     al,':'
        stosb                   ;poner dos puntos
        sub     bx,2            ;apuntar al primer par de bytes
        call    conv2hex
        mov     cx,5
        mov     al,2
        pop     bx
```

```
        retn
absoluto        endp

cod311nc        proc    near
;---------------------------------------------------------------
;Este procedimiento encuentra el codigo de 3 bits del byte siguiente
;al opcode en SI. Es un procedimiento de ayuda.
;---------------------------------------------------------------
        mov     cx,3
        mov     bl,[si+cx+1]
        call    cod3bit
        retn
cod311nc        endp

incognita       proc    near
;---------------------------------------------------------------
;Este procedimiento corresponde al desensamblado de un codigo
;totalmente inválido, y establece los parámetros correspondientes.
;---------------------------------------------------------------
        mov     si,offset opera?
        mov     cx,1            ;???
        call    respoarray
        mov     al,al           ;terminador
        stosb
        mov     cx,1            ;long.
        mov     al,255
        retn
incognita       endp


        public  desens

desens  proc    near
;---------------------------------------------------------------
;Este es el procedimiento principal. Genera un string desde ES:DI
;terminado en 0, desensamblado de la instrucción apuntada por DS:SI.
;Preserva todos los registros que no transfieren parámetros.
;
;Salida:        CX = longitud de la instrucción
;               AL = tipo de operador:
;                       0 = byte
;                       1 = word
;                       2 = dword
;                     255 = indefinido
;---------------------------------------------------------------
        pushf
        push    bx
        push    dx
        push    bp
        push    ds
        push    si
        push    di
```

```
        cld
        mov     bx,si
        mov     ax,@data
        push    es
        push    di
        mov     es,ax
        mov     di,offset opcode
        mov     cx,7
        rep     movsb           ;transferir instrucción al segmento de datos
        mov     ds,ax
        mov     addr,bx         ;dirección de la instrucción
        mov     si,offset opcode
        mov     al,[si]
        mov     ah,al
        and     ah,0e7h
        cmp     ah,26h          ;prefijo de redef. de segmento?
        jne     nopref
        mov     ll,1            ;opcode desplazado un byte
        and     al,18h
        mov     cl,3
        shr     al,cl           ;número del segmento 0 3
        push    si
        mov     si,offset segreg
        xor     ah,ah
        add     si,ax
        mov     al,[si]
        pop     si
        mov     mne,al
        jmp     short proced
nopref: mov     ll,0
        mov     mne,0

proced: mov     memop,0
        mov     bx,ll
        mov     ah,[si+bx]      ;opcode en ah
        cmp     ah,0fch
        jb      menfe
        pop     di
        pop     es
        push    di
        call    casefetoff
        jmp     short ajustar
menfe:  mov     di,offset casos
        mov     al,ah
        xor     bx,bx
buscop: scasb
        jb      encontrado
        inc     bx
        jmp     short buscop
encontrado:
        pop     di
```

```
        pop     es
        shl     bx,1
        add     bx,offset direct
        push    ax
        call    [bx]

ajustar:cmp     ax              ;incluir el prefijo de redefinición de
        cmp     ll,0            ;segmento de ser posible y necesario
        je      fin
        cmp     memop,0
        je      prpref

        push    ax
        push    cx
        push    di
        mov     al,'['
        mov     cx,30
        repne   scasb           ;buscar un [
        push    di
        xor     al,al
        repne   scasb           ;buscar un null
        pop     cx
        sub     cx,ax           ;cx debe tener el numero de caracteres que
        neg     cx              ;van a moverse a la derecha para que pueda
        inc     cx              ;entrar el prefijo de redefinición de seg.

        push    ds
        push    es
        pop     ds
        mov     si,di
        dec     si
        add     di,2            ;preparar el movimiento
        std                     ;hacia atrás
        rep     movsb           ;acomodar string
        cld
        pop     ds

        mov     word ptr es:[di-1],':S'
        mov     al,mne
        mov     es:[di-2],al
        pop     di
        pop     cx
        pop     ax
        inc     cx              ;la long. de instrucción incluye al prefijo
        jmp     short fin
prpref:
        mov     al,mne
        mov     es:[di],al
        mov     word ptr es:[di+1],':S'
        mov     byte ptr es:[di+3],0
        mov     al,255
        mov     cx,1
```

```
        jmp     short fin6
case3:  mov     si,offset segreg
        mov     cl,bl
        and     cx,7
        call    readarray
        mov     al,':'
        stosw
fin6:   mov     cx,1
        mov     al,1
        pop     si
        jmp     short fin0to3f
copyout6:
        push    si
        cmp     bl,3
        ja      case3
        mov     si,offset operpp
        mov     cx,1            ;pop
        call    readarray
        mov     si,offset segreg
        mov     cl,bl
        call    readarray
        mov     al,'S'
        stosb
        jmp     short fin7
case3:  mov     si,offset oper2
        mov     cl,bl
        and     cx,3
        call    readarray
fin7:   mov     cx,1
        mov     al,1
        pop     si
fin0to3f:
        push    ax
        xor     al,al
        stosb
        pop     ax
        retn
case0to3f       endp

case40to5f      proc    near
        mov     bl,ah
        and     bl,18h
        mov     cl,3
        shr     bl,cl
        mov     cl,bl
        xor     ch,ch
        mov     si,offset oper3
        call    readarray
        mov     al,' '
        stosb
        mov     si,offset reg2
        mov     cl,ah
```

```
        push    si
        mov     si,offset operix
        call    readarray
        pop     si
        and     ah,1
        or      ah,2            ;ok
        mov     dx,offset atastr
        mov     astring,dx
        mov     dx,offset atostr
        mov     bstring,dx
        call    modregrm
        push    ax
        xor     al,al
        stosb
        pop     ax
        retn
case84to87      endp

case88to8c      proc    near
        push    si
        mov     si,offset operm?
        xor     cx,cx           ;mov
        call    readarray
        pop     si
        and     ah,3
        mov     dx,offset atastr
        mov     astring,dx
        mov     dx,offset atbstr
        mov     bstring,dx
        call    modregrm
        push    ax
        xor     al,al
        stosb
        pop     ax
        retn
case88to8b      endp

case8cto8f      proc    near
        push    si
        test    ah,1
        jnz     leapop
        call    cod3llinc
        mov     si,offset operm?
        xor     cx,cx           ;mov
        cmp     bl,3
        jbe     valido
        inc     cx              ;???
valido: call    readarray
        and     bx,3
        mov     si,offset segreg
        shl     bl,1
        add     si,bx
```

```
        push    es
        push    di
        push    cs
        pop     es
        mov     di,offset atastr
        mov     astring,di
        movsb
        mov     al,' '
        stosb
        mov     di,offset atbstr
        mov     bstring,di
        pop     di
        pop     es
        mov     al,1
        pop     si
        call    modrm
        mov     dh,ah
        and     dh,2            ;d
        call    order
        jmp     short fin8cto8f
leapop: test    ah,2
        jz      eslea
        call    cod31inc
        or      bl,bl
        jz      espop
        mov     si,offset opera?
        mov     cx,1            ;???
        call    readarray
        mov     al,' '
        stosb
        jmp     short inclmod
espop:  mov     si,offset operpp
        mov     cx,1            ;pop
        call    readarray
inclmod:pop     si
        mov     dx,offset atbstr
        mov     bstring,dx
        mov     al,1
        call    modrm
        call    addbstr
        jmp     short fin8cto8f
eslea:  mov     si,offset operlea
        xor     cx,cx           ;lea
        call    readarray
        mov     dx,offset atastr
        mov     astring,dx
        mov     dx,offset atbstr
        mov     bstring,dx
        mov     ah,3
        pop     si
        call    modregrm
fin8cto8f:
```

```
        push    ax
        xor     al,al
        stosb
        pop     ax
        retn
case8cto8f      endp

case90  proc    near
        mov     si,offset opernop
        xor     cx,cx           ;nop
        call    readarray
        xor     al,al
        stosb
        mov     al,1
        mov     cx,1
        retn
case90  endp

case91to97      proc    near
        mov     si,offset operxax
        xor     cx,cx           ;xchg ax
        call    readarray
        mov     al,','
        stosb
        mov     cl,ah
        and     cx,7
        mov     si,offset reg2
        call    readarray
        xor     al,al
        stosb
        mov     al,1
        mov     cx,1
        retn
case91to97      endp

case98to99      proc    near
        mov     si,offset opercc
        xor     cx,cx           ;cwd
        mov     al,ah
        and     al,1            ;w
        jnz     wd
        inc     cx              ;cbw
wd:     call    readarray
        mov     cx,1
        push    ax
        xor     al,al
        stosb
        pop     ax
        retn
case98to99      endp

case9a  proc    near
```

```
        push    si
        mov     si,offset operjc
        mov     cx,1            ;call
        call    readarray
        pop     si
        call    specluto
        push    ax
        xor     al,al
        stosb
        pop     ax
        retn
case7a  endp

case9b  proc    near
        mov     si,offset operwai
        xor     cx,cx           ;wait
        call    readarray
        xor     al,al
        stosb
        mov     al,255
        mov     cx,1
        retn
case9b  endp

case9cto9f      proc    near
        mov     si,offset oper5
        mov     cl,ah
        and     cx,3
        call    readarray
        mov     ax,46h          ;F
        stosw
        mov     al,1
        mov     cx,1
        retn
case9cto9f      endp

caseA0toA3      proc    near
        push    si
        mov     byteop,1
        mov     si,offset operw?
        xor     cx,cx           ;mov
        call    readarray
        pop     si
        mov     al,' '
        stosb
        mov     al,ah
        and     al,1
        mov     dx,offset reg1
        jz      prfil
        mov     dx,offset reg2
prfil:  push    ax
        test    ah,2
```

```
        jz      abr
        call    portb
        mov     al, ,
        stosb
        call    pona
        jmp     short finaOtoa3
abr:    call    pona
        mov     al, .
        stosb
        call    ponb
finsOtoa3:
        xor     al,al
        stosb
        pop     ax
        mov     cx,3
        retn
pona:   xor     cx,cx
        push    si
        mov     si,dx
        call    readarray
        pop     si
        retn
ponb:   mov     al,'['
        stosb
        mov     bx,11
        inc     bx
        call    conv2hex
        mov     al,']'
        stosb
        retn
caseaOtoa3      endp

casea4toa7      proc    near
        mov     si,offset operca
        xor     cx,cx           ;caps
        test    ah,2
        jnz     escaps
        inc     cx              ;movs
escaps: call    readarray
        call    wob
        push    ax
        xor     al,al
        stosb
        pop     ax
        retn
casea4toa7      endp

caseaBtoa9      proc    near
        push    si
        mov     si,offset opertx
        xor     cx,cx           ;test
        call    readarray
```

```
        mov     al,ah
        and     al,1
        mov     si
        call    accsetz
        push    ax
        xor     al,al
        stosb
        pop     ax
        retn
case8to8        endp

case8atob       proc    near
        mov     si,offset operstb
        xor     cx,cx           ;stos
        call    readarray
        call    woo
        push    ax
        xor     al,al
        stosb
        pop     ax
        retn
case8atob       endp

case8ctof       proc    near
        mov     si,offset opersl
        xor     cx,cx           ;scas
        test    ah,2
        jnz     esscas
        inc     cx              ;lods
esscas: call    readarrav
        call    woc
        push    ax
        xor     al,al
        stosb
        pop     ax
        retn
case8ctof       endp

caseb0tob:      proc    near
        push    si
        mov     si,offset opere?
        xor     cx,cx           ;mov
        call    readarray
        mov     al,' '
        stosb
        mov     al,ah
        and     al,8
        mov     cl,3
        shr     al,cl
        mov     si,offset reg1
        jz      esr1
        mov     si,offset reg2
```

```
        mov     astring,ax
        mov     ax,offset atostr
        mov     bstring,ax
        mov     an,3
        call    macregs
        over    2
        xor     al,al
        stost
        pop     ax
        retn
case4toc5       endp

case6toc7       proc    near
        call    cod311inc
        push    si
        mov     si,offset opera7
        xor     cx,cx           ;mov
        or      bl,bl
        jz      asmov
        inc     cx              ;???
asmov:  call    readarray
        pop     si
        mov     dl,ah
        and     dl,1            ;sw
        call    datadic
        push    ax
        xor     al,al
        stosb
        pop     ax
        retn
case6toc7       endp

case8toc9       proc    near
        call    incognita
        retn
case8toc9       endp

casecatocb      proc    near
        push    si
        mov     si,offset operf
        xor     cx,cx           ;retf
        call    readarray
        pop     si
        call    datret
        push    ax
        xor     al,al
        stosb
        pop     ax
        retn
casecatocb      endp

casecc  proc    near
```

```
        ouer    si
        mov     si,offset opere
        call    readarray
        pop     si
        mov     al,' '
        stosb
        mov     al,ah
        and     al,1
        mov     dx,offset atostr
        mov     bstring,dx
        call    modrm
        call    accostr
        push    ax
        mov     al,','
        stosb
        test    ah,2
        jz      es1
        mov     ax,'LC'
        stosw
        jmp     short find0tod7
es1:    mov     al,'1'
        stosb
find0tod7:
        xor     al,al
        stosb
        pop     ax
        retn
cased0tod3      endp

cased4tod5      proc    near
        push    si
        mov     si,offset operaa
        xor     cx,cx           ;aad
        test    ah,1
        jnz     esaad
        inc     cx              ;aam
esaad:  call    readarray
        pop     si
        mov     bx,11
        mov     bl,[si+bx+1]
        cmp     bl,10
        je      norm
        mov     al,' '
        stosb
        mov     al,bl
        call    convhex
norm:   xor     al,al
        stosb
        mov     cx,2
        retn
cased4tod5      endp
```

```
case0o  proc    near
        call    incognita
        retn
case0o  endp

case07  proc    near
        mov     si,offset operxla
        xor     cx,cx           ;xlat
        call    readarray
        xor     al,al
        stosb
        mov     cx,1
        retn
case07  endp

case08to0f      proc    near
        call    cod311inc
        push    si
        mov     si,offset operesc
        xor     ch,ch           ;esc
        call    readarray
        pop     si
        mov     al,ah
        and     al,7
        mov     cl,3
        shl     al,cl
        add     al,bl
        call    convhex
        mov     al,','
        stosb
        mov     dx,offset atostr
        mov     ostring,dx
        xor     al,al           ;esto e dudoso
        call    modrm
        call    addostr
        mov     al,255
        retn
case08to0f      endp

case0e0to3      proc    near
        push    si
        mov     si,offset operjl
        mov     al,ah
        and     al,3
        xor     cx,cx           ;jcxz
        cmp     al,3
        jne     esloop
        call    readarray
        jmp     short pondir
esloop: inc     cx              ;loop
        call    readarray
        test    ah,2
```

```
        jnz     condir
        test    ah,1
        jz      esne
        mov     al,'E'
        stosb
        jmp     short condir
esne:   mov     ax,'EN'
        stosw
condir: pop     si
        xor     al,al
        mov     cx,2
        call    realaddr
        push    ax
        xor     al,al
        stosb
        pop     ax
        retn
case0to3        endp

case4to7        proc    near
        mov     al,an
        and     al,1
        push    si
        mov     si,offset operand
        test    ah,2
        jz      esin
        xor     cx,cx           ;out
        call    readarray
        pop     si
        mov     bx,11
        mov     dx,ax
        mov     al,[si+bx+1]
        call    convhex
        mov     ax,'A,'
        stosw
        or      dl,dl
        mov     al,'X'
        jnz     esax
        mov     al,'L'
esax:   stosb
        jmp     short fin4to7
esin:   mov     cx,1            ;in
        call    readarray
        mov     dx,ax
        mov     al,'A'
        stosb
        or      dl,dl
        mov     al,'X'
        jnz     esregax
        mov     al,'L'
esregax:mov     ah,','
        stosw
```

```
        pop     si
        mov     dx,11
        mov     al,[si+bx+11
        call    convhex
fine4toe7:
        xor     al,al
        stosb
        mov     ax,dx
        mov     cx,2
        retn
case4toe7       endp

case8toe9       proc    near
        push    si
        mov     si,offset operjc
        xor     cx,cx           ;jmp
        test    ah,1
        jnz     esjmp
        inc     cx              ;call
esjmp:  call    readarray
        pop     si
        mov     al,1
        mov     cx,3
        call    realadr
        push    ax
        xor     al,al
        stosb
        pop     ax
        retn
case8toe9       endp

caseea  proc    near
        push    si
        mov     si,offset operjc
        xor     cx,cx           ;jmp
        call    readarray
        pop     si
        call    absoluto
        push    ax
        xor     al,al
        stosb
        pop     ax
        retn
caseea  endp

caseeb  proc    near
        push    si
        mov     si,offset operjc
        xor     cx,cx           ;jmp
        call    readarray
        pop     si
        xor     al,al
```

```
        mov     cx,2
        call    realaor
        push    ax
        xor     al,al
        stosb
        pop     ax
        ret
caseeb  endp

caseectoef      proc    near
        mov     al,ah
        and     al,1
        mov     si,offset operoic
        test    ah,2
        jz      esin_
        mov     cx,2            ;out dx
        call    readarray
        mov     cx,ax
        mov     ax,'A,'
        stosw
        mov     al,'X'
        or      dl,dl
        jnz     esrax
        mov     al,'L'
esrax:  stosb
        jmp     short finectoef
esin_:  mov     cx,1            ;in
        call    readarray
        mov     dx,ax
        mov     al,'A'          ,
        stosb
        mov     al,'X'
        or      dl,dl
        jnz     esrqax
        mov     al,'L'
esrqax: mov     ah,','
        stosw
        mov     ax,'XD'
        stosw
finectoef:
        mov     cx,1
        xor     al,al
        stosb
        mov     ax,dx
        ret
caseectoef      endp

casef0  proc    near
        mov     si,offset operlk
        xor     cx,cx           ;lock
        call    readarray
        xor     al,al
```

```
        stosb
        mov     al,255
        mov     cx,1
        retn
casef0  endp

casef1  proc    near
        call    incognita
        retn
casef1  endp

casef2tof3      proc    near
        push    si
        xor     cx,cx           ;repe
        test    ah,1
        jz      esrepne
        mov     bx,11
        mov     al,[es:bx+1]
        and     al,0a6h
        cmp     al,0a6h
        je      finf2tof3
        inc     cx              ;rec
        jmp     short finf2tof3
esrepne:mov     cx,2            ;repne
finf2tof3:
        mov     si,offset operrrr
        call    readarray
        xor     al,al
        stosb
        pop     si
        mov     al,255
        mov     cx,1
        retn
casef2tof3      endp

casef4  proc    near
        mov     si,offset operhlt
        xor     cx,cx           ;hlt
        call    readarray
        xor     al,al
        stosb
        mov     al,255
        mov     cx,1
        retn
casef4  endp

casef5  proc    near
        mov     si,offset opercmc
        xor     cx,cx           ;cmc
        call    readarray
        xor     al,al
        stosb
```

```
        mov     al,1
        mov     cx,1
        retn
case7f  endp

case6to7f       proc    near
        call    cod3lllinc
        mov     cl,bl
        xor     ch,ch
        push    si
        mov     si,offset oper7
        call    readarray
        pop     si
        or      bl,bl
        jz      pondat
        mov     al,' '
        stosb
        mov     dx,offset atbstr
        mov     bstring,dx
        mov     al,an
        and     al,1
        call    movrs
        call    addbstr
        jmp     short fin6to7
pondat: mov     dl,ah
        and     dl,1
        call    datadic
fin6to7:
        push    ax
        xor     al,al
        stosb
        pop     ax
        retn
case6to7f       endp

case8tofd       proc    near
        mov     si,offset oper8
        mov     cl,ah
        and     cx,7
        call    readarray
        xor     al,al
        stosb
        mov     al,1
        mov     cx,1
        retn
case8tofd       endp

caseffeff       proc    near
        call    cod3lllinc
        push    si
        mov     si,offset oper9
        mov     al,ah
```

```
        and     al,1
        jnz     normal
        cmp     bl,2
        jb      normal
        mov     cx,7
        jmp     short finfetoff
normal: mov     cl,bl
        xor     ch,ch
        xor     dh,dh
        cmp     cl,3
        je      espec
        cmp     cl,5
        jne     finfetoff
espec:  mov     dh,2
finfetoff:
        call    readarray
        pop     si

        mov     dl,al
        mov     al,' '
        stosb
        mov     ax,offset atostr
        mov     bstring,ax
        mov     ax,dx
        call    modrm
        call    addbstr
        xor     al,al
        stosb
        mov     ax,dx
        or      ah,ah
        jz      same
        mov     al,ah
same:   retn
casefetoff      endp

        public  desensDet
desensDet       proc    near
;-------------------------------------------------------------------------
;Este procedimiento genera un string a partir del código de máquina apuntado
;por DS:SI, desde ES:DI, terminado en 0, consistente en:
;       - Cuatro caracteres que representan al valor inicial de SI en hex.
;       - Cinco caracteres del mnemotécnico de la instruccion.
;       - Veintiún caracteres de parámetros del mnemotécnico.
;Además, se retorna:
;       AL      tipo de instrucción (0,1,2 o 255)
;       CX      longitud de instrucción
;-------------------------------------------------------------------------
        push    di
        cld
        mov     ax,si
        push    ax
        mov     al,ah
```

```
        call    convhex
        pop     ax
        call    convhex
        call    desens
        push    ax
        push    bx
        push    cx
        push    di
        push    si
        push    es
        mov     bx,di           ;DI=comienzo del anemotecnico
        mov     cx,6            ;busco un 0 hasta el sexto carácter
        mov     al,0
        repne   scasb
        jne     ConParam        ;si no se lo encuentra, hay parámetros
        add     cx,21           ;poner 21 espacios al final del unico anemo
        mov     al,' '
        dec     di
        rep     stosb
        mov     al,0            ;poner un 0 al final
        stosb
        jmp     FinDesDet
ConParam:
        mov     di,bx           ;recuperar el comienzo del anemotécnico
        mov     cx,6            ;buscar un espacio
        mov     al,' '
        repne   scasb
        cmp     cx,1            ;en que posición esta el espacio?
        jb      sexta           ;si CX=0, sexta posición
        je      PonSpcFin       ;si CX=1, quinta posición
        mov     cx,di
        mov     ax,es           ;ci CX>=2, cuarta o menos.
        mov     ds,ax           ;en este caso, mover los parámetros a la
        sub     di,bx           ;derecha
        mov     dx,5
        sub     dx,di           ;DX=número de posiciones por mover a la derecha
        mov     di,bx
        mov     bx,cx           ;BX=comienzo de los párametros
        mov     cx,40
        mov     al,0
        repne   scasb           ;buscar el fin de string
        dec     di
        mov     si,di
        add     di,dx
        std                     ;dirección inversa
CicDer: lodsb
        stosb
        cmp     si,bx           ;se llego al comienzo de los parámetros?
        jae     CicDer          ;mover el string
        mov     al,' '
CicSpc: cmp     di,si
        je      PonSpcFin
```

```
;!!!!!!!!!!!!!!!!PROGRAMA ENSAMBLADOR!!!!!!!!!!!!!!!!!!
;!!!!!!!!!!!!!!!!!!POR IVAN ORDOREZ!!!!!!!!!!!!!!!!!!!!!

        .model  small

        .data

transit db      32 dup (?)

enemo   db      'ADd',0,'PUSh',6,'POp',7,'Or',8
        db      'ADc',10H,'SBb',18H,'ANd',20H,'DAa',27H
        db      'SUb',28H,'DAs',2FH,'XOr',30H,'AAa',37H
        db      'CMp',38H,'NEAr',39H,'FAr',3AH,'PTr',3BH
        db      'BYTe',3CH,'WORd',3DH,'DWORd',3EH,'AAs',3FH
        db      'INc',40H,'DEc',48H,'SHORt',50H,'GWORd',51H,'NOt',52H,'NEg',53H
        db      'MUl',54H,'IMUl',55H,'DIv',56H,'IDIv',57H

        db      'JNAe',72H,'JNb',73H,'Jb',72H,'JAe',73H
        db      'Jc',74H,'JNz',75H,'JNe',76H,'JNBe',77H
        db      'JPe',7AH,'JPo',7BH,'JNBe',7CH,'JNl',7DH
        db      'JNg',7EH,'JNLe',7FH,'Jo',70H,'JNo',71H
        db      'Jc',72H,'JNc',73H,'Je',74H,'JNe',75H
        db      'JBe',76H,'Ja',77H,'Js',78H,'JNs',79H
        db      'Jo',7AH,'JNp',7BH,'Jl',7CH,'JGe',7DH
        db      'JLe',7EH,'Jg',7FH,'Db',80H,'Dw',81H

        db      'TESt',84H,'XCHg',86H,'MOv',88H,'LEa',8DH
        db      'NOp',90H,'CBw',98H,'CWd',99H,'CALl',9AH,'WAIt',9BH
        db      'PUSHf',9CH,'POPf',9DH,'SAHf',9EH,'LAHf',9FH
        db      'MOVSb',0A4H,'MOVSw',0A5H,'CMPSb',0A6H,'CMPSw',0A7H
        db      'STOSb',0AAH,'STOSw',0ABH,'LODSb',0ACH,'LODSw',0ADH
        db      'SCASb',0AEH,'SCASw',0AFH,'REt',0C2H,'RETn',0C2H
        db      'LEa',0C4H,'LDs',0C5H,'RETf',0CAH,'INt',0CCH
        db      'INTo',0CEH,'IREt',0CFH,'ROl',0D0H,'ROr',0D1H

        db      'RCl',0D2H,'RCr',0D3H,'AAm',0D4H,'AAd',0D5H
        db      'XLAt',0D7H,'ESc',0D8H,'SHl',0DCH,'SHr',0DDH
        db      'SAl',0DCH,'SAr',0DFH,'LOOPNe',0E0H,'LOOPNz',0E0H
        db      'LOOPe',0E1H,'LOOPz',0E1H,'LOOp',0E2H,'JCXz',0E3H
        db      'In',0E4H,'OUt',0E6H,'JMp',0EAH,'LOCk',0F0H
        db      'REPNe',0F2H,'REPNz',0F2H,'REp',0F3H,'REPe',0F3H,'REPz',0F3H
        db      'HLt',0F4H,'CMc',0F5H,'CLc',0F8H,'STc',0F9H
        db      'CLi',0FAH,'STi',0FBH,'CLd',0FCH,'STd',0FDH
        db      0FFH

regs    db      'ALCLDLBLAHCHDHBH'
        db      'AXCXDXBXSPBPSIDI'

sregs   db      'ESCSSSDS'

loners  db      27H,2FH,37H,3FH,90H,98H,99H,9BH,9CH,9DH,9EH,9FH
        db      0A4H,0A5H,0A6H,0A7H,0AAH,0ABH,0ACH,0ADH,0AEH,0AFH
```

```
        db      0CEH,0CFH,0D4H,0D5H,0D7H
        db      0F4H,0F5H,0F8H,0F9H,0FAH,0FBH,0FCH,0FDH

cmers   db      6,7,46H,47H,52H,53H,54H,55H,56H,57H
        db      70H,71H,72H,73H,74H,75H,76H,77H
        db      78H,79H,7AH,7BH,7CH,7DH,7EH,7FH
        db      9AH,0C2H,0DAH,0DCH,0E0H,0E1H,0E2H,0E3H,0EAH

        .code

NEsp    equ     10
delspc  proc    near
;---------------------------------------------------------------
;Este procedimiento salta los espacios y los TABs en el string por
;ensamblar y deja SI apuntando al primer byte que no sea un espacio.
;
;Entrada:       DS:SI   apunta al punto del string desde el cual
;                       se desea proceder.
;Salida:        DS:SI   apunta al primer byte que no es un espacio
;               CY      es 1 si hay mas de NEsp espacios (error), en
;                       cuyo caso DS:SI no es alterado
;---------------------------------------------------------------
        push    ax
        push    cx
        push    bp
        mov     cx,NEsp                 ;máximo NEsp espacios
        mov     bp,si
busc:   lodsb
        or      al,al
        jz      errdel                  ;si se llega al final del string
        cmp     al,' '                  ;hay un espacio?
        je      spac                    ;si, saltarlo
        cmp     al,9                    ;hay un TAB?
        jne     encont                  ;no, hecho
spac:   loop    busc                    ;saltar el espacio y ver sig. car.

errdel: mov     si,bp
        stc
        jmp     short findl
encont: dec     si                      ;apuntar al carácter válido
        clc
findl:  pop     bp
        pop     cx
        pop     ax
        retn
delspc  endp

encmem  proc    near
;---------------------------------------------------------------------
;Este procedimiento encuentra el pseudocódigo del mnemotécnico que está al
;comienzo del string por ensamblarse.
;
```

```
;Entrada:       DS:SI apunta al string, que debe terminar en 0.
;               ES:DI es la direccion en donde se pondrá el pseudocódigo.
;
;Salida:        DS:SI apunta al final del mnemotecnico + 2 en el string si
;                    este fue encontrado (pues debe terminar con espacio)
;               ES:DI apunta al final del pseudocodigo + 1
;               CY     es 1 si el mnemotécnico es inválido o si hay error
;                    como en DEL_SPC , en cuyo caso DS:SI y ES:DI no son
;                    alterados (excepto si habia espacios desde DS:SI)
;               AX     contiene el pseudocódigo (AL=0, AH= cod.)
;-------------------------------------------------------------------------
        call    delsec          ;encontrar el mnemotécnico
        jc      finenc2         ;en caso de error, terminar
        push    bx
        push    ds
        push    es
        push    di
        mov     ax,@data
        mov     es,ax           ;hacer que ES:DI apunte a los mnemotécnicos
        mov     di,offset mnem
        mov     dx,si           ;preservar el comienzo del mnemotecnico

busca:  lodsb                   ;carácter del string en AL
        mov     ah,es:[di]      ;carácter de la tabla en AH
        cmp     ah,'z'          ;es una minuscula, implicando fin de mnemo?
        jb      nofinmnemo      ;no
        cmp     ah,0ffh         ;es el fin de la tabla?
        jne     uff             ;no, probar el siguiente mnemo de la tabla
        stc                     ;si, el código era inválido pues no
        mov     si,dx           ;constaba en la tabla, asi que se debe generar
        jmp     short finenc    ;un bit de error
uff:    mov     bl,1            ;indicar fin de mnemo haciendo BL = 1
        and     ah,0dfh         ;hacer mayúscula a AH
        jmp     short sigcomp   ;comparar el ultimo carácter
nofinmnemo:
        xor     bl,bl           ;indicar que todavia no hay fin de mnemo
sigcomp:and     al,0dfh         ;hacer mayúscula a AL
        cmp     al,ah           ;son iguales los dos caracteres?
        jne     noeseste        ;no, intentar con el siguiente mnemo
        or      bl,bl           ;si, era el ultimo carácter del mnemo?
        jz      puedeser        ;no, seguir comparando
        lodsb                   ;comparación exitosa, verificar que
        cmp     al,'['          ;haya un corchete,
        je      escorch
        cmp     al,' '          ;un espacio, TAB o 0  despues del mnemo
        ja      noeseste        ;si no, intentar con el siguiente mnemo
        jmp     short sieseste
escorch:
sieseste:                       ;en caso de corchete, debe dejarse SI
        dec     si              ;apuntando a este
        mov     ah,es:[di+1]    ;encontrado, obtener el pseudocódigo
        pop     di
```

```
        loop    srchreg
        stc                     ;si no se lo encontró, error
        jmp     short errreg
foundreg:
        mov     al,1ch
        sub     al,cl           ;encontrar el código del registro
        or      al,20h
        clc
errreg: pop     di
        pop     es
finencreg:
        pop     dx
        pop     cx
finencreg1:
        retn
encreg  endp

        public  encsreg
encsreg proc    near
;--------------------------------------------------------------
;Este procedimiento encuentra el código del registro de segmento
;al que apunta DS:SI.
;
;Entrada:       DS:SI apunta al ascii del registro de segmento
;
;Salida:        DS:SI es alterado según DELSPC
;               AH    es alterado
;               AL    código del registro
;               CY    1 si no es un registro o según DELSPC
;--------------------------------------------------------------
        call    delspc
        jc      finencsreg1     ;si hay demasiados espacios, fin
        push    cx
        push    es
        push    di
        mov     ax,@data
        mov     es,ax
        mov     di,offset sregs ;apuntar ES:DI a la tabla de registros de seg
        mov     ax,[si]         ;ascii del reg en AX
        and     ax,0dfdfh       ;en mayúsculas
        mov     cx,4
srchsreg:
        cmp     ax,es:[di]
        je      foundsreg
        inc     di
        inc     di
        loop    srchsreg
malsreg:stc                     ;si no se lo encontró, error
        jmp     short errsreg
foundsreg:
        mov     al,[si+2]       ;tercer carácter
        and     al,0dfh         ;hacerlo mayúscula
```

```
        cmp     al, A           ;si es una letra, mal
        jb      okreg
        cmp     al,'Z'
        jbe     malsreg
okreg:  mov     al,6
        sub     al,cl           ;encontrar el código del registro
        or      al,40h
        clc
errsreg:pop     di
        pop     es
finencsreg:
        pop     cx
finencsreg1:
        retn
encsreg endp

ascahex proc    near
;-------------------------------------------------------------------------
;Este procedimiento transforma el string en AX (con AL = MSD) a un byte
;en AL. CY = 1 si no hay tal hex en AX.
;-------------------------------------------------------------------------
        push    cx
        cmp     ah, 'f'
        ja      errhex
        cmp     ah,'a'
        jb      tstal
        and     ah,0dfh         ;transformar el carácter en AH a mayúscula
tstal:  cmp     al,'f'
        ja      errhex
        cmp     al,'a'
        jb      trns
        and     al,0dfh         ;transformar el carácter en AL a mayúscula
trns:   cmp     ah,'A'          ;ajustar el carácter en AH si es alfabético
        jb      numah
        sub     ah,'A'-'9'-1
numah:  cmp     al,'A'          ;lo mismo con AL
        jb      numal
        sub     al,'A'-'9'-1
numal:  sub     ax,'00'         ;pasar de ascii a numérico
        cmp     al,16
        jae     errhex
        cmp     ah,16
        jae     errhex
        mov     cl,4
        shl     al,cl
        or      al,ah           ;condensar el número
        clc
        jmp     short finhex
errhex: stc
finhex: pop     cx
        retn
ascahex endp
```

```
esletr? proc    near
;-----------------------------------------------------------------------
;Este procedimiento determina si el caracter contenido en AL es un digito
;o una letra, en cuyo caso CY = 0, y de lo contrario, CY = 1.
;AL es alterado.
;-----------------------------------------------------------------------
        cmp     al, 0
        jb      noeslt
        cmp     al,'9'
        jbe     sieslt
        and     al,0dfh         ;mayúscula
        cmp     al,'A'
        jb      noeslt
        cmp     al,'Z'
        jbe     sieslt
noeslt: stc
        retn
sieslt: clc
        retn
esletr? endp

        public  eshex?
eshex?  proc    near
;-----------------------------------------------------------------------
;Este procedimiento determina si el string al que apunta DS:SI es un numero
;hexadecimal, en cuyo caso encuentra su valor.
;
;Entrada:       DS:SI apunta al string, excluyendo signos +, - o espacios
;
;Salida:        AL  = 16 si el número es de 1 o 2 digitos, 32 en otro caso
;               AH  = longitud del string del número
;               BX  = valor del número
;               CY  = 1 si lo apuntado por DS:SI no es un número o si es un
;                       número de mas de 32 bits
;-----------------------------------------------------------------------
        push    dx
        xor     bx,bx
trycero:cmp     byte ptr[si+bx],'0' ;buscar ceros iniciales
        jne     noceroin
        inc     bx              ;contarlos en BX
        jmp     short trycero
noceroin:
        mov     dh,bl           ;número de ceros en DH
        or      bx,bx
        jz      noproblem       ;si no hay ceros iniciales
        mov     ah,[si+bx]
        mov     al,'0'
        call    ascahex         ;si el string era solo ceros, ya no debe
        jnc     noproblem       ;haber mas caracteres hexadecimales válidos
        mov     ax,16
        xor     bx,bx           ;salir con cero
```

```
        jmp     finnum
noprobleg:
        mov     al,[si+bx+1]
        call    esletr?
        jc      undig
        mov     al,[si+bx+2]
        call    esletr?
        jc      dosdig
        mov     al,[si+bx+3]
        call    esletr?
        jc      tresdig

        mov     al,[si+bx+4]
        call    esletr?                 ;tiene mas de 4 digitos?
        jc      N4Dig                   ;no, OK
        stc                             ;si, error
        jmp     short errnum
N4Dig:  mov     ax,[si+bx]              ;el numero tiene 4 digitos
        call    ascahex
        jc      errnum
        mov     dl,al
        mov     ax,[si+bx+2]
        call    ascahex
        jc      errnum
        mov     bl,al
        mov     bh,dl
        mov     ax,420h         ;AL=32, AH=4
        jmp     short finnum
tresdig:mov     al,'0'          ;número de tres digitos
        mov     ah,[si+bx]
        call    ascahex
        jc      errnum
        mov     dl,al
        mov     ax,[si+bx+1]
        call    ascahex
        jc      errnum
        mov     bl,al
        mov     bh,dl
        mov     ax,320h         ;AL=32, AH=3
        jmp     short finnum
dosdig: mov     ax,[si+bx]              ;número de dos digitos
        call    ascahex
        jc      errnum
        mov     bl,al
        xor     bh,bh
        mov     ax,210h         ;AL=16, AH=2
        jmp     short finnum
undig:  mov     al,'0'
        mov     ah,[si+bx]
        call    ascahex
        jc      errnum
        mov     bl,al
```

```
        xor     cn,cn
        mov     ax,1100h        ;AL=1b, Ax=1
finnum: add     cn,cn           ;incluir los ceros en la longitud
        cmp     ch,2
        jae     longbyt
        mov     cl,20h
longbyt:clc
errnum: pop     cx
        retn
eshex?  endp

esreg?  proc    near
;---------------------------------------------------------------------
;Este procedimiento es similar al anterior: encuentra si el string al que
;apunta DS:SI es uno de los registros BX, BP, SI o DI.
;
;Entrada:       DS:SI apunta al string excluyendo signos (+ o -) y espacios
;
;Salida:        CY =1 si hay error, caso contrario:
;               AH = 2 (longitud obvia del string)
;               AL = 1 si SI
;                    2 si DI
;                    4 si BX
;                    8 si BP
;---------------------------------------------------------------------
        mov     ax,[si]
        and     ax,0dfdfh       ;transformar string a mayúsculas
        cmp     ax,'IS'         ;registro SI?
        jne     nosi            ;no
        mov     ax,201h         ;si, codificar
        jmp     short bienreg
nosi:   cmp     ax,'ID'         ;registro DI?
        jne     nodi            ;no
        mov     ax,202h         ;si, codificar
        jmp     short bienreg
nodi:   cmp     ax,'XB'         ;registro BX?
        jne     nobx            ;no
        mov     ax,204h         ;si, codificar
        jmp     short bienreg
nobx:   cmp     ax,'PB'         ;registro BP?
        jne     nobp            ;no
        mov     ax,208h         ;si,codificar
bienreg:clc
        jmp     short finesreg
nobp:   stc
finesreg:
        retn
esreg?  endp

encmem  proc    near
;---------------------------------------------------------------------
;Este procedimiento decodifica un string de la forma ES:[SI+BX-3], por ej.
```

```
;
;Entrada:       DS:SI direccion del string
;               ES:DI direccion donde se quiere el código
;
;Salida:        DS:SI apunta al final de dicho string + 1
;               ES:DI es incrementado en 4 si hay exito
;               CY    set si hay error, en cuyo caso DS:SI y ES:DI son
;                     inalterados
;------------------------------------------------------------------------
        push    ax
        push    bx
        push    cx
        push    dx
        push    bp
        mov     bp,si
        call    encsreg         ;comienza con un sreg?
        jc      noopref         ;no
        inc     si              ;si, pasarlo
        inc     si
        or      al,4            ;codificarlo
        mov     cl,4
        shl     al,cl
        mov     dh,al           ;en dh
        call    delspc
        jc      errml
        lodsb
        cmp     al,':'          ;hay los dos puntos reglamentarios?
        jne     errml           ;no
        call    delspc
        jnc     corchetes
errml:  jmp     errmem
nooref: xor     dh,dh           ;codificar inexistencia de sreg
corchetes:
        xor     dl,dl           ;inicializar primera parte del código
        xor     bx,bx           ;inicializar desplazamiento
        mov     cx,bx
        lodsb
        cmp     al,'['          ;el carácter actual debe ser el
        jne     errml           ;corchete reglamentario
        call    delspc
        jc      errml
        cmp     byte ptr[si],'+';inicia con un signo +?
        jne     tstmin          ;no
        inc     si              ;si, ignorarlo
        call    delspc
        jc      errml
analiz: call    eshex?          ;hay un número hex?
        jc      nohl            ;no.
        mov     cx,bx           ;si, preservar su valor
        jmp     short rdynum
nohl:   call    esreg?          ;hay un registro reglamentario?
        jnc     rdynum          ;si
```

```
endmac  endp

enchex  proc    near
;---------------------------------------------------------------------------
;Este procedimiento determina si lo apuntado por DS:SI es un número
;hexadecimal, en cuyo caso encuentra su valor y lo coloca en ES:DI después
;del código correspondiente. El criterio para considerar al dato como un byte
;o un word no es su valor, sino la longitud del string que lo representa: si
;es de 3 o más caracteres, es un word, p. ej 0001.
;
;Entrada:       DS:SI   apunta al string que representa al número
;               ES:DI   apunta a donde debe ir el código
;
;Salida:        DS:SI   es ajustado
;               ES:DI   es ajustado
;               CY      set si hay error, en cuyo caso DS:SI y ES:DI no son
;                       alterados
;---------------------------------------------------------------------------
        push    ax
        push    bx
        push    dx
        call    delspc
        jc      finenchex
        xor     dx,dx           ;positivo por default
        mov     al,[si]
        cmp     al,'+'
        jne     nomas
        inc     si
        call    delspc
        jc      finenchex
        jmp     short decode
nomas:  cmp     al,'-'
        jne     decode
        not     dl              ;marcar negativo
        inc     si
        call    delspc
        jc      finenchex
decode: call    eshex?
        jc      finenchex
        or      dl,dl           ;es negativo?
        jz      poshex          ;no
        neg     bx              ;si
poshex: mov     dl,ah           ;longitud del string del número
        add     si,dx           ;actualizar SI
        mov     al,80h          ;código de dato de un byte
        cmp     ah,2            ;longitud de 2 o menos?
        jbe     unbyte          ;si
        or      al,20h          ;no, poner código de word
        stosb
        mov     ax,bx
        stosw
        jmp     short noerrhex
```

```
unoyte: mov     ah,bl
        stosw
noerrnex:
        clc
finenchex:
        pop     dx
        pop     bx
        pop     ax
        retn
enchex  endp


ensoaso1        proc    near
;-----------------------------------------------------------------------
;Este procedimiento ejecuta el primer paso de compilación, es decir, la
;traducción de un string alfanumérico a seudocódigos que serán transformados
;a código verdadero por otro procedimiento.
;
;Entrada:       DS:SI   apunta al string por compilarse, que debe terminar
;                       en 0.
;               ES:DI   apunta al lugar donde se desea el seudocódigo
;
;Salida:        DS:SI   es alterado
;               ES:DI   es alterado
;               CY      set si hay error
;-----------------------------------------------------------------------
        push    ax
        push    bx
        push    cx
        push    dx
        call    encsreg         ;es la instrucción un prefijo como ES: ?
        jc      nosregpref      ;no
        inc     si              ;si, proceder en consecuencia
        inc     si
        call    delsoc
        mov     ah,al           ;preservar código
        lodsb
        cmp     al,':'          ;por sintaxis, debe haber :
        jne     errpaso1
        mov     al,ah
        stosb
        jmp     short bienpaso1
nosregpref:
        call    encnem          ;lo primero debe ser un mnemotécnico
        jc      errpaso1
        cmp     ah,0f0h         ;es un prefijo REP o LOCK?
        jb      nopref1r
        cmp     ah,0f3h
        ja      nopref1r
        call    delspc          ;si, buscar siguiente mnemotécnico
        jc      bienpaso1       ;si no hay mas que el prefijo, terminar
        call    encnem
```

```
        jc      errpaso1
        cmp     ah,0f0h         ;es un segundo prefijo REP o LOCK?
        jz      noprefir
        cmp     ah,0f3h
        je      noprefir
        call    delspc          ;si, buscar siguiente mnemotécnico
        jc      bienpaso1       ;si no hay mas que el segundo prefijo, terminar
        call    encmem
        jc      errpaso1
        cmp     ah,0f0h         ;no puede haber un tercer prefijo
        jz      noprefir
        cmp     ah,0f3h
        jbe     errpaso1
noprefir:
        cmp     ah,39h          ;el mnemotécnico inicial no puede ser NEAR,
        jb      bienmne         ;PTR, BYTE, etc
        cmp     ah,3eh
        jbe     errpaso1
        cmp     ah,50h
        je      errpaso1
        cmp     ah,51h
        je      errpaso1

bienmne:mov     dx,ds           ;preservar DS
        mov     bx,@data
        mov     ds,bx
        mov     bx,offset loners
        mov     cx,33           ;averiguar si el mnemotécnico es uno de los
lone?:  cmp     ah,[bx]         ;que no aceptan argumentos
        je      esloner         ;si
        inc     bx
        loop    lone?
        jmp     short hayarg
esloner:mov     ds,dx
        mov     byte ptr es:[di-2],1    ;marcarlo como loner
                                ;apuntar al punto inmediatamente despues
        call    delspc          ;del mnemotécnico y buscar el fin de string
        jnc     errpaso1        ;si hay mas caracteres válidos, error

bienpaso1:
        mov     al,0ffh
        stosb
        clc
        jmp     short finpaso1
errpaso1:
        stc
finpaso1:
        pop     dx
        pop     cx
        pop     bx
        pop     ax
        retn
```

```
navarg: mov     ax,offset oners
        mov     cx,35           ;averiguar si el onemotécnico es uno de los
one?:   cmp     ah,[bx]         ;que aceptan un solo argumento
        je      esoner          ;si
        inc     bx
        loop    one?
        jmp     short dosarg
esoner: mov     byte ptr es:[di-2],3    ;marcarlo como oner
        cmp     ah,9ah          ;CALL?
        je      admitedp        ;si, puede haber dos puntos
        cmp     ah,0eah         ;JMP?
        jne     solouno         ;no
admitedp:
        or      byte ptr es:[di-2],4    ;marcarlo como JMP o CALL
        mov     cl,2            ;marcar que puede haber un argumento doble
        jmp     short args      ;de dirección absoluta
solouno:
        mov     cl,1            ;marcar que debe haber solo un argumento
        jmp     short args
dosarg: xor     cl,cl           ;marcar que debe haber dos argumentos
args:   mov     ds,dx
        call    delspc
        jnc     enarg           ;confirmar presencia de argumentos
        cmp     ah,0c0h         ;su no presencia solo es admisible con
        je      bienpaso1       ;RET, RETN o RETF
        cmp     ah,0c2h
        je      bienpaso1
        cmp     ah,0cah
        je      bienpaso1
        jmp     short errpaso1
enarg:  call    encmem          ;buscar argumentos
        jnc     segarg
        call    encreg
        jc      noreg1
        stosb
        inc     si
        inc     si
        jmp     short segarg
noreg1: call    encsreg
        jc      nosreg1
        stosb
        inc     si
        inc     si
        jmp     short segarg
nosreg1:call    enchex
        jnc     segarg
        call    encmem          ;buscar por ejemplo BYTE PTR
        jc      errpaso1
        call    encmem          ;el PTR es opcional
        call    encmem
        jnc     segarg
```

```
        call    encnex          ;puede también ser un número (para saltos)
        jc      errpaso1
segarg: or      ci,ci           ;dos argumentos?
        jz      hayces          ;si
        call    delsoc          ;no, uno solo
        jc      bien1
        cmp     ci,2            ;era JMP o CALL?
        jne     errp1           ;no
        lodsb                   ;solo se aceptan dos argumentos aqui si estan
        cmp     al,':'          ;separados por :
        jne     errp1
        call    encnex
        jc      errp1
        jmp     bienpaso1

hayces: call    delsoc
        jc      errp1           ;si hay un solo argumento, error
        lodsb
        cmp     al,','          ;el separador entre argumentos es la coma
        jne     errp1
        call    encmem          ;buscar segundo argumento
        jnc     bien1
        call    encreg
        jc      noreg2
        stosb
        inc     si
        inc     si
bien1:  jmp     bienpaso1
noreg2: call    encsreg
        jc      nosreg2
        stosb
        inc     si
        inc     si
        jmp     bienpaso1
nosreg2:call    enchex
        jnc     bien1
        call    encmem          ;buscar por ejemplo BYTE PTR
        jc      errp1
        call    encmem          ;el PTR es opcional
        call    encmem
        jnc     bien1
errp1:  jmp     errpaso1
enspaso1        endp


tipo    proc    near
;-----------------------------------------------------------------------
;Este procedimiento determina el tipo de pseudocódigo al que apunta DS:SI y
;proporciona información útil sobre dicho código.
;
;Entrada:       DS:SI   apunta al inicio del pseudocódigo
;
```

```
;Salida:        DS:SI   apunta al siguiente pseudocódigo
;               CY      1 ssi encontró un FFh, fin de instrucción
;
;               DL =    0 si es un mnemotécnico, en cuyo caso
;                       AL = código del mnemotécnico
;                       AH = información sobre loners, oners, jmps y calls
;
;               DL =    1 si es un registro, en cuyo caso
;                       AL = 0000wReg
;
;               DL =    2 si es un registro de segmento, en cuyo caso
;                       AL = 000000Sr
;
;               DL =    3 si es un Mem, en cuyo caso
;                       AX = desplazamiento
;                       BL = Md000Mem
;                       BH = xPSrxxxx con P = presencia de SReg
;
;               DL =    4 si es un dato de byte, en cuyo caso
;                       AL = dato
;                       AH = 0
;
;               DL =    5 si es un dato de word, en cuyo caso
;                       AX = dato
;-----------------------------------------------------------------------------
        push    cx
        lodsb
        cmp     al,0ffh         ;fin de instrucción?
        je      fininst
        mov     ch,ah
        mov     ah,al
        mov     cl,5
        shr     al,cl           ;obtener el tipo
        mov     dl,al           ;en DL
        jz      esinstr
        dec     al
        jz      esregis
        dec     al
        jz      essregi
        dec     al
        jz      esmemor
        dec     al
        jz      esdatab
        lodsw                   ;es dato de word
        jmp     short rdytipo
esdatab:lodsb                   ;es dato de byte
        xor     ah,ah
        jmp     short rdytipo
esinstr:lodsb                   ;es mnemotécnico
        jmp     short rdytipo
esregis:mov     al,ah           ;es registro
        and     al,0fh
```

```
        mov     ah,ch
        jmp     short rdvtipo
essregi:mov     al,ah           ;es registro de segmento
        and     al,7
        mov     ah,ch
        jmp     short rdvtipo
essmemori:mov   al,ah           ;es memoria
        mov     cl,3
        shl     al,cl
        and     al,0c0h
        and     ah,7
        or      ah,al
        lodsb
        xchg    al,ah
        mov     bx,ax
        lodsw
        mov     cx,4
rdvtipo:clc
        jmp     short fintipo
fininst:stc
fintipo:pop     cx
        ret
tipo    endp

getpref proc    near
;-----------------------------------------------------------------------
;Este procedimiento obtiene el prefijo de redefinición de segmento a partir
;del numero de registro de segmento.
;
;Entrada:       BH      = xP5rxxxx con P = presencia de prefijo
;
;Salida:        AH      = 0 si no hay prefijo, o
;                       prefijo
;-----------------------------------------------------------------------
        xor     ah,ah
        test    bh,40h
        jz      fingtp
        mov     ah,bh
        shr     ah,1
        and     ah,18h
        or      ah,26h
fingtp: retn
getpref endp

pref?   proc    near
;-----------------------------------------------------------------------
;Este procedimiento determina si debe incluirse un prefijo de redefinición
;de segmento antes del opcode, del cual solo debe haber 1 byte, y ES:DI
;debe estar apuntando a la dirección siguiente a dicho byte. Hace la
;inclusión y los ajustes del caso de ser necesario.
;
;Entrada:       AH      0 si no debe haber prefijo
```

```
;                       opcode del prefijo en caso contrario
;               ES:DI   dirección del byte de opcode + 1
;
;Salida:        AL      es inalterado o contiene el opcode
;               DI      es ajustado
;               CX      incrementado en 1 si hay prefijo
;-------------------------------------------------------------------------
        cmp     en,0            ;hay prefijo?
        je      noprefijo
        mov     al,en           ;si, ajustar
        xchg    al,es:[di-1]
        stosb
        inc     cx
noprefijo:
        retn
pre?    endp

sizedisp        proc    near
;-------------------------------------------------------------------------
;Determina dados los bits 6 y 7 de AL, que debe tener el byte de modregr/m,
;el tamano en bytes del desplazamiento u offset. Luego incluye este
;desplazamiento en el opcode.
;
;Entrada:       DL o DX desplazamiento
;               AL      byte de modo de direccionamiento
;               ES:DI   apunta al lugar del opcode donde va el desplazamiento
;
;Salida:        AX      es alterado
;               ES:DI   es ajustado
;               CX      incrementado segun el desplazamiento
;-------------------------------------------------------------------------
        test    al,0c0h         ;obtener el mod
        jz      despl?          ;si es 00, averiguar si hay desplazamiento
        and     al,0c0h
        cmp     al,0c0h         ;es 11?
        je      nodesp          ;si, tampoco hay desplazamiento
        inc     cx
        cmp     al,40h          ;es 01?
        jne     dosbyt          ;no
        mov     al,dl           ;si, desplazamiento de un byte
        stosb
        jmp     short nodesp
despl?: and     al,7            ;si mod = 00 y r/m = 110,
        cmp     al,6            ;el modo es directo
        jne     nodesp
dosbyt: mov     ax,dx           ;desplazamiento de dos bytes
        stosw
        inc     cx
nodesp: retn
sizedisp        endp

sizedata        proc    near
```

```
;---------------------------------------------------------------------------
;Similar al anterior: determina el tamaño del dato segun el bit 0 de AL, y lo
;coloca en el opcode.
;
;Entrada:       AL      su bit 0 es w
;               BP      dato
;               ES:DI   apunta al lugar del opcode donde debe ir el dato
;
;Salida:        AX      es alterado
;               DI      es ajustado
;               CX      incrementado segun los datos
;---------------------------------------------------------------------------
        inc     cx
        test    al,1            ;w = 1?
        mov     ax,bp
        jnz     dosdata         ;si
        stosb                   ;no
        jmp     short fsdata
dosdata:inc     cx
        stosw
fsdata: retn
sizedata        endp

modrm   proc    near
;---------------------------------------------------------------------------
;Este procedimiento analiza el pseudocódigo de modo de direccionamiento y
;genera datos necesarios para la compilación.
;
;Entrada:       DS:SI   debe apuntar al pseudocódigo de modo de dir.
;                                                    —
;Salida:        AH      prefijo de redefinición, de haberlo, o 0
;               BL      = Md000r/m
;               BH      = h000000w; h=validez de w
;               DX      desplazamiento, de haberlo; si no, es alterado
;               CL      incrementado en 1 si no hay error
;               CH      = 0
;               CY      set si hay error, en cuyo caso los registros
;                       quedan inalterados
;---------------------------------------------------------------------------
        push    bx
        push    cx
        push    dx
        push    si
        push    bp
        push    ax
        lodsb
        dec     si
        test    al,0e0h         ;mnemo?
        jnz     mrmnemo         ;no
        call    tipo
        jc      errmodrm
        cmp     al,3ch          ;BYTE?
```

```
        jz      errmodrm
        je      bytptr
        cmp     al,3eh          ;WORD o DWORD?
        ja      errmodrm
        mov     dh,81h          ;marcar word
        jmp     short eiptr
bytptr: mov     dh,80h          ;marcar byte
eiptr:  call    getptr          ;saltar PTR si existe
        jmp     short procrm
errnoesa:
        xor     dh,dh           ;marcar size indefinido
procrm:call     tipo
        jz      errmodrm
        cmp     di,3            ;mem?
        je      rmmem
        cmp     di,1            ;reg?
        jne     errmodrm
        mov     bh,dh
        test    bh,80h          ;size definido?
        jz      noesreg
        shl     dh,1            ;si, contradictorio con size del reg?
        shl     dh,1
        xor     dh,al
        test    dh,8
        jnz     errmodrm        ;si
        jmp     short rdysiz    ;no
noesreg:
        mov     bh,al
        shr     bh,1
        shr     bh,1
        shr     bh,1
        or      bh,80h          ;crear definicion
rdysiz: and     al,7            ;aislar r/m = reg
        or      al,0c0h         ;mod = 11
        mov     bl,al
        xor     ah,ah           ;sin prefijo
        jmp     short bienmodrm
rmmem:  mov     bp,ax           ;desplazamiento
        call    getpref         ;pref en AH
        mov     bh,dh
        mov     dx,bp
bienmodrm:
        inc     cx
        mov     ch,ah
        pop     ax              ;restaurar AL
        mov     ah,ch
        xor     ch,ch
        pop     bp
        add     sp,8
        clc
        retn
```

```
errmodrm:
        pop     ax
        pop     bp
        pop     si
        pop     dx
        pop     cx
        pop     bx
        stc
        retn
modrm   endp

modregrm        proc    near
;-----------------------------------------------------------------------
;Este procedimiento genera datos necesarios para el ensamblado de instrucciones
;que tienen dos operadores (como ADD), dados los pseudocódigos generados en el
;primer paso de compilación.
;
;Entrada:       DS:SI   debe apuntar a los pseudocódigos de los operadores.
;
;Salida:        AL      = 000000dw
;               AH      prefijo de redefinición, de haberlo, o 0
;               BL      byte de modo de direccionamiento
;               DX      desplazamiento de haberlo, si no, es alterado
;               CX      incrementado en 1 si no hay error
;               CY      set si hay error, en cuyo caso los registros quedan
;                       inalterados
;-----------------------------------------------------------------------
        push    ax
        push    cx
        push    dx
        push    si
        push    bp
        push    bx
        lodsb
        dec     si
        and     al,0e0h
        cmp     al,20h          ;reg?
        jne     mrmnoreg
        call    tipo
        call    modrm
        jc      errmrr
        call    arregla
        jc      errmrr
        or      al,2            ;d=1
        jmp     short bienmrr
mrmnoreg:
        call    modrm
        jc      errmrr
        mov     bp,dx
        call    tipo
        jc      errmrr
        cmp     dl,1            ;mem?
```

```
        jne     errarr
        mov     dx,bp
        call    arregla
        jc      errarr
diamarr:mov     ch,bl
        pop     bx
        mov     cl,cs
        xor     ch,cs
        pop     ax
        adc     sr,9
        clc
        retn
errarr: pop     dx
        pop     bp
        pop     si
        pop     dx
        pop     cx
        pop     ax
        stc
        retn
arregla:test    bh,80h          ;especificado size?
        mov     bp,ax
        jz      mrrnoesp
        shr     al,1
        shr     al,1
        shr     al,1
        xor     al,bh
        test    al,1            ;incompatibilidad?
        jnz     errarr
        mov     ax,bp
mrrnoesp:
        mov     bh,al
        and     bh,8
        shr     bh,1
        shr     bh,1
        shr     bh,1
        and     al,7
        shl     al,1
        shl     al,1
        shl     al,1
        or      bl,al
        mov     al,bh
        clc
        retn
errarr: stc
        retn
modregrm        endp

modrmdata       proc    near
;-----------------------------------------------------------------------
;Similar al anterior, cuando el segundo operador es un dato inmediato. Produce
;resultados diferentes si el primer operador es un acumulador.
```

```
        retn
mnormbeta       endp

delptr  proc    near
;-----------------------------------------------------------------------
;El mnemotécnico PTR es innecesario en todos los casos. Si DS:SI apunta
;al pseudocódigo de este mnemotécnico, este procedimiento lo salta y deja
;a SI apuntando al siguiente pseudocódigo; en otro caso, no hace nada.
;-----------------------------------------------------------------------
        push    ax
        push    bx
        push    dx
        push    bp
        mov     bp,si
        call    tipo
        jc      noptr
        or      dl,dl           ;mnemo?
        jnz     noptr
        cmp     al,3bh          ;PTR?
        je      siptr
noptr:  mov     si,bp
siptr:  pop     bp
        pop     dx
        pop     bx
        pop     ax
        retn
delptr  endp

japsh?  proc    near
;-----------------------------------------------------------------------
;Este procedimiento determina si la distancia entre AX y DI esta entre
;+127 y -128, en cuyo caso pone esta distancia en AX y CY = 0; de lo
;contrario, CY = 1.
;-----------------------------------------------------------------------
        sub     ax,di
        cmp     ax,127
        jg      notshrt
        cmp     ax,-128
        jl      notshrt
        clc
        retn
notshrt:stc
        retn
japsh?  endp

enspaso2        proc    near
;-----------------------------------------------------------------------
;Este procedimiento ejecuta el segundo paso de compilación, es decir, la
;traducción de los pseudocódigos a un código verdadero.
;
;Entrada:       DS:SI   apunta al pseudocódigo
;               ES:DI   apunta al lugar donde se desea el código
```

```
;
;Salida:        DS:SI   es alterado
;               ES:DI   es ajustado
;               CX      longitud del código
;               CY      set si hay error, en cuyo caso los registros
;                       pueden inalterados
;-------------------------------------------------------------------------
        push    ax
        push    bx
        push    dx
        push    bp
        push    si
        push    di
        push    cx
        xor     cx,cx           ;longitud inicial = 0
start2:call     tipo            ;determinar tipo de pseudocodigo inicial
        jc      er2
        or      dl,dl           ;es una instrucción?
        jnz     prsrg?          ;no, puede ser sreg
        test    ah,1            ;es un loner?
        jz      noeslor         ;no
        cmp     al,0d4h         ;es AAM?
        je      dosb            ;si
        cmp     al,0d5h         ;es AAD?
        jne     unb             ;no
dosb:   mov     ah,0ah          ;es AAM o AAD, instrucciones cuyo opcode es
        stosw                   ;de 2 bytes
        add     cx,2            ;incrementar longitud en 2
        jmp     bienpaso2
unb:    stosb
        inc     cx
        jmp     bienpaso2
prsrg?: cmp     dl,2            ;sreg?
        jne     er2             ;no, error
        shl     al,1
        shl     al,1
        shl     al,1
        or      al,26h          ;hallar el prefijo
        stosb
        inc     cx
        jmp     bienpaso2
er2:    jmp     errp2

noeslon:cmp     al,70h          ;es un salto condicional?
        jb      nojcond
        cmp     al,7fh
        ja      nojcond
saltorel:                       ;si
        stosb
        call    tipo
        jc      er2
        cmp     dl,4            ;lo siguiente es data?
```

```
        jb      er2             ;no. error
        dec     ax
        call    jmpsh?          ;calcular distancia relativa
        jc      er2             ;y determinar si el salto es posible
        stosb
        add     cx,2
        jmp     bienpaso2

nojcond:cmp     al,0e0h         ;LOOP o JCXZ?
        jb      nolpjcx
        cmp     al,0e3h
        jne     saltorel
nolpjcx:cmp     al,8dh          ;LEA?
        jne     diflea
regmemdisp:                     ;si
        stosb
        call    modregrm        ;hallar el código de los argumentos
        jc      er2
        cmp     al,3            ;d y w deben ser 1
        jne     er2
        call    pref?
        mov     al,bl           ;byte de modo de direccionamiento
        stosb
        call    sizedisp
        inc     cx
bp2:    jmp     bienpaso2
diflea: cmp     al,0c4h         ;LDS o LES?
        jb      noldsles
        cmp     al,0c5h
        jbe     regmemdisp
noldsles:
        cmp     al,0f0h         ;prefijo REP o LOCK?
        jb      alterables
        cmp     al,0f3h
        ja      alterables
        stosb                   ;si, incluir el prefijo
        inc     cx
        lodsb
        cmp     al,0ffh
        je      bp2
        dec     si
        jmp     startp2         ;empezar otra vez con lo que sigue al prefijo

alterables:
        test    ah,2            ;oner?
        jnz     sioner
        jmp     nooner
sioner: test    ah,4            ;JMP o CALL?
        jz      nojc
        jmp     jmpocall
nojc:   cmp     al,0c2h         ;es oner.
        jb      nori            ;si no es RET o INT
```

```
        jmp     retoint
norm:   cmp     al,7            ;PUSH o POP?
        je      nopushpop       ;no
        mov     cx,al
        lodsb
        dec     si
        mov     bl,al
        and     bl,0e0h
        cmp     bl,20h          ;reg?
        je      poreg           ;si
        mov     al,dh
        call    segrd           ;no, es?
        jc      ponomas         ;no
        cmp     bh,80h          ;si, w=1?
        je      er2a            ;no, error
        cmp     al,7            ;POP?
        mov     al,8fh
        je      espop1          ;si, dejar su código
        mov     al,0ffh         ;no, código de PUSH
espop1: stosb
        inc     cx
        call    pref?
        cmp     al,8fh          ;POP?
        je      espop2
        or      bl,30h          ;no, compensar byte de direcc.
espop2: mov     al,bl
        stosb
        call    sizedisp
        jmp     bienpaso2
er2a:   jmp     errp2

ponomas:call    tipo
        cmp     dl,0
        je      er2a
        cmp     dl,4
        jae     er2a
        shl     al,1            ;es sreg
        shl     al,1
        shl     al,1
        or      al,7
        cmp     dh,7
        je      espop4
        and     al,0feh
espop4: stosb
        inc     cx
        jmp     bienpaso2

poreg:  test    al,8            ;reg de 16 bits?
        jz      er2a            ;no, error
        and     al,7            ;dejar solo los tres LSBits
        or      al,58h          ;poner header del byte
        cmp     dh,7
```

```
varesta:mov     bl,0e8h
        cmp     ch,9ah          ;call?
        je      escall1
        inc     cl
escall1:rcng    a:,al
        stosb
        mov     cl,3
        mov     al,bl
        stosw
        jmp     short jbien
jsh?:   cmp     ch,0ear         ;jmp?
        jne     errj            ;solo se admite SHORT en JMP
        sub     ax,2
        sub     ax,di
        or      ah,ah
        jz      jshort
        cmp     ah,0ffh
        jne     errj
jshort: mov     ah,al
        mov     al,0ebh
        stosw
        inc     ci
        inc     cl

jbien:  xor     cn,ch
        jmp     bienpaso2
errj:   xor     ch,cn
        jmp     errp2

jnodat: cmp     dl,1            ;reg?
        jne     noreg16
        or      dh,dh           ;no debe haber restricciones de BYTE PTR, etc
        jne     errj
        mov     bl,al
        mov     al,0ffh
        stosb                   ;opcode = FF
        inc     cl
        mov     al,bl
        and     al,7            ;aislar número del registro
        or      al,0c0h         ;mod = 11
        cmp     ch,0eah         ;jmp?
        je      esjmpreg
        or      al,10h
        jmp     short fjd
esjmpreg:
        or      al,20h
fjd:    stosb
        inc     cl
        jmp     short jbien
noreg16:cmp     dh,1            ;es mem, short es inadmisible
        je      errj
        sub     si,4            ;restaurar pseudocódigo
```

```
        stosb
        call    sizedata
        inc     cx
        jrt     bienpaso?
simacc: or      al,30h
        stosb
        call    pref?
        xchg    al,al
        stosb
        call    sizedisp
        mov     al,bl
        shr     al,1
        xor     al,1
        call    sizedata
        inc     cx
        jmp     bienpaso2
mayque40h:
        cmp     al,84h          ;TEST?
        jne     noestest
        call    modregrm
        jc      toat'
        and     al,1
        or      al,84h          ;generar codigo
        stosb
        inc     cx
        call    pref?
        mov     al,bl
        stosb
        call    sizedisp
        jmp     bienpaso2
toat?:  xor     al,al
        call    modrmdata
        jc      er2f
        test    al,4
        jnz     taccd
        or      al,0f6h
        stosb
        inc     cx
        call    pref?
        mov     bh,al
        mov     al,bl
        stosb
        call    sizedisp
        mov     al,bh
        call    sizedata
        jmp     bienpaso2
taccd:  and     al,1
        or      al,0a8h
        stosb
        inc     cx
        call    sizedata
        jmp     bienpaso2
```

```
noestest:
        cmp     al,8eh          ;XCHG?
        jne     noesxchg
        call    tipo
        jc      er2f
        or      dl,dl           ;mreg?
        jz      er2f            ;si, error
        cmp     dl,2            ;sreg?
        je      er2f            ;si, error
        cmp     dl,4            ;datos?
        jae     er2f            ;si, error
        cmp     dl,1            ;reg?
        je      xreg
        sub     si,4            ;no
        jmp     short xmerg
xreg:   test    al,8            ;reg de 16 bits?
        jnz     xrio
        dec     si              ;no
        jmp     short xmerg
xrio:   mov     am,al
        call    tipo
        jc      er2f
        or      dl,dl           ;el segundo operando tampoco
        jz      er2f            ;puede ser mmem, sreg ni datos
        cmp     dl,2
        je      er2f
        cmp     dl,4
        je      er2f
        cmp     dl,1            ;reg?
        je      x2reg
        sub     si,5            ;no
        jmp     short xmerg
er2f:   jmp     errp2
x2reg:  test    al,8            ;segundo reg. de 16 bits?
        jz      er2f            ;no, error por incompatibilidad
        cmp     ah,8            ;primer reg. acumulador?
        je      rgenal
        cmp     al,8            ;no, y el segundo?
        je      rgenah
        sub     si,2            ;no, ninguno es acumulador
        jmp     short xmerg
rgenah: mov     al,ah
rgenal: and     al,7
        or      al,90h
        stosb
        inc     cx
        jmp     bienpaso2
xmerg:  call    modregrm
        jc      er2f
        or      al,86h
        stosb
```

```
        inc     di
        call    oref?
        mov     al,bl
        stosb
        call    sizedisp
        jmp     bienpaso1
noesmov:
        cmp     al,88h          ;MOV?
        je      siesmov
        jmp     noesmov

;-----Lo siguiente es el procesamiento de MOV, que es bastante complejo----
siesmov:push    si
        call    tipo
        jc      er2f
        or      dl,dl           ;mem-mem?
        jnz     mnomme
        pop     si
        jmp     mesmet
mnomme: cmp     al,1            ;reg?
        je      msireg
        jmp     mnoreg
msireg: test    al,1            ;si, acumulador?
        jnz     mnoac
        call    moorm           ;seg. oper. [disp]?
        cmp     bl,6
        jne     mnoac
        test    bh,80h          ;size especificado?
        jz      macany
        shl     bh,1
        shl     bh,1
        shl     bh,1
        mov     dl,al
        and     al,8
        xor     al,bh           ;sizes contradictorios?
        jnz     er2e            ;si, error
        mov     al,dl
macany: shr     al,1
        shr     al,1
        shr     al,1
        or      al,0c0h
        stosb
        inc     di
        call    sizedisp
        pop     si
        jmp     bienpaso2
mnoac:  pop     si
        call    modregrm
        jc      noc1
        jmp     mcaso1
noc1:   lodsb
        and     al,15           ;volver a obtener el reg
```

```
        jmp     bienpaso2
err2:   xor     al,al
        call    moordmdata
        jc      er2d
        test    al,4
        jnz     er2d            ;solo se admite una de las formas
        or      al,0c6h
        stosb
        inc     cx
        mov     bh,al
        call    pref?
        mov     al,bl
        stosb
        call    sizedisp
        mov     al,bh
        call    sizedata
        jmp     bienpaso2
;----------Fin del procesamiento de MOV-------------

noesmov:cmp     al,0d8h         ;ESC?
        jne     noesesc
        call    tipo
        cmp     di,4            ;el primer operador debe ser un dato
        jb      er2a
        cmp     ax,3fh          ;de 6 bits
        ja      er2d
        mov     ah,al
        shr     al,1
        shr     al,1
        shr     al,1
        or      al,0d8h
        stosb
        inc     cx
        mov     al,ah
        call    modrm
        jc      er2c
        call    pref?
        and     al,7
        shl     al,1
        shl     al,1
        shl     al,1
        or      al,bl
        stosb
        call    sizedisp
        jmp     bienpaso2

noesesc:cmp     al,0e4h         ;IN?
        jae     esinout
        call    modrm           ;Rot.o Shft...
        jc      er2c
        test    bh,80h
        jz      er2c            ;size debe especificarse
```

```
        cmp     al,0dch
        jb      rnoaj
        sub     al,8
rnoaj:  and     al,7
        shl     al,1
        shl     al,1
        shl     al,1
        or      bl,al
        mov     bp,dx
        mov     ch,ah
        call    tipo
        jc      errs
        cmp     dl,1
        je      r2reg
        cmp     dl,4
        jb      errs            ;segundo argumento 1 o CL
        cmp     ax,1
        jne     errs
        mov     ah,ch
        xor     ch,ch
        mov     al,bh
        or      al,0d0h
        jmp     short comprot
r2reg:  cmp     al,1            ;CL?
        jne     errs
        mov     ah,ch
        xor     ch,ch
        mov     al,bh
        or      al,0d2h
comprot:stosb
        inc     cx
        call    pref?
        mov     al,bl
        stosb
        mov     dx,bp
        call    sizedisp
        jmp     bienpaso2
errs:   xor     ch,ch
er2c:   jmp     errp2

esinout:cmp     al,0e6h         ;OUT?
        je      esout
        call    tipo            ;no, IN
        cmp     dl,1
        jne     er2c
        test    al,7
        jnz     errp2
        shr     al,1
        shr     al,1
        shr     al,1
        or      al,0e4h
        mov     bl,al
```

```
        stosb
        inc     cx
        jmp     short bienpaso2


error2: pop     cx
        pop     di
        pop     si
        stc
        jmp     short finp2
bienpaso2:
        add     sp,6
        clc
finp2:  pop     bp
        pop     dx
        pop     bx
        pop     ax
        retf
ensPaso2        endp

        public  assem

assem   proc    near
;---------------------------------------------------------------------
;Este es el ensamblador, que llama a los pasos 1 y 2 de compilación.
;
;Entrada:        DS:SI   apunta al string por compilarse
;                ES:DI   apunta al lugar donde se desea el código
;
;Salida:         DS:SI   es alterado
;                ES:DI   es alterado
;                CX      longitud del código
;                CY      set si hay error
;---------------------------------------------------------------------
        push    es
        push    di
        push    ax
        cld
        mov     ax,ds
        mov     es,ax
        mov     di,offset transit
        call    ensPaso1
        jc      error
        mov     si,offset transit
        pop     ax
        pop     di
        pop     es
        call    ensPaso2
        retn
assem   endp
error:  pop     ax
        pop     di
```

```
        .model small

        .code

        public  fill
fill    proc    near
;-----------------------------------------------------------------
;Este procedimiento llena un bloque de memoria con un valor de byte
;determinado.
;
;Entrada:       ES:DI   apunta al inicio del bloque
;               CX      contiene el tamaño del bloque
;               AL      contiene el valor por ponerse en el bloque
;-----------------------------------------------------------------
        cld
        push    cx
        push    di
        rep     stosb
        pop     di
        pop     cx
        retn
fill    endp

        public  search
search  proc    near
;-----------------------------------------------------------------------
;Este procedimiento busca un string en un bloque de memoria.
;
;Entrada:       ES:DI   apunta al inicio del bloque
;               CX      máximo número de bytes que deben ser explorados
;               DS:SI   apunta al string por buscarse
;               BX      longitud del string
;
;Salida:        ES:DI   apunta al string que se encontro, o al final del bloque
;                       si no se lo encontro
;               CX      >0 si se encontro, 0 si no se encontro
;-----------------------------------------------------------------------
        cld
BuscaStr:
        push    cx
        push    si
        push    di
        mov     cx,bx
        repe    cmpsb
        pop     di
        pop     si
        pop     cx
        je      FinSrch
        inc     di
        dec     cx
        jnz     BuscaStr
FinSrch:retn
```

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;EJECUTOR.ASM;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;Ejecutor de porciones de código. Por IVAN ORDOÑEZ.
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

        .model small

        .data

        public  XferAddr,PointerList,Vector2F
        public  RegIp,RegCs,RegSp,RegSs,RegBp,flags
        public  RegSi,RegDs,RegDi,RegEs,RegAx,RegBx,RegCx,RegDx
        public  TablaBrk,FLoadExe,FileSize,FileName
        public  FinNormal,PSPParent

MyLength        equ     1300h

Vector2F        dd      ?
ChildName       db      'Child.exe',0
FileName        db      256 dup (0)

ParBlock        dw      0
PtrCommLine     dw      offset CommLine
CommSeg         dw      ?
                dw      4 dup (?)

ParBlockNew     dd      ?

CommLine        db      0,0,0

FileSize        dw      0
FilePtr         dw      0
ExeFlag         db      0
FinNormal       db      0

RegIp           dw      ?
RegCs           dw      ?
RegSp           dw      ?
RegSs           dw      ?
RegBp           dw      ?
RegDs           dw      ?
RegEs           dw      ?
RegDi           dw      ?
RegSi           dw      ?
RegDx           dw      ?
RegCx           dw      ?
RegBx           dw      ?
RegAx           dw      ?
flags           dw      ?

RegIpInic       dw      ?
RegCsInic       dw      ?
RegSpInic       dw      ?
```

```
RegSsInic       dw      ?
RegEpInic       dw      ?
RegDsInic       dw      ?
RegEsInic       dw      ?
RegDiInic       dw      ?
RegSiInic       dw      ?
RegDxInic       dw      ?
RegCxInic       dw      ?
RegBxInic       dw      ?
RegAxInic       dw      ?
flagsInic       dw      ?

ParSp           dw      ?
ParSs           dw      ?

PSPParent       dw      ?
PSPChild        dw      ?

startName       dw      ?
lenghtName      db      ?
NBytesFaltan    dw      ?

FHandle         dw      ?
rwArea          db      512 dup (?)

storeprv        dw      ?
storeprv2       db      256 dup (?) ;zona de preservacion de inst. anterior
PrsrvPrg        db      32 dup (?)
PrsrvPsp        db      256 dup (?)
vector          dd      ?       ;area de preservación de vectores de int.
vector2         dd      ?
PointerLst      dd      ?       ;apuntador a la última inst. (en EXHASTA)
PointerNxt      dd      ?
XferAddr        dd      ?       ;dirección de transferencia tras la ejec.
XferProv        dd      ?
nada            db      30 dup (?)
PrCodIns        db      16 dup (?)

TipoCompEC      db      ?
RegCompEC       db      ?
ValCompEC       dw      ?

RetardoTF       db      ?       ;bandera indicadora de si se produce retardo en
                                ;la interrupción de trap flag.
FLoadExe        db      ?
TablaBrk        db      4096 dup (?)

        .code
        extrn   desens:proc,assem:proc
loadRegs        proc    near
;---------------------------------------------------------------
;Este procedimiento carga los registros con los valores de memoria.
```

```
;-----------------------------------------------------------------
        mov     ax,@data
        mov     ds ax
        cli     ax              ;extraer la dirección de retorno de esta
        mov     word ptr cs:retloadip,ax        ;rutina
        mov     word ptr cs:retloadcs,cs        ;ponerla en el Jmp final
        mov     bx,flags
        mov     di,RegIp
        mov     es,RegCs
        mov     al,es:[di]      ;opcode de la instrucción por ejecutar
        cmp     al,0fah         ;CLI?
        jne     NoCli
        and     bx,0fdffh       ;si, borrar I
        jmp     short PonFl
NoCli:  cmp     al,0fbh         ;STI?
        jne     NoSti
        or      bx,200h         ;si, set I
        jmp     short PonFl
NoSti:  mov     di,RegSp
        mov     es,RegSs
        cmp     al,9dh          ;POPF?
        jne     NoPopf
        mov     bx,es:[di]      ;si, obtener flags de stack
        and     bx,0feffh       ;borrar T
        jmp     short PonFl
NoPopf: cmp     al,0cfh         ;IRET?
        jne     PonFl
        mov     bx,es:[di+4]    ;si, obtener flags de stack
        and     bx,0feffh       ;borrar T
PonFl:  push    bx
        mov     flags,bx
        popf
        mov     ax,regAx
        mov     bx,regBx
        mov     cx,regCx
        mov     dx,regDx
        mov     bp,regBp
        mov     sp,regSp
        mov     si,regSi
        mov     di,regDi
        mov     es,regEs
        mov     ss,regSs
        mov     ds,regDs
        db      0eah            ;JMP FAR
retloadip:
        dw      ?
retloadcs:
        dw      ?
loadRegs        endp

storeRegs       proc    near
;-----------------------------------------------
```

```
;Este procedimiento pone en memoria los registros.
;-----------------------------------------------
        pushf
        push    cs
        push    ax
        mov     ax,@data
        mov     ds,ax
        pop     ax
        mov     regAx,ax
        mov     regBx,bx
        mov     regCx,cx
        mov     regDx,dx
        mov     regBp,bp
        mov     regSp,sp
        add     regSp,12        ;compensar el CALL y los PUSH
        mov     regSi,si
        mov     regDi,di
        mov     regEs,es
        mov     regSs,ss
        pop     ax
        mov     regDs,ax
        pop     ax              ;AX=flags nuevas
        mov     bx,flags        ;BX=flags anteriores
        and     bx,200h         ;dejar solo la bandera de interrupción
        or      ax,bx           ;dejar el estado anterior de I en flags
        mov     flags,ax
        retn
storeRegs       endp

        public  TestTrap
TestTrap        proc    near
;-----------------------------------------------------------------------------
;Este procedimiento determina como se comporta la computadora en cuanto a si
;hay o no retardo en la interrupción 1, que debería producirse en cuanto sea 1
;la trap flag, pero que suele retardarse en una instrucción en ciertos
;microprocesadores. De ser este el caso, RetardoTF es 1, y de lo contrario 0.
;-----------------------------------------------------------------------------
        push    ax
        push    bx
        push    dx
        push    ds
        push    es
        mov     ax,3501h        ;get vector 1
        int     21h
        mov     ax,@data
        mov     ds,ax
        mov     word ptr vector,bx
        mov     word ptr vector+2,es    ;preservar vector original
        mov     ax,cs
        mov     ds,ax
        mov     dx,offset AfterTest
        mov     ax,2501h        ;poner como vector la dirección de AFTERTEST
```

```
        int     21h

        xor     bx,bx           ;BX = 0
        pushf                   ;flags en el stack
        pop     ax              ;flags en AX
        or      ax,1            ;T = 1
        push    ax              ;flags cambiados de vuelta al stack
        popf
        nop                     ;tras esta instrucción, debe producirse la int
        mov     bx,1            ;si hay retardo, se ejecutará esta instrucción
        nop
        nop
AfterTest:
        add     sp,6            ;restaurar stack, alterado al producirse la int
        mov     ax,@data
        mov     ds,ax
        mov     RetardoTF,0
        cmp     bx,0
        je      YaProbado
        inc     Byte Ptr RetardoTF
YaProbado:
        lds     dx,vector
        mov     ax,2501h        ;restaurar vector 1
        int     21h
        pop     es
        pop     ds
        pop     dx
        pop     bx
        pop     ax
        retn
TestTrap        endp

        public  ExUniq
ExUniq  proc    near
;-------------------------------------------------------------------------
;Este procedimiento coloca el código de SETTRPFLG inmediatamente antes de la
;instrucción que se desea ejecutar (preservando lo que estaba antes), ejecuta
;dicha instrucción, y luego restaura el código. Si la instrucción es una
;interrupción, el procedimiento es diferente: se simula dicha interrupción
;de forma que se recupera el control una vez ejecutada.
;
;Entrada:       RegIp
;               RegCs:          Deben contener la dirección absoluta de la
;                               instrucción que se quiere ejecutar.
;               RetardoTF:      Debe ser 1 si hay que considerar un retardo
;                               en INT 1 tras hacer T=1, 0 en caso contrario.
;               XFERADDR:       Debe contener la dirección absoluta donde debe
;                               transferirse el control una vez finalizado este
;                               procedimiento.
;
;Salida:        RegIp
;               RegCs:          Contienen la dirección de la siguiente
```

```
;                               instruccion, despues de la ejecutada.
;
;IMPORTANTE: Este procedimiento debe invocarse mediante un JMP, y NO mediante
;           un CALL.
;-------------------------------------------------------------------------------
        cli
        mov     ax,@data
        mov     es,ax
        mov     ds,ax
        mov     cl,RetardoTF    ;bandera de retardo en CL
        mov     si,RegIp
        mov     ds,RegCs        ;DS:SI = apuntador a la instruccion
        mov     ax,ds
        cmp     ax,0a000h       ;ROM BIOS?
        jb      noRB
        mov     ax,@data
        mov     ds,ax
        jmp     dword ptr XferAddr      ;si, no hacer nada
noRB:   mov     al,[si]
        cmp     al,0cch         ;es una INT ?
        je      SiInt
        jmp     NoInt
SiInt:  mov     al,[si+1]       ;número de la int.
        cmp     al,21h
        jne     NoInt21
SaltInt:jmp     ExPaso          ;siempre saltar la int 21
NoInt21:mov     ah,35h
        int     21h             ;get vector
        mov     dx,es
        cmp     dx,0a000h       ;ROM BIOS?
        jae     SaltInt         ;si, saltar
        mov     dx,@data
        mov     ds,dx
        mov     ParSp,sp
        mov     ParSs,ss
        mov     sp,RegSp
        mov     ss,RegSs
        push    flags
        popf
        cli
        pushf
        push    RegCs           ;Simular la interrupción
        mov     dx,RegIp
        add     dx,2
        push    dx
        mov     RegSp,sp
        mov     RegSs,ss
        mov     sp,ParSp
        mov     ss,ParSs
        mov     RegIp,bx        ;apuntar a la rutina
        mov     RegCs,es
        jmp     dword ptr xferaddr      ;finalizar
```

```
Main:
        cmp     al,8Ch          ;mov sreg ?
        je      SiMSRg
        cmp     al,8Eh
        jne     NoMSrg
SiMSRg: jmp     ExPaso
NoMSrg: and     al,0E5h
        cmp     al,6            ;push o pop sreg?
        je      SiMSrg
        push    cs
        dec     si
        dec     si              ;apuntar a los dos bytes anteriores
        mov     dx,si           ;preservar este apuntador
        mov     di,offset storeprv ;ES:DI acuna a donde se preserva el codigo
        movsb                   ;preservar código anterior a la instrucción
        movsb
        mov     ax,ds
        mov     es,ax
        mov     di,cs           ;ES:DI apunta a donde se colocara POPF
        mov     ax,909dh        ;código de POPF y NOP
        cmp     cl,6
        jne     AlterCod
        xchg    al,ah           ;si no hay retardo, poner el NOP antes del POPF
AlterCod:
        stosb                   ;colocar el código alterado
        mov     al,cn
        stosb
        mov     ax,3501h        ;get vector 1
        push    dx
        int     21h
        mov     ax,@data
        mov     ds,ax
        mov     word ptr vector,bx
        mov     word ptr vector+2,es    ;preservar vector original
        mov     ax,cs
        mov     ds,ax
        mov     dx,offset afterex
        mov     ax,2501h        ;poner como vector la dirección de AFTEREX
        int     21h
        pop     dx
        pop     ax
        mov     word ptr cs:dirip,dx
        mov     word ptr cs:dircs,ax    ;dirección del salto
        mov     ax,@data
        mov     ds,ax
        mov     ParSp,sp
        mov     ParSs,ss
        call    loadRegs        ;poner los valores de los registros

        pushf                   ;flags en el stack
        push    ax
```

```
        push    bp
        mov     bp,sp           ;apuntar BP al stack
        mov     ax,[bp+4]       ;flags en AX
        or      ah,1            ;T = 1
        mov     [bp+4],ax       ;flags cambiados de vuelta al stack
        pop     bp
        pop     ax
        db      0eah            ;JMP FAR
dirip:  dw      ?
dircs:  dw      ?

alterex:                        ;aqui se pasa el control luego de ejecutar la
        push    ax              ;instruccion única por efecto de la int.
        push    bp
        push    ds
        mov     ax,@data
        mov     ds,ax
        mov     bp,sp
        mov     ax,[bp+6]       ;obtener la dirección de retorno,
        mov     word ptr pointernxt,ax
        mov     ax,[bp+8]
        mov     word ptr pointernxt+2,ax        ;preservar esta direccion
        pop     ds
        pop     bp
        pop     ax
        call    storeRegs       ;salvar registros
        cld
        mov     ax,@data
        mov     ds,ax
        mov     sp,ParSp
        mov     Ss,ParSs
        mov     di,RegIp
        mov     es,RegCs        ;ES:DI = apuntador la instrucción ejecutada
        dec     di              ;apuntar al código alterado
        dec     di
        mov     si,offset storeprv
        movsb                   ;restaurar código
        movsb
        lds     dx,vector
        mov     ax,2501h        ;restaurar vector 1
        int     21h
        mov     ax,@data
        mov     ds,ax
        mov     ax,word ptr pointernxt  ;poner la direccion de la nueva
        mov     RegIp,ax                ;instrucción como resultado
        mov     ax,word ptr pointernxt+2
        mov     RegCs,ax
        jmp     dword ptr xferaddr      ;finalizar

exuniq  endp

        public  exhasta
```

```
ejecuta proc    near
;-------------------------------------------------------------------------
;Este procedimiento ejecuta desde la dirección de inicio indicada (inclusive)
;hasta la de finalización (exclusive) indicada.
;
;Entrada:       RegIp
;               RegCs:          Debe contener la dirección absoluta de la
;                               instrucción desde la que se quiere ejecutar.
;               POINTERLST:     Debe contener la dirección absoluta de la
;                               instrucción hasta la que se debe ejecutar.
;               XFERADDR:       Debe contener la dirección absoluta donde debe
;                               transferirse el control una vez finalizado este
;                               procedimiento.
;
;IMPORTANTE: Este procedimiento debe invocarse mediante un JMP, y NO mediante
;           un CALL.
;-------------------------------------------------------------------------
        cld
        mov     ax,@data
        mov     es,ax
        mov     ah,62h          ;encontrar la dirección de mi PSP
        int     21h
        mov     PSPChild,bx     ;Preservarla, asumiendo que este es un child
        mov     dx,offset aftergo
        call    SetTerm         ;por si hay un EXIT, fijo dirección de term.
        les     si,pointerlst   ;ES:SI = apuntador a la última instrucción
        mov     ax,es
        cmp     ax,0a000h       ;ROM BIOS?
        jb      noRBh
        jmp     dword ptr XferAddr      ;si, no hacer nada
noRBh:  mov     al,es:[si]      ;preservar lo que hay allí
        mov     byte ptr storebrv,al
        mov     byte ptr es:[si],0cch   ;colocar un breakpoint (INT 3)
        mov     ax,3503h        ;get vector 3
        int     21h
        mov     ax,@data
        mov     ds,ax
        mov     word ptr vector,bx
        mov     word ptr vector+2,es    ;preservar vector original
        mov     ax,cs
        mov     ds,ax
        mov     dx,offset aftergo
        mov     ax,2503h        ;poner como vector 3 la dirección de AFTERGO
        int     21h
        mov     ax,@data
        mov     ds,ax
        mov     ax,RegIp        ;preparar ejecución
        mov     word ptr cs:goip,ax
        mov     ax,RegCs
        mov     word ptr cs:gocs,ax
        mov     ParSp,sp
        mov     ParSs,ss
```

```
        call    loadRegs
        dd      0ear            ;JMP FAR
goip:   dw      ?
goCs:   dw      ?

xferaddr:                       ;aqui se pasa el control luego del breakpoint
        call    storeRegs
        cld
        mov     ax,@data
        mov     ds,ax
        mov     sp,ParSp
        mov     ss,ParSs
        mov     ah,62h
        int     21h             ;obtengo direccion de PSP actual
        cmp     bx,PSPChild     ;es este un proceso hijo?
        je      EsHijo
        call    SavePspChild    ;no, preservar parámetros del PSP
        call    ReduceLevel     ;bajar de nivel
        mov     ax,@data
        mov     es,ax
        call    RestPspChild    ;restaurar parametros del PSP
        mov     FinNormal,1     ;mostrar mensaje de finalizacion de proceso
        jmp     short RestDir
EsHijo: mov     ax,word ptr pointerlst  ;poner la direccion de la nueva inst.
        mov     RegIp,ax                ;como resultado
        mov     ax,word ptr pointerlst+2
        mov     RegCs,ax
RestDir:les     si,pointerlst   ;ES:SI = pointer al breakpoint
        mov     al,byte ptr storeprv
        mov     es:[si],al      ;restaurar
        lds     dx,vector
        mov     ax,2503h        ;restaurar vector 3
        int     21h
        mov     ax,@data
        mov     ds,ax
        jmp     dword ptr xferaddr      ;finalizar

exhasta endp

        public  expaso
expaso  proc    near
;-------------------------------------------------------------------------
;Este procedimiento ejecuta desde la direccion de inicio indicada (inclusive)
;hasta la siguiente intrucción (exclusive). Es útil para saltar subrutinas.
;
;Entrada:       RegIp
;               RegCs:          Deben contener la dirección absoluta de la
;                               instrucción desde la que se quiere ejecutar.
;               XFERADDR:       Debe contener la dirección absoluta donde debe
;                               transferirse el control una vez finalizado este
;                               procedimiento.
;
```

```
exCond  proc    near
;-------------------------------------------------------------------------
;Este procedimiento ejecuta desde la dirección de inicio indicada (inclusive)
;en adelante hasta que se cumpla una condicion determinada.
;
;Entrada:       RegIp
;               RegCs:          Deben contener la dirección absoluta de la
;                               instruccion desde la que se quiere ejecutar.
;               AL:             0 si la condición es igualdad de un registro
;                               con un valor
;                               1 si la condicion es desigualdad entre un
;                               registro y un valor, en ambos casos:
;                               AH:     xxx0wReg para registros normales
;                                       xxx1xxSr para registros de segmento
;                               BX:     valor
;                               2 si la condicion es de flags, en este caso:
;                               AH:     xxxxCond
;               XFERADDR:       Dirección de terminación al finalizar
;Salida:        RegIp
;               RegCs:          Apuntan a la proxima intruccion tras cumplirse
;                               la condición
;NOTA: Este procedimiento debe invocarse mediante Jmp, y no Call.
;-------------------------------------------------------------------------
        mov     dx,@data
        mov     ds,dx
        les     dx,XferAddr     ;preservar el XferAddr
        mov     word ptr XferProv,dx
        mov     word ptr XferProv+2,es
        mov     word ptr XferAddr+2,cs
        cmp     al,2                    ;condicion de flag?  -
        jne     RegCond
        mov     dx,offset FlCondRet     ;si, fijar terminacion
        mov     word ptr XferAddr,dx
        and     ah,0fh                  ;generar código de un Jcondition
        or      ah,70h                  ;cambiar el código para que se produzca
        mov     byte ptr cs:condic,ah   ;un salto en la condición dada
        jmp     ExUniq
RegCond:
        mov     dx,offset RegCondRet
        mov     word ptr XferAddr,dx
        mov     TipoCompEC,al
        mov     RegCompEC,ah
        mov     ValCompEC,bx
        jmp     ExUniq
FlCondRet:
        mov     ax,@data
        mov     ds,ax
        cmp     FinNormal,1
        je      GoExitCond
        push    flags
        popf
Condic:
```

```
        mov     dx,RegBp
        jmp     short RdVal
S4EC:   cmp     al,4
        jne     S7EC
        mov     dx,RegSi
        jmp     short RdVal
S7EC:   mov     dx,RegDi
RdVal:  and     al,3
        jnz     wordReg
        and     ValCompEC,0ffh          ;recortar a 8 bits el valor de comp.
        and     dx,0ffh                 ;eliminar MSByte del registro
        cmp     al,4
        jb      WordReg
        jne     S5EC
        mov     dl,byte ptr RegAx+1
        jmp     short WordReg
S5EC:   cmp     al,5
        jne     S6EC
        mov     dl,byte ptr RegCx+1
        jmp     short WordReg
S6EC:   cmp     al,6
        jne     S7EC
        mov     dl,byte ptr RegDx+1
        jmp     short WordReg
S7EC:   mov     dl,byte ptr RegBx+1
WordReg:
        mov     ax,ValCompEC
        cmp     byte ptr TipoCompEC,1
        je      Desig
        cmp     ax,dx                   ;comparar el registro con el valor
        je      ExitCond
        jmp     ExUniq
Desig:  cmp     ax,dx
        jne     ExitCond
        jmp     ExUniq
ExitCond:
        mov     ax,@data
        mov     ds,ax
        les     dx,XferProv             ;restaurar XferAddr
        mov     word ptr XferAddr,dx
        mov     word ptr XferAddr+2,es
        jmp     dword ptr XferAddr
exCond  endp

        public  AddBrkPt
AddBrkPt        proc    near
;-------------------------------------------------------------------------
;Este procedimiento añade un breakpoint a la tabla, de no haber existido
;antes uno en la dirección indicada o con el nombre indicado, caso contrario,
;elimina dicho breakpoint para crear uno nuevo.
;
;Entrada:       DS:SI   apunta al nombre del breakpoint (10 caracteres
```

```
;                       siempre, forzamos final
;               DX:BX   dirección absoluta del breakpoint
;
;Salida:        CF      número del nuevo breakpoint
;               CF      set si hubo error, excediendo el límite de 255
;-------------------------------------------------------------------------
        xor     cl,cl           ;referencia por dirección a DelBrkPt
BorraB: call    DelBrkPt        ;buscar y eliminar de la tabla el breakpoint
        jnc     BorraB
        mov     cl,1
BorraN: call    DelBrkPt        ;eliminar también los de nombre igual
        jnc     BorraN
        push    ax
        push    si
        push    es
        push    di
        cld
        mov     ax,@data
        mov     es,ax
        mov     di,offset TablaBrk
        mov     al,cl                   ;número de breakpoint
SrchFt: cmp     byte ptr es:[di],0      ;fin de tabla?
        je      FinTab
        add     di,16
        inc     al
        jz      ErrAdBr         ;si se llega a 256, error
        jmp     short SrchFt
FinTab: stosb                   ;guardar en la tabla el número del brkpt
        mov     ah,al
        mov     al,1
        stosb                   ;marca activo al nuevo breakpoint
        mov     cx,10
        rep     movsb           ;poner el nombre en la tabla
        mov     cl,ah           ;parámetro de retorno
        xor     ch,ch
        mov     ax,bx           ;guardar dirección en tabla
        stosw
        mov     ax,dx
        stosw
        clc
        jmp     short FinAdBr
ErrAdBr:stc
FinAdBr:pop     di
        pop     es
        pop     si
        pop     ax
        retn
AddBrkPt        endp

        public  DelBrkPt
DelBrkPt        proc    near
;-------------------------------------------------------------------------
```

```
;Este Procedimiento elimina un breakpoint de la tabla.
;
;Entrada:       si CL = 0 entonces
;                       DX:BX   direccion absoluta del breakpoint por eliminar
;               si CL <> 0 entonces
;                       DS:SI apunta al nombre del breakpoint por eliminar
;
;Salida:        CY      si el breakpoint no estaba en la tabla
;-------------------------------------------------------------------------------
        push    ax
        push    cx
        push    si
        push    di
        push    ds
        push    es
        cld
        mov     ax,@data
        mov     es,ax
        mov     di,offset TablaBrk
SrchBrk:cmp     byte ptr es:[di],0
        je      FinTabla
        cmp     cl,0            ;referencia por nombre?
        jne     SrchNam         ;si
        cmp     es:[di+12],bx   ;no
        jne     NoEsEste
        cmp     es:[di+14],dx
        jne     NoEsEste
        jmp     Encontrado
SrchNam:push    si
        push    di
        push    cx
        inc     si
        inc     di              ;apunto al nombre en la tabla
        mov     cx,10
CmpNam: lodsb
        mov     ah,es:[di]
        inc     di
        cmp     ah,al
        jne     FinSrchNam
        loop    CmpNam
FinSrchNam:
        cmp     cx,0
        pop     cx
        pop     di
        pop     si
        je      Encontrado
NoEsEste:
        add     di,16
        jmp     short SrchBrk
FinTabla:
        stc
        jmp     short FinDlBrk
```

```
Encontrado:
        mov     si,di
        add     si,16
        mov     ax,es
        mov     ds,ax
MovTab: cmp     byte ptr [si],0
        je      YaBorrado
        lodsw
        dec     al
        stosw                   ;ajustar el número del breakpoint
        mov     cx,7            ;una vez encontrado el brkpt, eliminarlo
        rep     movsw           ;moviendo hacia atras el resto de la tabla
        jmp     short MovTab
YaBorrado:
        movsb                   ;mover el fin de tabla
        clc
FinDiBrk:
        pop     es
        pop     ds
        pop     di
        pop     si
        pop     cx
        pop     ax
        retn
DelBrkPt        endp

        public  TogBrkPt
TogBrkPt        proc    near
;-------------------------------------------------------------------------
;Este Procedimiento invierte el estado de un breakpoint de la tabla.     -
;
;Entrada:       si CL = 0 entonces
;                       DX:BX   dirección absoluta del breakpoint
;               si CL <> 0 entonces
;                       DS:SI apunta al breakpoint
;
;Salida:        CY      si el breakpoint no estaba en la tabla
;-------------------------------------------------------------------------
        push    ax
        push    cx
        push    si
        push    di
        push    es
        cld
        mov     ax,@data
        mov     es,ax
        mov     di,offset TablaBrk
EncBrk: cmp     byte ptr es:[di],0
        je      FinTablaB
        cmp     cl,0            ;referencia por nombre?
        jne     EncNam          ;si
        cmp     es:[di+12],bx   ;no
```

```
        jne     NoEsEsteB
        cmp     es:[di-14],cx
        jne     NoEsEsteB
        jmp     EncontradoB
EncNam: push    si
        push    di
        push    cx
        inc     di
        inc     di              ;apunto al nombre en la tabla
        mov     cx,10
CmpNom: lodsb
        mov     en,es:[di]
        inc     di
        cmp     al,al
        jne     FinEncNam
        loop    CmpNom
FinEncNam:
        cmp     cx,0
        pop     cx
        pop     di
        pop     si
        je      EncontradoB
NoEsEsteB:
        add     di,16
        jmp     short EncBrk
FinTablaB:
        stc
        jmp     short FinHabBrk
EncontradoB:
        xor     byte ptr es:[di+1],1    ;toggle el breakpoint
FinHabBrk:
        pop     es
        pop     di
        pop     si
        pop     cx
        pop     ax
        retn
TogBrkPt        endp

        public  ExTotal
exTotal proc    near
;-------------------------------------------------------------------------
;Este procedimiento ejecuta desde la dirección de inicio indicada (inclusive)
;hasta encontrar alguno de los breakpoints listado en la tabla. Si no hay
;breakpoints, se ejecuta hasta el fin del programa.
;
;Entrada:       RegIp
;               RegCs:          Deben contener la dirección absoluta de la
;                               instrucción desde la que se quiere ejecutar.
;               XFERADDR:       Debe contener la direccion absoluta donde debe
;                               transferirse el control una vez finalizado este
;                               procedimiento.
```

```
;               TablaBrk        Tabla de breakpoints. Cada uno es un registro
;                               de 16 bytes, y el final de la tabla es un 0.
;                               La estructura de cada registro es:
;
;                       Offset  Longitud        Contenido
;                         0       1         Numero (1 a 255)
;                         1       1         Estado (1 = hab, 0 = desh.)
;                         2      10         Nombre (forzado a 10)
;                        12       4         Dirección absoluta
;
;IMPORTANTE: Este procedimiento debe invocarse mediante un JMP, y NO mediante
;           un CALL.
;-------------------------------------------------------------------------------
        clc
        mov     ax,@data
        mov     de,ax
        mov     ah,62h          ;encontrar la dirección de mi PSP
        int     21h
        mov     PSPChild,bx     ;Preservarla, asumiendo que este es un child

        mov     ax,@data
        mov     de,ax
        les     bx,XferAddr
        mov     word ptr XferProv,bx
        mov     word ptr XferProv+2,es
        mov     word ptr XferAddr,offset AftPrimInst
        mov     word ptr XferAddr+2,cs
        jmp     ExUniq          ;ejecutar la primera instrucción por si habia
                                ;un breakpoint alli.

AftPrimInst:
        les     bx,XferProv
        mov     word ptr XferAddr,bx
        mov     word ptr XferAddr+2,es
        cmp     FinNormal,1
        jne     NoAcabo
        jmp     dword ptr XferAddr
NoAcabo:mov     dx,offset AfterGoTot
        call    SetTera         ;por si hay un EXIT, fijo dirección de tera.
        mov     si,offset TablaBrk
        mov     di,offset StorePrv2
Prsrv:  cmp     byte ptr [si],0
        je      FinPrsrv
        cmp     byte ptr [si+1],0
        js      NxtBrkPt        ;ignorarlo si esta deshabilitado
        mov     bx,[si+12]
        mov     es,[si+14]      ;obtener dirección del BrkPt
        mov     al,es:[bx]      ;preservar lo que hay alli
        mov     [di],al
        mov     byte ptr es:[bx],0cch   ;colocar un breakpoint (INT 3)
        inc     di
NxtBrkPt:
```

```
        sbc     si,ix
        jbe     short Preserv
FinPreserv:
        mov     ax,3503h        ;get vector 3
        int     21h
        mov     ax,@data
        mov     es,ax
        mov     word ptr vector2,bx
        mov     word ptr vector2+2,es   ;preservar vector original
        mov     ax,cs
        mov     ds,ax
        mov     dx,offset AfterGoTot
        mov     ax,2503h        ;poner como vector 3 la direccion de AFTERGOTOT
        int     21h
        mov     ax,@data
        mov     ds,ax
        mov     ax,Regip        ;preparar ejecucion
        mov     word ptr cs:goTotIp,ax
        mov     ax,RegCs
        mov     word ptr cs:goTotCs,ax
        mov     ParSp,sp
        mov     ParSs,ss
        call    loadRegs
        db      0eah            ;JMP FAR
GoTotIp:dw      ?
GoTotCs:dw      ?

AfterGoTot:                     ;aqui se pasa el control luego del breakpoint
        push    ax
        push    bp
        push    ds
        mov     ax,@data
        mov     ds,ax
        mov     bp,sp
        mov     ax,[bp+6]               ;obtener la dirección de retorno
        dec     ax                      ;ignorar el breakpoint
        mov     word ptr pointernxt,ax
        mov     ax,[bp+8]
        mov     word ptr pointernxt+2,ax        ;preservar esta dirección
        pop     ds
        pop     bp
        pop     ax
        call    storeRegs
        cld
        mov     ax,@data
        mov     ds,ax
        mov     sp,ParSp
        mov     ss,ParSs
        mov     ah,62h
        int     21h             ;obtengo dirección de PSP actual
        cmp     bx,PSPChild     ;es este un proceso hijo?
        je      EsHijo2
```

```
        call    SavePspChild    ;mc, preservar parametros del PSP
        call    ReduceLevel     ;bajar de nivel
        mov     ax,@data
        mov     ds,ax
        call    RestPspChild    ;restaurar parámetros del PSP
        mov     FinNormal,1     ;mostrar mensaje de fin de proceso
        jmp     short RestBrk
EsHijo2:
        mov     ax,word ptr pointernxt  ;poner la direccion de la nueva inst.
        mov     Regip,ax                ;como resultado
        mov     ax,word ptr pointernxt+2
        mov     RegCs,ax
RestBrk:mov     di,offset StoreFrv2
        mov     si,offset TablaBrk
Rstr:   cmp     byte ptr [si],0
        je      FinRstr
        cmp     byte ptr [si+13,0
        je      NxtBrk          ;ignorar el brkpt si esta deshabilitado
        mov     bx,[si+12]
        mov     es,[si+14]
        mov     al,[di]         ;restaurar lo alterado por los breakpoints
        mov     es:[bx],al
        inc     di
NxtBrk: add     si,16
        jmp     short Rstr
FinRstr:
        lds     dx,vector2
        mov     ax,2503h        ;restaurar vector 3
        int     21h
        mov     ax,@data
        mov     ds,ax
        jmp     dword ptr xferaddr      ;finalizar

exTotal endp

        public ExIns
ExIns   proc    near
;-------------------------------------------------------------------
;Este procedimiento ejecuta una instrucción no perteneciente al código.
;
;Entrada:       DS:SI   apunta a un asciiz que contiene el mnemotécnico
;                       de la instrucción por ejecutar
;               XFERADDR= punto de transferencia al finalizar.
;Debe ejecutarse mediante JMP.
;-------------------------------------------------------------------
        push    ds
        push    si
        mov     ax,@data
        mov     ds,ax
        mov     es,ax
        mov     si,Regip
        mov     ds,RegCs
```

```
        mov     di,offset PrCodIns
        mov     es:[bx+10],si
        mov     es:[bx+12],cs   ;preservar RegCs,RegIp
        mov     cx,4
        rep     movsw           ;preservar ip actual en RegCs:RegIp
        mov     ax,es
        mov     ds,ax
        mov     di,RegIp
        mov     es,RegCs
        pop     si
        pop     ds
        call    asiem           ;ensamblar, CX tiene la long. del cod. generado
        jc      FinExIns
        mov     ax,@data
        mov     ds,ax
        les     ax,XferAddr
        mov     word ptr XferProv,ax
        mov     word ptr XferProv+2,es
        mov     word ptr XferAddr,offset AftExIns
        mov     word ptr XferAddr+2,cs
        pushf
        jmp     E_ump10
AftExIns:
        pop     cx
        mov     ax,@data
        mov     ds,ax
        les     ax,XferProv
        mov     word ptr XferAddr,ax
        mov     word ptr XferAddr+2,es
        mov     ax,word ptr[PrCodIns+12]
        cmp     ax,RegCs
        jne     RestCod
        mov     ax,word ptr[PrCodIns+10]
        mov     dx,ax
        add     ax,cx
        cmp     ax,RegIp
        jne     RestCod
        mov     RegIp,dx
RestCod:mov     si,offset PrCodIns
        mov     es,word ptr[PrCodIns+12]
        mov     di,word ptr[PrCodIns+10]
        mov     cx,4
        rep     movsw
        clc
FinExIns:
        jmp     dword ptr xferaddr
ExIns   endp

        public AcortaMiMem
AcortaMiMem     proc    near
;-----------------------------------------------------------------------
;Este procedimiento reduce la memoria localizada para el proceso padre a
```

```
;myLenght parrafos, para dejar memoria libre.
;       CY      set si hay error.
;-------------------------------------------------------------------
        mov     ax,@data
        mov     ds,ax
        mov     ah,62h          ;obtener direccion de mi PSP
        int     21h
        mov     PSPparent,bx
        mov     es,bx
        mov     bx,myLength     ;acortar la memoria que utilizo
        mov     ah,4ah
        int     21h
        retn
AbortaMiMem     endp

                public InstHandler2F
InstHandler2F   proc    near
;-------------------------------------------------------------------------
;Este procedimiento instala el Handler de servicio de la interrupción 2F, de
;no estar ya instalado, y además contiene el handler.
;-------------------------------------------------------------------------
        mov     ax,@data
        mov     ds,ax
        mov     ax,8000h        ;preguntar si ya esta instalado el handler
        int     2fh
        or      al,al
        jnz     yaIns
        mov     ax,3524h        ;no, hacer un getVector para preservar el
        int     21h             ;vector original
        mov     word ptr Vector2F+2,es
        mov     word ptr Vector2F,bx
        mov     dx,offset Handler2F
        mov     ax,cs
        mov     ds,ax
        mov     ax,2524h        ;SetVector de mi rutina
        int     21h
Yains:  retn

Handler2F:
        cmp     ah,80h          ;este es el servicio a la interrupción
        je      mio
        push    ax
        push    es
        mov     ax,@data
        mov     ds,ax
        mov     ax,word ptr Vector2f
        mov     word ptr cs:[Salto+1],ax
        mov     ax,word ptr Vector2f+2
        mov     word ptr cs:[salto+3],ax
        pop     ds
        pop     ax
Salto:  db      0eah
```

```
        cd      ?

dio:    cmp     al,0f8h         ;funcion reservada?
        jc      noReserv
        iret                    ;si, fin
noReserv:
        or      al,al           ;es un request de existencia?
        jnz     noRequest
        mov     al,0ffh         ;si, decir que ya esta instalado
        iret
noRequest:
        jmp     Int2fProc       ;pasa..er control
Int2fHandler endp

        public  SavePspChild
SavePspChild    proc    near
;-------------------------------------------------------------------
;Este procedimiento preserva la segunda mitad del PSP del proceso Child.
;-------------------------------------------------------------------
        push    ax
        push    cx
        push    si
        push    di
        push    ds
        push    es
        mov     si,80h
        mov     di,offset PrsrvPsp
        mov     ax,ds
        mov     es,ax
        mov     ds,PSPChild
        mov     cx,40h
        rep     movsw
        pop     es
        pop     ds
        pop     di
        pop     si
        pop     cx
        pop     ax
        retn
SavePspChild    endp

        public  RestPspChild
RestPspChild    proc    near
;-------------------------------------------------------------
;Este procedimiento restaura la segunda mitad del PSP del Child.
;-------------------------------------------------------------
        mov     si,offset PrsrvPsp
        mov     es,PSPChild
        mov     di,80h
        mov     cx,40h
        rep     movsw
        retn
```

```
RestPspChild    endp

        public ReduceLevel
ReduceLevel     proc    near
;-------------------------------------------------------------------------
;Este procedimiento ejecuta Child.exe, que simplemente devuelve el control a
;ChildProc, de forma que se baja el nivel a procedimiento hijo. Restaura los
;registros a sus valores iniciales. Retorna CY=1 si por algun razon no es
;posible correr Child.exe.
;-------------------------------------------------------------------------
        mov     ax,@data
        mov     ds,ax
        mov     es,ax
        mov     ax,PSPChild
        add     ax,10h
        mov     ds,ax
        xor     si,si
        mov     di,offset PrsrvPrg
        mov     cx,10h                  ;preservar lo que hacia antes desde
        rep     movsw                   ;PSPChild+10:0, 32 bytes
        mov     ax,es
        mov     ds,ax
        mov     CodeSeg,ax
        mov     dx,offset ChildName
        mov     bx,offset ParBlock
        mov     ax,4b00h
        mov     ParSs,ss
        mov     ParSp,sp        ;preserva Stack
        int     21h             ;EXEC
        mov     ax,@data        ;El control no se transfiere si hay error, y
        mov     ds,ax           ;viene aqui, asi que recupero Stack y retorno.
        mov     ss,ParSs        ;Notese que CY es 1 en este punto, por el fallo
        mov     sp,ParSp        ;de la funcion del DOS.
        retn
ChildProc:
        mov     ax,@data
        mov     ds,ax
        mov     ss,ParSs
        mov     sp,ParSp        ;Recuperar Stack
        mov     ax,PSPChild
        add     ax,10h
        mov     es,ax
        mov     si,offset PrsrvPrg
        xor     di,di
        mov     cx,10h
        rep     movsw           ;Restaurar lo alterado debido al EXEC
        mov     ah,62h          ;encontrar la dirección del PSP del child
        int     21h             ;process
        mov     PSPChild,bx     ;Preservarla
FinRedLev:
        mov     ax,RegIpInic
        mov     RegIp,ax
```

```
        mov     ax,RegCsInic
        mov     RegCs,ax
        mov     ax,RegSpInic
        mov     RegSp,ax
        mov     ax,RegSsInic
        mov     RegSs,ax
        mov     ax,RegBpInic
        mov     RegBp,ax
        mov     ax,RegDsInic
        mov     RegDs,ax
        mov     ax,RegEsInic
        mov     RegEs,ax
        mov     ax,RegDiInic
        mov     RegDi,ax
        mov     ax,RegSiInic
        mov     RegSi,ax
        mov     ax,RegDxInic
        mov     RegDx,ax
        mov     ax,RegCxInic
        mov     RegCx,ax
        mov     ax,RegBxInic
        mov     RegBx,ax
        mov     ax,RegAxInic
        mov     RegAx,ax
        mov     ax,flagsInic
        mov     flags,ax
        retn
ReduceLevel     endp

        public  SetTerm
SetTerm         proc    near
;-------------------------------------------------------------------
;Este procedimiento fija la direccion de terminación del proceso hijo.
;
;Entrada:       DX = dirección de terminacion
;-------------------------------------------------------------------
        mov     ax,PSPChild
        mov     es,ax
        mov     es:[10],dx
        mov     es:[12],cs
        retn
SetTerm         endp

        public SaveFile
SaveFile        proc    near
;-------------------------------------------------------------------------
;Este procedimiento graba en el archivo de nombre representado por un ASCIIZ
;en FileName desde la posición indicada por FilePtr, el número de bytes
;resultante de (FileSize)-(FilePtr). De tratarse de un modulo .exe (ExeFlag=1),
;se graba desde ParBlockNew+2:0, de lo contrario, desde RegCsInic:100h o 0h
;segun sea o no un archivo .com. No se puede generar modulos .exe nuevos.
;-------------------------------------------------------------------------
```

```
        lodsb
        and     al,0dfh
        cmp     al, 'Z'
        jne     Datos
        mov     cx,1300h
Datos:  mov     ds,ArgCsInic
        jcxz    short WriteFile
SiEsExe:mov     ds,word ptr ParBlockNew-2
WriteFile:
        mov     ah,40h
        int     21h
        jc      ErrSaveFile
        cmp     ax,cx           ;grabado todo exitosamente?
        je      FinSave
ErrSaveFile:
        mov     ax,@data
        mov     ds,ax
        mov     bx,FHandle
        mov     ah,3eh
        int     21h
        stc
        jmp     short FSav
FinSave:
        mov     ah,3eh          ;cerrar archivo, BX ya tenía el handle
        int     21h
FSav:   pop     ds
        pop     di
        pop     dx
        pop     cx
        pop     bx
        pop     ax
        retn
SaveFile        endp

        public LoadExe
LoadExe         proc    near
;---------------------------------------------------------------------
;Este procedimiento carga el archivo cuyo nombre se pasó como parámetro
;a este programa en el PSP del proceso padre, al final del PSP del hijo.
;Luego calcula los valores iniciales de los segmentoregistros, y pasa los
;siguientes parámetros al PSP del hijo. CY es 1 si se pasó un nombre de
;archivo incorrecto o si hubo problemas al cargar el archivo.
;Obtiene la posición del Load Module dentro del archivo que se carga.
;
;Además se retorna la siguiente información en FLoadExe:
;       0       si es un archivo .exe
;       1       si es un archivo .com
;       2       si es un archivo de datos
;       3       si no se especificó un nombre de archivo
;       4       si el archivo especificado no existe
;       5       si no se pudo cargar el archivo .exe
;---------------------------------------------------------------------
```

```
        mov     ax,@data
        mov     ds,ax
        mov     ParSp,sp
        mov     ParSs,ss
        mov     ax,PSPParent
        mov     es,ax           ;ES apunta al PSP del padre.
        mov     di,81h          ;Voy a buscar el nombre del archivo.
        mov     cx,7fh
        cmp     byte ptr es:[80h],0
        je      noParam
        mov     al,' '
        cld
        repe    scasb           ;encontrar el primer elemento diferente de '
        cmp     cx,0
        je      NoParam
        dec     di
        inc     cx
        mov     si,di
        mov     ax,es
        mov     ds,ax           ;DS apunta al PSP del padre
        mov     ax,@data
        mov     es,ax
        mov     di,offset FileName
XferName:
        lodsb
        cmp     al,' '
        je      noEndN
        jmp     endNombreArch
noEndN: stosb
        dec     cx
        jz      NotFound
        jmp     short xferName
NotFound:
        mov     byte ptr FLoadExe,4
        jmp     SinProgramaExe
NoParam:
        mov     byte ptr FLoadExe,3
        jmp     SinProgramaExe

LoadNoExe:
        mov     dx,@data
        mov     ds,dx
        mov     ds,PSPParent
        mov     si,cx
        lodsb
        cmp     al,':'
        jne     NoCom
        lodsb
        and     al,0dfh
        cmp     al,'C'
        jne     NoCom
        lodsb
```

```
        and     al,0dfh
        cmp     al,'C'
        jne     NoCom
        lodsb
        and     al,0dfh
        cmp     al,'M'
        je      SiCom
NoCom:  mov     es,dx
        mov     byte ptr FLoadExe,2
        jmp     short LoadFile
SiCom:  mov     ds,dx
        mov     byte ptr FLoadExe,1
LoadFile:
        mov     dx,offset FileName
        mov     ax,3d40h
        int     21h             ;abrir archivo
        jc      NotFound
        mov     FHandle,ax
        mov     bx,ax
        mov     dx,100h
        mov     cx,0ffffh
        mov     ds,PSPChild
        mov     ah,3fh
        int     21h             ;leer archivo
        jc      NotFound
        mov     bx,@data
        mov     ds,bx
        mov     TablaBrk,0      ;eliminar breakpoints
        mov     FileSize,ax
        mov     bx,FHandle
        mov     ah,3eh
        int     21h             ;cerrar archivo

SinProgramaExe:
        mov     ax,@data
        mov     ds,ax
        mov     ss,ParSs
        mov     sp,ParSp
        mov     ExeFlag,0
        mov     FilePtr,0       ;posicionar pointer al inicio de archivo
        mov     ax,PSPChild
        mov     RegIp,100h
        mov     RegIpInic,100h
        cmp     FLoadExe,2
        jne     Ip100
        mov     RegIp,0
        mov     RegIpInic,0
        add     ax,10h
Ip100:  mov     RegCs,ax
        mov     RegCsInic,ax
        mov     RegDs,ax
        mov     RegEs,ax
```

```
        jae     ListoPSPChild
        stosb
        jmp     short Comclet
ListoPSPChild:
        mov     ds,0
        mov     si,StartAddr
        sub     si,4            ;apuntar al la extension del archivo
BuscClase:
        mov     ds,PSPParent
        mov     cx,si
        lodsb
        cmp     al,'.'
        je      BienP
        jmp     LoadNoExe
BienP:  lodsb
        and     al,0dfh
        cmp     al,'E'
        je      BienE1
        jmp     LoadNoExe
BienE1: lodsb
        and     al,0dfh
        cmp     al,'X'
        je      BienX
        jmp     LoadNoExe
BienX:  lodsb
        and     al,0dfh
        cmp     al,'E'
        je      BienE2
        jmp     LoadNoExe
BienE2: mov     ds,dx
        mov     dx,offset FileName
        mov     ax,3d40r        ;Abrir archivo
        int     21h
        jnc     bien2
        jmp     NotFound
bien3:  mov     FHandle,ax
        mov     bx,ax
        mov     dx,offset rwArea
        mov     cx,512
        mov     ah,3fh
        int     21h             ;Leer los primeros 512 bytes del archivo
        jnc     bien4
        jmp     NotFound
bien4:  mov     ah,3eh
        mov     bx,FHandle      ;Cerrar el archivo
        int     21h
        jnc     bien5
        jmp     NotFound
bien5:  mov     si,offset rwArea
        cmp     word ptr [si],5a4dh
        je      BienL           ;Verificar la marca del Linker
        mov     cx,@data
```

```
        mov     cs,cx
        mov     cx,StartName
        sub     cx,4
        jmp     LoadNoExe
Bien1:  mov     ax,[si+10h]     ;preservar en memoria los valores de Stack y
        mov     RegSp,ax        ;de transfer address del programa que se
        mov     ax,[si+14h]     ;va a controlar
        mov     Reglp,ax
        mov     RegIpInic,ax
        mov     ax,[si+4]
        mov     cl,8
        shl     ax,cl
        add     ax,[si+2]       ;obtener la longitud del archivo
        mov     FileSize,ax
        mov     ax,[si-2]
        mov     FilePtr,ax      ;posición del Load Module en el archivo
        mov     bx,PSPChild
        add     bx,10h          ;poner al archivo al final del PSP hijo
        mov     word ptr ParBlockNew,bx
        mov     word ptr ParBlockNew+2,ax
        mov     ax,cs
        mov     es,ax
        mov     dx,offset FileName
        mov     bx,offset ParBlockNew
        mov     ax,4b03h
        int     21h             ;EXEC, cargar el archivo sin ejecutarlo
        jnc     bien6
        jmp     NotFound
bien6:
        mov     ax,@data
        mov     ds,ax
        mov     ExeFlag,1
        mov     TablaBrk,0      ;borrar breakpoints
        mov     si,offset rwArea
        mov     ax,word ptr ParBlockNew+2       ;Relocation factor
        mov     bx,[si+16h]
        add     bx,ax
        mov     RegCs,ax        ;Fijar nuevo valor de CS
        mov     RegCsInic,bx
        mov     bx,[si+0eh]
        add     bx,ax
        mov     RegSs,bx        ;Fijar Nuevo valor de SS
        mov     ax,RegSp
        mov     RegBp,ax
        mov     ax,PSPChild
        mov     RegEs,ax        ;Crear los seudoregistros
        mov     RegDs,ax
        xor     ax,ax
        mov     RegAx,ax
        mov     RegBx,ax
        mov     RegCx,ax
        mov     RegDx,ax
```

```
        .model  small

        .data
        public  tabbuf,tabinf,nmaxbuf
        extrn   dirret:near,graux1:word,graux2:word,otroaux:byte
        extrn   videoseg:word,valotr:word,colven:near
        extrn   soaux:byte,moaux1:byte,tieprt:byte
        extrn   areamen:near,bpsac:byte,bmac:byte,bver:byte
        extrn   RegAx:word,RegBx:word,RegCx:word,RegDx:word
        extrn   RegDs:word,RegEs:word,RegCs:word,RegSS:word
        extrn   RegSi:word,RegDi:word,RegBp:word,RegSp:word
        extrn   Flags:word,RegIp:word
        extrn   PointerList:dword,XferAddr:dword
        extrn   vector2f:dword,FLoadExe:byte
        extrn   PSPparent:byte,FileSize:word,FileName:byte
        extrn   FinNormal:byte,TableBrk:byte

nmaxbuf equ     31                                      ;# de buffers

tabbuf  db      01h,00h,00h,00h,00h,00h,00h             ;principal
        db      11h,00h,00h,00h,00h,00h,00h             ;archivos.e.
        db      11h,01h,00h,00h,00h,00h,00h             ;er
        db      11h,02h,00h,00h,00h,00h,00h             ;er
        db      11h,03h,00h,00h,00h,00h,00h             ;al
        db      11h,04h,00h,00h,00h,00h,00h             ;ag
        db      11h,05h,00h,00h,00h,00h,00h             ;ar
        db      21h,00h,00h,00h,00h,00h,00h             ;informacion
        db      31h,09h,00h,00h,00h,00h,00h             ;depurador
        db      31h,01h,00h,00h,00h,00h,00h             ;oa
        db      31h,02h,00h,00h,00h,00h,00h             ;oc
        db      31h,03h,00h,00h,00h,00h,00h             ;od
        db      31h,04h,00h,00h,00h,00h,00h             ;or
        db      31h,14h,00h,00h,00h,00h,00h             ;orc
        db      31h,24h,00h,00h,00h,00h,00h             ;ora
        db      31h,34h,00h,00h,00h,00h,00h             ;orm
        db      31h,44h,00h,00h,00h,00h,00h             ;drm
        db      31h,54h,00h,00h,00h,00h,00h             ;drf
        db      31h,05h,00h,00h,00h,00h,00h             ;di
        db      31h,06h,00h,00h,00h,00h,00h             ;de
        db      31h,16h,00h,00h,00h,00h,00h             ;deg
        db      31h,26h,00h,00h,00h,00h,00h             ;deb
        db      31h,36h,00h,00h,00h,00h,00h             ;def
        db      31h,07h,00h,00h,00h,00h,00h             ;dm
        db      31h,17h,00h,00h,00h,00h,00h             ;dml
        db      31h,27h,00h,00h,00h,00h,00h             ;dmo
        db      31h,37h,00h,00h,00h,00h,00h             ;dmc
        db      31h,47h,00h,00h,00h,00h,00h             ;dmf
        db      31h,08h,00h,00h,00h,00h,00h             ;df
        db      41h,00h,00h,00h,00h,00h,00h             ;termina
        db      51h,00h,00h,00h,00h,00h,00h             ;funciones

tabinf  dw      offset exp0,offset menu0,offset rut0    ;principal
```

```
        db      1
        dw      offset edi,offset menu0,offset regs     ;ori
        db      1
        dw      offset edi,offset menu0,offset reg      ;de
        db      1
        dw      offset ef,offset menu0,offset run       ;termina t
        db      0
        dw      offset edi,offset menu0,offset fi       ;funciones(f)
        db      1

exc     db      ' ',0
ea      db      'Funciones relativas a archivos',0
ean     db      'Fija el nombre del (Objetivo)',0
eat     db      'Fija el tamaño del archivo a grabar',0
eal     db      'Lee el archivo nombrado',0
eag     db      'Graba archivo nombrado en su estado actual .(
ai      db      'Informacion sobre PRO.EXE .)
ec      db      'Depurador interactivo',0
ecp     db      'Ejecuta la proxima instruccion .(
ecc     db      'Corre hasta cumplir un evento: direccion o condicion .'
ecj     db      'Ejecuta hasta terminar o encontrar un punto de ruptura .
ecr     db      'Puntos de ruptura .(
ecrc    db      'Marca un punto de ruptura .
ecra    db      'Activa o desactiva un punto de ruptura',0
ecrb    db      'Elimina un punto de ruptura',0
ecrm    db      'Muestra la tabla de puntos de ruptura',0
edi     db      'Ejecuta una instrucción',0
ede     db      'Ensambla instrucciones',0
edee    db      'Ensambla y coloca en memoria',0
edeb    db      'Ensambla y busca en el segmento actual',0
edm     db      'Operaciones en la ventana principal de memoria',0
edml    db      'Llena desde la posición actual en memoria con un valor .(
edmb    db      'Busca un grupo de datos desde posicion actual en memoria  .
edmc    db      'Copia desde posicion actual en memoria a otra area .(
ed*     db      'Funciones especiales ,0
ef      db      'Fin del programa ,0


menu0   db      'Archivos   Información   Depurador
        db      '   Termina   Funciones',0,5,'AIDTF',0ffh
ma      db      'Nombre  Tamaño  Lee  Graba  Funciones',0,5,'NTLBF',0ffh
md      db      'Paso  Corre  eJecuta  Rupturas  Ins  Ens  Memoria'
        db      '  Funciones',0,8,'PCJRIEMF',0ffh
mde     db      'Ensamble  Busca  Funciones',0,3,'EBF',0ffh
mdm     db      'Llena  Busca  Copia  Funciones',0,4,'LBCF',0ffh
mcr     db      'Cree  Active/desactive  Borra  Muestra'
        db      '  Funciones',0,5,'CABMF',0ffh


;!!!!!!!!!!! datos del programa !!!!!!!!!!!!
revers  db      70h             ;video reverso
normal  db      7
```

```
evtcolo db      9
colo    db      04h

colour  db      70h,4,7,7,70h,70h ;atributos para menues en color
bnflujo db      70h,7,7,7,70h     ;y monocromaticos
vmtmono db       probm.vmt',0     ;archivos de ventanas:monocromaticos
vmtcolor db      procolor.vmt',0  ;a color
arcmacr db       pro.mcr ,0       ;archivo de macros
vlin0   db       LINEA0',0        ;nombres de las ventanas
vlin1   db       LINEA1 ,0
vcod    db      'CODIGO',0
vpro    db      'PRO',0
vpro1   db      'PRO1',0
vreg    db      'REGISTRO',0
vpsw    db      'PSW',0
vexoc   db      'EXPCOD',0
vstk    db      'STACK',0
vocm1   db      'OPCMEM1',0
vocm2   db      'OPCMEM2',0
vmc     db      'MEMEM ',0
vac     db      'ASCMEM',0
vcc     db      'CCC ',0
vc1     db       COP1 ,0
vc2     db       COP2 ,0
vc2s    db      'COP2SEG',0
vc2o    db      'COP2OFF',0
vc2h    db      'COP2HEX ,0
vc2d    db      'COP2DESP',0
vdh     db      'DHEX',0
vda     db      'DASC',0
vr      db      'REG',0
vb      db      'DBIN',0
vhas    db      'HASTA',0
vens    db      'ENS',0
vcond   db      'COND ',0
vcban   db      'CBAND',0
vrnom   db      'RUPNOR',0
vrdir   db      'RUPDIR',0
vfnorm  db      'FINNORM',0
vrv     db      'REGVAL',0
vf9     db      'DECHEX',0
vasc1   db      'ASCII1',0
vasc2   db      'ASCII2',0
vrup    db      'RUPTURAS',0
vll     db      'LLENAR',0
vcopia  db      'COPIAR ',0
vfunc   db      'FUNCION',0
vayp    db      'AYP',0
vaya    db      'AYA',0
vayd    db      'AYD',0
vaye    db      'AYE',0
vaym    db      'AYM',0
```

```
InsEnt  dt      20 dup(0)
atrore, dt      0
atrorev dw      0
atrore, dt      0
atrorev dw      0
rorreg  dt      2,0,0,0,0
rupact  dt      0               ;rupotura actual - 1
brib    dt      0               ;bandera para rut. de rupturas
StackLow dw     0
StackHi dw      0
bufai   dw      7 dup(0)
chk1    db      0ch
chk21   db      13
chk22   db      33

regs    dt      'DSESSSCSAXBXCXDXSIDIBP'
bcrx    db      '[BX]',0        ;opciones para el offset de
cbbxt   dt      '[BX     ]',0   ;la ventana de memoria.
bbbr    dt      '[BP]',0
bbbbt   dt      '[BP     ]',0
bbb.    dt      '[BI] ',0
bbsib   dt      '[SI     ]',0
bbi1    db      '[DI]',0
bbdib   db      '[DI     ]',0
bbbxsi  db      '[BX+SI]',0
bbbxsib db      '[BX+SI     ]',0
bbbxdi  db      '[BX+DI]',0
bbbxdib db      '[BX+DI     ]',0
bbbbsi  db      '[BP+SI]',0
bbbbsib db      '[BP+SI     ]',0
bbbbdi  dt      '[BP+DI] ',0
bbbbdib db      '[BP+DI     ]',0

linea0  dt      44 dup(' '),20 dup(0)
linea1  dt      80 dup(' ')
l0ini   db      'Ayuda con F1 o ^P ',0
l1ini   db      '1Editor 2Ayuda  3DirRel 4PntUsr 5
        dt      '6       7Direcc 8SiSubr 9Dec/Hex10ASCII '
mono?   db      'ENTER: Monitor monocromático.    ESC: A color. ',0
123d_   db      'TAB: proxima ventana  SHIFT TAB: ventana
        db      ' anterior  ESC: menú principal',0
124d_   db      'Acceso a las ventanas del depurador',0
l0reg   db      'Cambio de valores de registros',0
l1reg   db      'Usa las flechas para pasar de uno a otro registro',0
l0ban   db      'Cambios en banderas del 8086',0
l1ban   dt      'Usa las flechas ',26,' y  ,27,' para pasar a otra bandera',0
l0coo   db      'Desensamblador: Página ',0
l1cod   db      26,' y ',27,' : pasar al otro campo.
        db      'PGUP y PGDN : cambiar de página',0
l1cod1  dt      '!! ATENCION: PGDN: próxima página    PGUP: página ? !!',0
l0bb1   dt      'Cambio de posición de ventana de memoria',0
l0bb2   db      'Segmento para ventana de memoria relativa',0
```

```
lscr1   db      26, ' y ',27, ' : Otra opcion.  ',24, ' y ',25, ' : Otra linea,
        db          PGDN y PGUP : Segmento ',29,' Offset .$
lscr22  db      Offset para ventana de memoria relativa .$
lscr2s  db      Escribe el desplazamiento con el signo .'
lscr1c  db      Escribe el offset .$
lscr2s  db      'Escribe el segmento .$
lvhex   db      Acceso hexadecimal a ventanas de memoria .$
lines   db      Flechas: movimiento en las ventanas.
        db        PGUP y PGDN: cambio de paginas',$
lvasc   db      'CR=ENTER,  TAB=[F10]TAB,  ESC=[F10]ESC',0
lscrs   db      'Direccion=     :     .  Escribe'
        db       nemotecnico o ESC para salir',0
errnem  db      'Este nemotécnico no es correcto',$
EnsBusc db      'Escribe el nemotécnico de la instruccion buscada',0
NoEncE  db      'No encuentro la instruccion en el segmento de código',0
EnsWi   db      'Puedo encontrar el código de la  instruccion, pero el flujo',0
EnsW2   db      del programa no  sera necesariamente  de la forma mostrada. ,0
instruc db      'Escribe el nemotécnico de la instruccion a ejecutarse .$
lscbkc  db      PGUP : PGDN: Pasar de pagina.   ESC: Salir .'
lscc    db      'TAB: Pasar al otro campo.  ESC: Sa...'
lscc    db      24, ' y ',25, ' : Pasar a otra opcion.
        db        ESC: Salir.  ENTER: Opcion .'
lscrmo  db      Escribe el registro , o ESC para salir .$
lscmem  db      ...y ahora el valor, o ESC para terminar .$
igual   db      '__  ____',0
noesreg db      Este no es un registro. Presiona una tecla
        db       o ESC para salir',0
nombre? db      'Escribe el nombre del <Objetivo> o ESC para salir',0
leo     db      'Leyendo...',0
grabo   db      'Grabando...',0                 -
nonomo  db      'Aun no tengo el nombre del archivo .$
noney   db      'Este archivo no existe .$
tamano? db      'Escribe el tamaño del archivo (hex) .$
long(   db      No puedo aceptar tamaño 0',$
errgrab db      'Error al grabar el archivo',0
valor?  db      'Escribe el valor o ESC para salir',0
bytes?  db      'Escribe el numero de bytes o ESC para salir .$
buscasc? db     'Secuencia en ASCII o hexadecimal (A/H)? ,0
busca?  db      'Escribe la secuencia o ESC para salir',0
NoHayStr db     'No encuentro la secuencia en todo el segmento',0
sigo?   db      'ENTER: Busca próxima.    ESC: Termina.',0
noesbrk db      'No tengo ese punto en mi lista',0

presen1 db      27 dup(0cdh),0                  ;col x
presen2 db      'PRO.EXE',0                     ;col x+2
presen3 db      'Depurador  para el 8086',0     ;col x+2
presen4 db      'I. Ordóñez    F. Ortega',0     ;col x+2
presen5 db      'E.P.N.   1989',0               ;col x+11
dinfo   db      'Iv',160,'nOrd',162,164,'ez'
        db      'FernandoOrtegaEPNESPONA'
dlin0.  db      'FOrtegaIOrd',162,164, 'ezEPN ,5 dup '0'
```

```
rfinal: mov     di,0            ;limpio pantalla al salir
        mov     cx,2000
        mov     ax,0720h
        rep     stosw
        grabmcr                 ;grabo macros
        cerrmcr
        les     dx,Vector2F     ;restauro int 2f
        mov     ax,252fh
        int     21h
        mov     ax,@data
        mov     ds,ax
        mov     Stacklow,sp     ;salvo stack y modo de nivel
        mov     Stackhi,ss
        mov     dx,offset rfin1
        call    SetTerm
        mov     ah,4ch
        int     21h
rfin1:  mov     ax,@data        ;recupero stack y regreso a terminar
        mov     ds,ax
        mov     es,videoseg
        mov     ss,stackhi
        mov     sp,stacklow
        retn

rut0:
        jmp     salida

rf:                             ;ayuda menu principal
        cmp     otroaux,0       ;ENTER, pasar
        je      rff
        cmp     mopaux,0
        jne     rff
        cmp     mopaux1,60      ;F2?
        jne     rff
        mov     bx,offset vayp  ;muestro ayuda
        mnt     'a'
        limpln  23,revers
        limpln  24,normal
        print   1,23,TecCont,revers
        getchar
        call    pdebug
        jmp     salida
rff:    jmp     rdfun           ;otras funciones

raf:                            ;ayuda menu archivos
        cmp     otroaux,0       ;ENTER, pasar
        je      raff
        cmp     mopaux,0
        jne     raff
        cmp     mopaux1,60      ;F2?
        jne     raff
```

```
        mov     bx,offset vava
        vnt     'a'
        limpln  23,revers
        limpln  24,normal
        print   1,23,TecCont,revers
        getchar
        call    pdebug
        jmp     salida
raff:   jmp     rdfun           ;otras funciones

rdf:                            ;ayuda menu depurador
        cmp     otroaux,0       ;ENTER, pasar
        je      rdxf
        cmp     mopaux,0
        jne     rdxf
        cmp     mopaux1,60      ;F2?
        jne     rdxf
        mov     bx,offset vavd
        vnt     'a'
        limpln  23,revers
        limpln  24,normal
        print   1,23,TecCont,revers
        getchar
        call    pdebug
        jmp     salida
rdxf:   jmp     rdfun           ;otras funciones

rdef:                           ;ayuda menu ensamblador
        cmp     otroaux,0       ;ENTER, pasar
        je      rdeff
        cmp     mopaux,0
        jne     rdeff
        cmp     mopaux1,60      ;F2?
        jne     rdeff
        mov     bx,offset vaye
        vnt     'a'
        limpln  23,revers
        limpln  24,normal
        print   1,23,TecCont,revers
        getchar
        call    pdebug
        jmp     salida
rdeff:  jmp     rdfun           ;otras funciones

rdmf:                           ;ayuda menu memoria
        cmp     otroaux,0       ;ENTER, pasar
        je      rdmff
        cmp     mopaux,0
        jne     rdmff
        cmp     mopaux1,60      ;F2?
        jne     rdmff
        mov     bx,offset vaym
```

```
        vnt     a
        limpln  23,revers
        limpln  24,normal
        print   1,23,TecCont,revers
        getchar
        call    poebug
        jmp     salida
roeff:  jmp     rdfun           ;otras funciones

rdrf:                           ;ayuda menu rupturas
        cmp     otroaux,0       ;ENTER, pasar
        je      rdrff
        cmp     mopaux,0
        jne     rdrff
        cmp     mopaux1,60      ;F2?
        jne     rdrff
        mov     bx,offset vayr
        vnt     'a'
        limpln  23,revers
        limpln  24,normal
        print   1,23,TecCont,revers
        getchar
        call    poecuc
        jmp     salida
rdrff:  jmp     rdfun           ;otras funciones

rinfo:                          ;informacion
        lmpnt                   ;limpio pantalla
        mov     dinfo+2,0a0h
        mov     dinfo+7,0a2h
        mov     dinfo+8,0a4h
        mov     bx,offset vinfo ;muestro ayuda
        mov     ax,ds
        mov     si,offset dinfo
        vnt     'a'
        limoln  23,revers
        limpln  24,normal
        print   1,23,TecCont,revers
        getchar
        call    poebug
        mov     si,offset l0ini ;preparo y muestro linea 0
        call    mlin0
        mov     bx,offset vlini
        mov     ax,ds
        mov     si,offset llini
        vnt     'a'
        jmp     salida

checkl: push    ax
        push    di
        mov     di,offset presen4
        mov     cx,92
```

```
        mov     al,0
chk1:   add     al,[di]
        inc     di
        loop    chk1
        cmp     al,chk1
        je      ck1f
        push    es
        push    ds
        pop     es
        mov     di,offset tabbuf
        mov     ax,5
        mov     cx,2000
        rep     stosb
        pop     es
ck1f:   pop     di
        pop     ax
        retn
ran:                            ;recibe el nombre del Objetivo
        call    check1
        limpln  23,revers
        limpln  24,normal
        print   1,23,nombre?,revers
        mov     gdreset,1
rrn1:   getline 1,24,archivo
        mov     gdreset,0
        cmp     gccod,4         ;ESC?
        je      rrnf
        cmp     gccod,1         ;valido?
        jne     rrn1
        cmp     archivo+1,0     ;vacio?
        je      rrnf
        mov     si,offset archivo+1     ;vale
        push    es              ;paso a PSP del padre y a FilsName
        mov     es,PSPparent
        mov     di,80h
        mov     cx,80
        rep     movsb
        mov     ax,@data
        mov     es,ax
        mov     di,offset FileName
        mov     si,offset archivo+2
        mov     cx,80
        rep     movsb
        pop     es
rrnf:   jmp     salida

rat:                            ;Recibo tamano del archivo
        call    rrt0
        jmp     salida

rrt0:   limpln  23,revers
        limpln  24,normal
```

```
        print   1,23,tamano?,revers
        mov     goreset,1
rrt1:   getdata 1,24,dhex1,'h'
        mov     goreset,0
        cmp     gocod,7
        jne     rrt2
        jmp     rrtf            ;ESC
rrt2:   cmp     gocod,0
        jne     rrt1            ;no valida,regreso
        mov     ax,word ptr dhex1+6
        cmp     ax,0            ;longitud cero no sera aceptada
        jne     rrt3
        limpln  23,revers
        print   1,23,long0,revers
        jmp     rrtf
rrt3:   mov     FileSize,ax
rrtf:   retn

ral:                            ;lee el Objetivo desde disco
        mov     StackLow,sp     ;salvo SS:SP
        mov     StackHi,ss
        limpln  23,revers
        limpln  24,normal
        push    es
        mov     es,PSPparent
        cmp     byte ptr es:[80h],1
        pop     es
        jae     rrla
        print   1,23,nonomb,revers      ;aun no tengo nombre
        jmp     rrlf
rrla:   print   1,23,leo,revers ;leyendo...
        call    LoadExe
        mov     ax,@data
        mov     ds,ax
        mov     es,videoseg
        jnc     rrlb
rrlm:   cmp     FLoadExe,4      ;errores en LoadExe
        jne     rrlm1
        print   1,23,MalParam,revers
        jmp     rrlbl
rrlm1:  print   1,23,MalExec,revers
        jmp     rrlf
rrlb:   mov     sp,StackLow     ;lectura OK
        mov     ss,StackHi
        print   1,23,listo,revers
rrlbl:  call    pdebug
rrlf:   mov     si,offset l0ini ;preparo y muestro linea 0
        call    mlin0
        mov     bx,offset vlini
        mov     ax,ds
        mov     si,offset llini
        vnt     'e'
```

```
        getchar
        jmp     salida

rag:                            ;grabo un archivo
        limpin  23,revers
        limpln  24,normal
        push    es
        mov     es,PSPparent
        cmp     byte ptr es:[80h],1
        pop     es
        jae     rrga
        print   1,23,noncab,revers      ;aun no tengo nombre
        jmp     rrgf
rrga:   cmp     FileSize,0      ;Longitud 0 no aceptada
        jne     rrg2
rrg1:   call    rri0            ;recibo longitud
        cmp     gccod,7         ;se presiono ESC?
        je      rrgf
        cmp     FileSize,0      ;sigue siendo cero?
        je      rrg1
rrg2:   limpln  23,revers       ;aqui ya tengo FileSize
        limpin  24,normal
        print   1,23,grabo,revers
        call    SaveFile
        jc      rrg3
        mov     ax,@data
        mov     ds,ax
        mov     es,videoseg
        print   1,23,listo,revers
        jmp     rrgf
rrg3:   print   1,23,errgrab,revers ;error en escritura
rrgf:   mov     si,offset l0ini ;preparo y muestro linea 0
        call    mlin0
        mov     bx,offset vlin1
        mov     ax,ds
        mov     si,offset l1ini
        int     'a'
        getchar
        jmp     salida

mlin0:  call    prelin          ;preparo linea 0
        mov     dlin0,'F'
        mov     dlin0+1,'0'
        mov     dlin0+7,'1'
        mov     dlin0+8,'0'
        mov     ax,ds
        mov     bx,offset vlin0
        mov     si,offset linea0
        int     'a'
        retn

pdebug: call    mreg            ;prepara y muestra: REGIST
```

```
        cmp     banens,1        ;Si BANENS=1,no mostrar codigo
        je      mdeb21
        mov     ax,regcs
        mov     segcod,ax
        mov     ax,regip        ;si REGIP esta en pagina actual,
        cmp     ax,offcod       ;no cambiar de inicio de pagina
        jb      mdeb1
        cmp     ax,proxins
        jb      mdeb2
mdeb1:  mov     offcod,ax
mdeb2:  call    mcod            ;CODIGO
mdeb21: mov     bx,offset vpro  ;ventana Pro
        mvt     'a'
        call    mpsw            ;FSW!
        call    check1
        call    mmem            ;MEM
        call    mstk            ;STCK
        call    mexpc           ;EXPCOD
        cmp     FinNormal,1     ;acabo?
        jne     mdeb3
        mov     bx,offset vfnorm ;ventana de terminacion
        mvt     'a'
        getchar
        xor     FinNormal,1
        jmp     mdebug
mdeb3:  retn

mcod:   push    ds              ;Preparo el buffer CODIGO con 13
        push    es              ;instrucciones desensambladas
        mov     si,offcod       ;y lo muestro en pantalla
        mov     di,offset codigo ;En PROXINS sale offset de prox.
        mov     ax,@data        ;instruccion
        mov     es,ax
        push    di
        mov     cx,13*33        ;limpio buffer
        mov     al,20h
        rep     stosb
        pop     di
        push    di              ;limpio TABRUP
        mov     di,offset tabrup
        mov     cx,13
        mov     al,0
        rep     stosb
        pop     di
        push    di
        mov     cx,segcod       ;guardo el valor del segmento que
        mov     al,ch           ;ira al inicio de todas las lineas
        hexasc
        mov     al,cl
        mov     cx,bx
        hexasc
        mov     ax,cx
```

```
        xchg    al,ah
        xchg    bl,bh
        mov     aux,ax
        mov     aux+2,bx
        pop     di
        mov     di,13           ;13-DL=# de linea (0-12)
        mov     dh,0ffh         ;# linea a resaltarse
pcod0:  call    check2
        mov     ax,segcod
        call    ptr0
        cmp     ax,regcs        ;coincide el CS?
        jne     pcod01
        cmp     si,regip        ;coincide el IP?
        jne     pcod01
        mov     dh,13           ;si ambos coinciden, linea debe resaltarse
        sub     dh,dl
pcod01: mov     ax,aux          ;escribo segmento
        stosw
        mov     ax,aux+2
        stosw
        mov     ds,segcod
        call    desensobi       ;desensamblo linea
        mov     bx,@data
        mov     ds,bx
        add     si,cx
        add     di,29
        cmp     byte ptr [di-1],' '
        jne     pcod3           ;Si la linea no es muy larga
        cmp     al,0            ;pongo el tipo de instruccion.
        jne     pcod1           ;AL = 0 --> byte ptr
        mov     byte ptr [di-1],'B'
        jmp     pcod3
pcod1:  cmp     al,1            ;AL = 1 --> word ptr
        jne     pcod2
        mov     byte ptr [di-1],'W'
        jmp     pcod3
pcod2:  cmp     al,2            ;AL = 2 --> dword ptr
        jne     pcod3           ;AL = 255 --> nada
        mov     byte ptr [di-1],'D'
pcod3:  dec     dl
        jnz     pcod0
        pop     es
        pop     ds
        mov     proxins,si      ;guardo dir. de prox. pagina
        mov     ax,ds           ;ventana Codigo.
        mov     bx,offset vcod
        mov     si,offset codigo
        mov     'a'
        cmp     dh,0ffh
        je      pcod5           ;no hay linea que resaltar
        add     dh,3
        mov     dl,10
```

```
        posdir  dl,dh
        mov     di,posir
        inc     di
        mov     cx,26           ;resalto linea CS:IP
        mov     al,70h
pcod4:  stosb
        inc     di
        loop    pcod4
pcod5:  call    pcr00           ;resalto bkpts
        retn

check2: push    ax
        push    di
        mov     al,dinfo
        xor     al,44h
        cmp     al,chk21
        jne     ck2e
        mov     al,dinfo+1
        xor     al,67h
        cmp     al,chk22
        je      ck2f
ck2e:   push    es
        push    ds
        pop     es
        mov     di,offset tabbuf
        mov     ax,5
        mov     cx,2000
        rep     stosb
        pop     es
ck2f:   pop     di
        pop     ax
        retn

pcr0:   ;veo si AX:SI pertenecen a TABLABRK y si es asi,
        ;pongo un 1 en la entrada respectiva de TABBRUP si
        ;es punto activo o 2 si es inactivo
        stodo
        mov     di,offset TablaBrk
        mov     ch,0            ;CH=# del bkpt buscado
pcr01:  mov     bl,ch
        mov     bh,0
        mov     cl,4
        shl     bx,cl           ;multiplico por 16
        cmp     byte ptr[di+bx],0 ;veo si termine la tabla
        je      pcr03
        cmp     [di+bx+12],si   ;offset igual?
        jne     pcr02           ;no
        cmp     [di+bx+14],ax   ;si, segmento igual?
        jne     pcr02           ;no
        mov     al,0
        cmp     [di+bx+1],al    ;si, hay un bkpt. Desactivado?
        jne     pcr011
```

```
        mov     al,1            ;si
pcr011: inc     al              ;AL=1 activos, 2 inactivos
        mov     di,13
        sub     di,cl
        mov     di,offset TabRup
        mov     bh,0            ;BX+DI acuerta a entraca en TABRUP
        mov     byte ptr[di+bx],al
        jmp     pcr03           ;salgo
pcr02:  inc     ch              ;preparo siguiente busqueda
        jnz     pcr01
pcr03:  rtodo
        retn
pcr00:  ;busco en TABRUP las lineas que tienen bkpts y las resalto.
        mov     nlinea,0
        mov     si,offset TabRup
pcr001: mov     bl,nlinea       ;para cada linea veo si debo resaltar
        mov     bh,0
        cmp     byte ptr[bx+si],0
        je      pcr002          ;no resalto
        mov     al,1            ;resalto
        cmp     byte ptr[bx+si],1 ;es activo?
        jne     pcr003
        mov     di,70h          ;si, video reverso; no, subrayado.
pcr003: mov     ch,bl
        add     ch,3
        posdir  0,ch
        mov     di,pddir
        inc     di
        mov     al,dl
        call    rdcd01          ;resalto 4 casillas
pcr002: inc     nlinea
        cmp     nlinea,13
        jne     pcr001
        retn

prelin: mov     bx,di           ;preparo LINEA0
        mov     di,offset linea0
        push    es
        push    ds
        pop     es
        mov     al,' '          ;primero borro
        mov     cx,44
        rep     stosb
prl1:   mov     cx,44           ;escribo LINEA0
        mov     di,offset linea0
prl11:  lodsb
        cmp     al,0
        je      prl2            ;salgo al encontrar un 0
        stosb
        loop    prl11           ;o escribir 44 bytes
prl2:   mov     cx,18           ;escribo nombres
        mov     si,offset dlin0
```

```
        mov     di,offset lines0+44
        rep     movsb
        pop     es
        ret

mreg:   mov     di,offset regist
        push    es              ;preparo buffer REGIST
        push    ds
        pop     es
        mov     ax,RegBp        ;Leo un registro del usuario
        xchg    al,ah           ;pongo MSB antes de LSB
        stosw                   ;y guardo en el buffer.
        mov     ax,RegDs        ;El orden de los registros es el
        xchg    al,ah           ;adecuado para la ventana REGISTRO.
        stosw
        mov     ax,RegSi
        xchg    al,ah
        stosw
        mov     ax,RegAx
        xchg    al,ah
        stosw
        mov     ax,RegEs
        xchg    al,ah
        stosw
        mov     ax,RegDi
        xchg    al,ah
        stosw
        mov     ax,RegBx
        xchg    al,ah
        stosw
        mov     ax,RegCs
        xchg    al,ah
        stosw
        mov     ax,RegIp
        xchg    al,ah
        stosw
        mov     ax,RegCx
        xchg    al,ah
        stosw
        mov     ax,RegSs
        xchg    al,ah
        stosw
        mov     ax,RegSp
        xchg    al,ah
        stosw
        mov     ax,RegDx
        xchg    al,ah
        stosw
        pop     es
        mov     ax,ds           ;ventana Registro
        mov     bx,offset vreg
        mov     si,offset regist
```

```
        mov     'h'
        retn

mpsw:   push    es              ;prepare buffer PSW1
        push    ds
        pop     es
        mov     ax,'0'          ;limpio el buffer
        mov     cx,8
        mov     di,offset psw1
        rep     stosb
        mov     di,offset psw1
        mov     bx,flags
        mov     cl,5
        rcl     bx,cl
        jnc     ppswo
        mov     ax,'1'
ppswo:  stosb                   ;O
        mov     ax,'0'
        rcl     bx,1
        jnc     ppswd
        mov     ax,'1'
ppswd:  stosb                   ;D
        mov     ax,'0'
        rcl     bx,1
        jnc     ppswi
        mov     ax,'1'
ppswi:  stosb                   ;I
        mov     ax,'0'
        rcl     bx,1
        rcl     bx,1
        jnc     ppsws
        mov     ax,'1'
ppsws:  stosb                   ;S
        mov     ax,'0'
        rcl     bx,1
        jnc     ppswz
        mov     ax,'1'
ppswz:  stosb                   ;Z
        mov     ax,'0'
        rcl     bx,1
        rcl     bx,1
        jnc     ppswa
        mov     ax,'1'
ppswa:  stosb                   ;A
        mov     ax,'0'
        rcl     bx,1
        rcl     bx,1
        jnc     ppswp
        mov     ax,'1'
ppswp:  stosb                   ;P
        mov     ax,'0'
        rcl     bx,1
```

```
        rcl     bx,1
        jnc     ppswc
        mov     ax,'1'
ppswc:  stosb                   ;0
        pop     es
        mov     ax,ds           ;Ventana Psw
        mov     bx,offset vpsw
        mov     si,offset psw1
        int     'a'
        retn

mmem:   call    predir          ;prepara direcciones.
mmemx:  mov     dx,ds
        mov     cx,32           ;preparo buffers MEM1 y MEM2
        mov     si,dir1         ;en base a DIR1 y DIR21 a DIR25
        mov     di,offset mem1
        push    es
        push    ds
        pop     es              ;ES=DS
        mov     ax,dir1+2
        xchg    ax,dx           ;DS apunta a seg. de datos de usuario
        mov     ds,dx
        rep     movsw
        xchg    ax,dx           ;listo MEM1. ES:DI apunta a MEM2.
        mov     ds,dx
        mov     nlinea,0
        mov     bx,offset dir21
mmem1:  mov     cx,4            ;preparo MEM2
        mov     si,[bx]         ;leo offset
        mov     ax,[bx+2]       ;y segmento
        xchg    ax,cx
        mov     ds,dx
        rep     movsw
        xchg    ax,dx
        mov     ds,dx
        add     bx,4
        inc     nlinea
        cmp     nlinea,5
        jne     mmem1
        pop     es              ;MEM2 listo
        mov     ax,ds           ;ventana Opcmem1
        mov     bx,offset vopm1
        mov     si,offset memo1
        int     'a'
        mov     ax,ds           ;ventana Opcmem2
        mov     bx,offset vopm2
        mov     si,offset memo2
        sub     si,8
        int     'a'
        mov     ax,ds           ;ventana Hexmem
        mov     bx,offset vhm
        mov     si,offset mem1
```

```
        vnt     'h'
        mov     ax,ds           ;ventana Ascmem
        mov     ox,offset vam
        mov     si,offset mem!
        vnt     'a
        retn

mstk:   pusn    es
        mov     cx,4            ;preparo buffer STCK
        mov     si,14           ;segunda columna.
        mov     bx,offset stck
        mov     di,0
        mov     bp,regsp
        mov     es,regss
pstk1:  mov     ax,es:[bp+di]
        xchg    al,ah
        mov     [bx+si],ax
        sub     si,4
        add     di,2
        loop    pstk1
        mov     cx,5            ;primera columna
        mov     si,16
pstk2:  mov     ax,es:[bp+di]
        xchg    al,ah
        mov     [bx+si],ax
        sub     si,4
        add     di,2
        loop    pstk2
        pop     es
        mov     ax,ds           ;Ventana Stack
        mov     bx,offset vstk
        mov     si,offset stck
        vnt     'h'
        retn

mexpc:  mov     ax,regcs        ;pongo en EXPCOD los valores de
        push    ax              ;CS e IP del usuario.
        xchg    al,ah
        mov     word ptr expcod,ax
        mov     ax,regip
        push    ax
        xchg    al,ah
        mov     word ptr expcod+2,ax
        mov     ax,@data        ;paso 10 bytes de codigo a EXPCOD+4
        mov     es,ax
        mov     di,offset expcod+4
        pop     si
        pop     ds
        mov     cx,10
        rep     movsb
        mov     ax,@data
        mov     ds,ax
```

```
        mov     es,videoseg
        mov     dx,offset vexpc ;presento ventana
        mov     ax,ds
        mov     si,offset expcad
        int     21h
        retn

predir: mov     nlinea,0        ;para cada linea...
prd1:   mov     bl,nlinea       ;calculo el segmento:
        mov     bh,0
        mov     al,opcseg[bx]   ;leo opcion de segmento
        add     bx,bx
        dec     al              ;la. opcion es la cero
        actuar                  ;en base a AL salto a una rutina
        dw      offset prd11,offset prd12,offset prd13,offset prd14
        dw      offset prd15,offset prd16,offset prd17,offset prd18
        dw      offset prd19,offset prd1a,offset prd1b,offset prd1c
prd11:  mov     ax,RegDs        ;DS
        jmp     prd1f
prd12:  mov     ax,RegEs        ;ES
        jmp     prd1f
prd13:  mov     ax,RegSs        ;SS
        jmp     prd1f
prd14:  mov     ax,RegCs        ;CS
        jmp     prd1f
prd15:  mov     ax,RegAx        ;AX
        jmp     prd1f
prd16:  mov     ax,RegBx        ;BX
        jmp     prd1f
prd17:  mov     ax,RegCx        ;CX
        jmp     prd1f
prd18:  mov     ax,RegDx        ;DX
        jmp     prd1f
prd19:  mov     ax,RegSi        ;SI
        jmp     prd1f
prd1a:  mov     ax,RegDi        ;DI
        jmp     prd1f
prd1b:  mov     ax,RegBp        ;BP
        jmp     prd1f
prd1c:  mov     ax,xxxxseg[bx]  ;XXXX
prd1f:  add     bx,bx
        add     bx,2            ;guardo el valor del segmento
        mov     dir21[bx],ax
prd2:   mov     bl,nlinea       ;calculo el Offset:
        mov     al,opcoff[bx]   ;leo opcion de offset.
        dec     al              ;el despl. sera sumado luego por
        add     bx,bx           ;lo que calculo sin desplazamiento.
        mov     dx,0
        clc
        shr     al,1
        actuar                  ;en base a AL salto a subrutina
        dw      offset prd21,offset prd22,offset prd23
```

```
        dw      offset prd24,offset prd25,offset prd26
        dw      offset prd27,offset prd28,offset prd29
prd25:  mov     dx,Reg5i        ;BX+SI
prd21:  add     dx,Reg8x        ;BX
        jmp     prd2a
prd27:  mov     dx,Reg5i        ;BP+SI
prd22:  add     dx,Reg8p        ;BP
        jmp     prd2a
prd23:  mov     dx,Reg8i        ;SI
        jmp     prd2a
prd24:  mov     dx,Reg8i        ;DI
        jmp     prd2a
prd26:  mov     dx,Reg8i        ;BX+DI
        add     dx,Reg8x
        jmp     prd2a
prd28:  mov     dx,Reg8i        ;BP+DI
        add     dx,Reg8p
        jmp     prd2a
prd29:  mov     dx,xxxxoff[bx]  ;XXXX
prd2a:  mov     bl,nlinea       ;veo si debo sumar despl.
        mov     al,opcoff[bx]
        mov     ah,signoff[bx]  ;leo el signo (+ o -)
        add     bx,bx
        dec     al
        test    al,1            ;si es par no necesita despl.
        jz      prd2f
        cmp     ah,'+'
        jne     prd2c
prd2b:  add     dx,xxxxoff[bx]  ;signo +, sumo despl.
        jmp     prd2f
prd2c:  sub     dx,xxxxoff[bx]  ;signo -, resto despl.
prd2f:  add     bx,bx
        mov     dir2[bx],dx     ;guardo el offset.
prd3:   inc     nlinea          ;fin del lazo
        cmp     nlinea,5        ;fin?
        je      prd4
        jmp     prd1            ;no.
prd4:   retn

preopc: mov     cx,75           ;preparo buffer MEMO1 en base a DIR1
        mov     di,offset memo2 ;y MEMO2 en base a OPCSEG, OPCOFF,
        mov     al,' '          ;XXXXSEG y XXXXOFF.
        push    es
        push    ds
        pop     es
        rep     stosb           ;primero borro MEMO2
        mov     di,offset memo1 ;escribo MEMO1
        mov     bx,dir1+2       ;segmento
        mov     dx,dir1         ;offset
        mov     cx,8
        mov     bp,bx           ;guardo BX.
popc1:  mov     bx,bp           ;escribo direcciones
```

```
        mov     al,dh           ;de la primera ventana.
        hexasc
        mov     ax,bx
        xchg    al,ah
        stosw
        mov     bx,bp
        mov     al,bl
        hexasc
        mov     ax,bx
        xchg    al,ah
        stosw
        mov     al,dh
        hexasc
        mov     ax,bx
        xchg    al,ah
        stosw
        mov     al,dl
        hexasc
        mov     ax,bx
        xchg    al,ah
        stosw
        add     bx,2
        loop    popc1
popc2:  mov     niinea,0        ;para cada linea de la ventana 2...
popc20: mov     bl,niinea       ;calculo la longitud.
        mov     bh,0
        mov     al,15
        mul     bl
        mov     di,offset memo2
        add     di,ax
        mov     al,3            ;long. minima del segmento y ":"
        mov     dl,opcseg[bx]
        cmp     dl,12
        jne     popc21
        add     al,2            ;en caso de XXXX
popc21: mov     ah,4            ;long. minima del offset y "[]"
        mov     dl,opcoff[bx]
        cmp     dl,17
        jne     popc22
        add     ah,2            ;direccionamiento directo
        jmp     popc3
popc22: mov     dh,dl
        dec     dl
        clc
        shr     dl,1
        inc     dl
        cmp     dl,4
        jbe     popc23          ;indexado simple
        add     ah,3            ;indexado doble
popc23: dec     dh
        test    dh,1            ;necesita desplazamiento?
        jz      popc3           ;no.
```

```
        add     ah,5            ;si.
popc3:  add     ah,al           ;longitud de la linea
        mov     al,15
        cwt     al,ah
        mov     ah,0
        add     di,ax           ;DI apunta a la posicion inicial
        mov     di,opcseg[bx]   ;escribo el segmento
        cmp     dl,12           ;es XXXX?
        jne     popc31
        add     bx,bx
        mov     cx,xxxxseg[bx]  ;escribo segmento
        mov     al,cn
        hexasc
        mov     ax,bx
        xchg    al,ah
        stosw
        mov     al,cl
        hexasc
        mov     ax,bx
        xchg    al,ah
        stosw
        mov     bl,nlinea       ;dejo listo BX para offset
        mov     bh,0
        jmp     popc32
popc31: dec     dl
        mov     dh,0            ;escribo registro que da el segmento
        mov     cx,dx
        mov     si,offset regs
        add     si,dx
        movsw
popc32: mov     al,':'          ;escribo ':'
        stosb
        mov     dl,opcoff[bx]   ;escribo offset
        cmp     dl,17           ;es XXXX?
        jne     popc4
        mov     al,'['          ;si, directo
        stosb
        add     bx,bx
        mov     cx,xxxxoff[bx]
        mov     al,ch
        hexasc
        mov     ax,bx
        xchg    al,ah
        stosw
        mov     al,cl
        hexasc
        mov     ax,bx
        xchg    al,ah
        stosw
        mov     al,']'
        stosb
        jmp     popc5
```

```
popct   dw      offset opax,offset opbx,offset opcx,offset opdx
        dw      offset opsi,offset opsix,offset opdi,offset opdix
        dw      offset opbxsi,offset opbxsix,offset opbxdi
        dw      offset opbxdix,offset opbpsi,offset opbpsix
        dw      offset opbpdi,offset opbpdix
popc4:  dec     cl              ;voy a salir a POPC44 con SI apuntando
        mov     dh,0            ;al inicio del string a mostrarse.
        add     ax,dx           ;indexado.
        mov     si,offset popct ;Tabla de direcciones de las opciones
        add     si,dx
        mov     si,cs:[si]
popc44: lodsb                   ;SI apunta al inicio del string.
        cmp     al,0            ;acaso linea?
        je      popc5
        cmp     al,' '          ;espera el despl.?
        je      popc45
        stosb
        jmp     popc44
popc45: mov     al,signoff[bx]  ;pongo el signo
        stosb
        add     bx,bx           ;pongo el desplazamiento
        mov     cx,xxxxoff[bx]
        mov     al,ch
        hexasc
        mov     ax,cx
        xchg    al,ah
        stosw
        mov     al,cl
        hexasc
        mov     ax,bx
        xchg    al,ah
        stosw
        mov     al,']'          ;pongo el corchete final
        stosb
popc5:  inc     nlinea
        cmp     nlinea,5
        je      popc6
        jmp     popc20
popc6:  pop     es
        retn

rdp:                            ;Corre un paso del programa
        usrpro
        mov     word ptr xferaddr,offset YaEj
        mov     word ptr xferaddr+2,cs
        cmp     toggsub,0       ;ingreso a subrutinas,reps,loops?
        je      rdp2
        jmp     expaso          ;no
rdp2:   jmp     exuniq          ;si
YaEj:   mov     ax,@data
        mov     ds,ax
        mov     es,videoseg
```

```
        usrproc
        call    pdebug
        jmp     salida

roc:                            ;Corre hasta la ocurrencia de un evento
        linpin  23,revers
        linpln  24,normal
        mov     word ptr xferaccr,offset \aEjHas
        mov     word ptr xferaddr+2,cs
        cmp     toggcorre,0     ;Hasta direccion?
        je      rdcd
        jmp     rdcc
rdco:   mov     ax,RegCs        ;Corrida hasta cierto punto
        mov     word ptr dhex1+6,ax
        xchg    al,ah           ;Hago que CS usuario sea seg. predef.
        push    ax              ;para lo que simulo condiciones
        hexasc
        pop     ax
        xchg    si,bp
        mov     word ptr dhex1+2,bx
        mov     bufcorr,bx
        mov     al,ah
        hexasc
        xchg    bl,bh
        mov     word ptr dhex1+4,bx
        mov     bufcorr+2,bx
        mov     byte ptr dhex1+1,4
rdcd1:  mov     bx,offset vhas  ;ventana para recibir datos
        mov     ax,ds
        mov     si,offset bufcorr
        vnt     'a'
        print   1,23,123cd,revers
        mov     gdreset,1
        cmp     byte ptr bufcorr,'_'
        je      rdcd11
        mov     ax,'__'
        mov     bufcorr,ax
        mov     bufcorr+2,ax
        jmp     rdcd2
rdcd11: call    rdcd0           ;resalto campo de segmento
        getdata 10,20,dhex1,'h' ;recibo segmento
        mov     al,atrprev      ;quito resaltado
        mov     di,dirprev
        call    rdcd01
        mov     gdreset,0
        cmp     gccod,0         ;dato valido?
        je      rdcd2
        cmp     gccod,7         ;ESC para abortar
        jne     rdcd12
        jmp     rdcf
rdcd12: cmp     gccod,5         ;TAB para pasar
        jne     rdcd11
```

```
rdcd2:  mov     gdreset,1
rdcd21: call    rdcd09          ;resalto campo de offset
        getdata 17,20,dhex2,'h' ;recibo offset
        mov     al,atrprev      ;quito resaltado
        mov     di,dirprev
        call    rdcd01
        mov     gdreset,0
        cmp     gccod,0         ;dato valido?
        je      rdcd3
        cmp     gccod,7         ;ESC para abortar
        jne     rdcd22
        jmp     rdcf
rdcd22: cmp     gccod,5         ;TAB para pasar
        jne     rdcd21
        jmp     rdcd1
rdcd3:  usrpro                  ;Pantalla de usuario<-->PRC
        mov     bx,word ptr dhex2+6
        mov     word ptr pointerlst,bx
        mov     ax,word ptr dhex1+6
        mov     word ptr pointerlst+2,ax
        jmp     exhasta
rdcc:   mov     bx,offset vcond ;ventana con las condiciones
        vnt     'a'
        print   1,23,123cc,revers
        mov     nlinea,0        ;para cada linea...
rdcc1:  call    rdcc0           ;resalto linea actual
        getchar
        mov     al,atrprev      ;quito el resaltado
        mov     di,dirprev
        call    rdcc01
        cmp     gccod,1         ;ENTER?
        je      rdcc2
        cmp     gccod,4         ;ESC?
        jne     rdcc12
        jmp     rdcf            ;abortar
rdcc12: cmp     gccod,5         ;especial?
        jne     rdcc1
        cmp     gcascii,72      ;si, arriba?
        jne     rdcc13
        dec     nlinea          ;si
        cmp     nlinea,0ffh
        jne     rdcc1
        mov     nlinea,2
        jmp     rdcc1
rdcc13: cmp     gcascii,80      ;abajo?
        jne     rdcc1
        inc     nlinea          ;si
        cmp     nlinea,3
        jne     rdcc1
        mov     nlinea,0
        jmp     rdcc1
rdcc2:  call    mexpc           ;restauro pantalla
```

```
        call    mstk
        cmp     nlinea,2        ;condicion de bandera?
        jne     rdcc21
        jmp     rdcc3
rdcc21: mov     bx,offset vrv   ;ventana para condicion
        vnt     'a'
        limpln  23,revers       ;registro = o <> xxxx
        print   1,23,123rcmp,revers
        mov     igual+3,'='
        cmp     nlinea,0        ;igualdad?
        je      rdcc22
        mov     igual+3,0d8h    ;no, desigualdad
rdcc22: print   10,20,igual,normal
        mov     gdreset,1
rdcc221:getline 10,20,nomreg
        mov     gdreset,0
        cmp     gccod,4         ;ESC?
        jne     rdcc23
        jmp     rdcf
rdcc23: cmp     gccod,1         ;recibi un reg?
        jne     rdcc221
        mov     si,offset nomreg+2 ;si recibi, es reg. normal?
        mov     cl,0            ;en CL enviare una bandera
        call    EncReg
        jnc     rdcc25
        mov     cl,1
        call    EncSreg         ;no, es de segmento?
        jnc     rdcc25
        limpln  23,revers       ;no, error
        print   1,23,noesreg,revers
        getchar
        mov     ax,getword
        cmp     al,27           ;ESC?
        je      rdcc24
        jmp     rdcc21
rdcc24: jmp     rdcf
rdcc25: ;aqui ya tengo el registro, ahora recibo el valor
        mov     aux,ax
        limpln  23,revers
        print   1,23,123campr,revers
        mov     gdreset,1
rdcc26: call    rdcc260         ;resalto el campo
        cmp     cl,0            ;registro normal?
        jne     rdcc261
        test    byte ptr aux,8  ;si, 8 o 16 bits?
        jnz     rdcc261
        getdata 17,20,dbyte,'h' ;8 bits
        jmp     rdcc262
rdcc261:getdata 15,20,dhexi,'h' ;16 bits
rdcc262:mov     gdreset,0
        cmp     gccod,7         ;ESC?
        jne     rdcc27
```

```
        jmp     rdcc4           ;si,salgo
rdcc27: cmp     gccod,0         ;tengo dato?
        jne     rdcc2e
        mov     ah,[            ;si, preparo llamada a ExCond
        cmp     ci,0            ;registro normal?
        je      rdcc28
        mov     ah,10h          ;no, de segm., bit 4 de AH = 1
rdcc28: mov     al,byte ptr [bx
        add     ah,al
        cmp     ci,0            ;reg. normal?
        jne     rdcc281
        test    byte ptr aux,8  ;si, de 8 o 16 bits?
        jnz     rdcc281
        mov     bl,dbyte+4      ;8 bits
        mov     bh,0
        jmp     rdcc282
rdcc281:mov     bx,word ptr dhex1+6 ;16 bits
rdcc282:push    ax
        usroro                  ;Pantalla de usuario<-->PRO
        pop     ax
        cmp     nlinea,0        ;igualdad?
        je      rdcc3
        jmp     rdcc4
rdcc3:  mov     al,0            ;registro = xxxx
        jmp     excond
rdcc4:  mov     al,1            ;registro <> xxxx
        jmp     excond
rdcc5:  mov     bx,offset vcban ;condicion de banderas
        int     'a'
        mov     nlinea,0        ;para cada linea...
rdcc51: call    rdcc50          ;resalto opcion actual
        getchar
        mov     al,atrprev      ;quito resaltado
        mov     di,dirprev
        call    rdcc501
        cmp     gccod,1         ;ENTER?
        je      rdcc6
        cmp     gccod,4         ;ESC?
        jne     rdcc53
        jmp     rdcf
rdcc53: cmp     gccod,5         ;especial?
        jne     rdcc5
        cmp     gcascii,72      ;arriba?
        jne     rdcc54
        dec     nlinea          ;si
        cmp     nlinea,0ffh
        jne     rdcc51
        mov     nlinea,15
        jmp     rdcc51
rdcc54: cmp     gcascii,80      ;abajo?
        jne     rdcc5
        inc     nlinea
```

```
        cmp     nlinea,16
        jne     rdcc51
        mov     nlinea,0
        jmp     rdcc51
rdcc6:  usrpro                  ;Pantalla de usuario<-->PRO
        mov     al,2            ;ya tengo la condicion
        mov     ah,nlinea
        jcc     excond
;aEjmas:mov     ax,@data
        mov     ds,ax
        mov     es,videoseg
        usrpro                  ;P. de usuario<-->PRO
rdc5:   call    pdebug
        jmp     salida

rdcd0:  posdir  10,20           ;resalto campo de segmento
rdcd0a: mov     di,pddir
        inc     di              ;calculo posicion
        mov     ah,es:[di]      ;atributo en ese lugar
        mov     atrprev,ah
        mov     dirprev,di
        mov     al,70h          ;reverso
rdcd01: mov     cx,4
rdcd02: stosb
        inc     di
        loop    rdcd02
        retn
rdcd00: posdir  17,20           ;resalto campo de offset
        jmp     rdcd0a          ;aqui incluido el RETN
rdcc0:  mov     cl,7            ;resalto uno de los tres casos
        mov     ch,nlinea       ;similar a rdcd0
        add     ch,18
        posdir  cl,ch
        mov     di,pddir
        inc     di
        mov     ah,es:[di]
        mov     atrprev,ah
        mov     dirprev,di
        mov     al,70h
rdcc01: mov     cx,17
        jmp     rdcd02          ;aqui ya se hace el RETN
rdcc50: mov     cl,19           ;resalto una de las 16 opciones
        mov     ch,nlinea
        add     ch,6
        posdir  cl,ch
        mov     di,pddir
        inc     di
        mov     ah,es:[di]
        mov     atrprev,ah
        mov     dirprev,di
        mov     al,70h
rdcc501:mov     cx,15
```

```
        jmp     rdcc02          ;aqui ya se hace el RETN
rdcc250:posdir  15,20           ;resalto campo de direccion
        mov     di,pddir
        inc     di
        mov     al,70h
        jmp     rdcd01          ;aqui ya se hace el RETN

rdj:                            ;ejecuta hasta ruptura o final
        usrpro
        mov     ax,offset rdj1
        mov     word ptr XferAddr,ax
        mov     word ptr XferAddr+2,cs
        jmp     ExTotal
rdj1:   mov     ax,@data
        mov     ds,ax
        mov     es,videoseg
        usrpro
        call    pdebug
        jmp     salida

rdra:                           ;activo/desactivo bkpt
        mov     dlin0,45h
        mov     brup,1          ;bandera BRUP=1 para activar/desactivar.
        jmp     rdrb1
rdrc:   mov     brup,2          ;crear brkpt, BRUP=2
        jmp     rdrb1
rdrb:   mov     brup,3          ;borro brkpt, BRUP=3
rdrb1:  mov     rupact,0        ;recibo nombre del Brkpt
        limpln  23,revers       ;RUPACT=ruptura actual-1
        limpln  24,normal
        print   1,23,123bkp,revers
        mov     bx,offset vrnom ;ventana para nombre
        mov     ax,ds
        mov     si,offset uline
        vnt     'a'
        mov     gdreset,1
rdb11a: call    rdr0            ;muestra ventana de bkpts
rdb11:  getline 12,19,nombrk    ;recibo nombre del brkpt
        mov     gdreset,0
        cmp     gccod,4         ;ESC?
        jne     rdb12
        jmp     rdrbf2
rdb12:  cmp     gccod,1         ;vale?
        jne     rdb13
        cmp     byte ptr nombrk[1],0 ;si, pero esta vacio?
        jne     rdb14
        jmp     rdrbf2          ;si,termino
rdb13:  cmp     gccod,5         ;especial?
        jne     rdb11
        cmp     gcascii,73      ;PGUP?
        je      rdb132
        cmp     gcascii,81      ;PGDN?
```

```
        jne     rdb11           ;no vale
        cmp     rupact,247      ;PGDN
        je      rdb11
        add     rupact,8
        mov     al,16
        mov     cl,rupact
        mul     cl
        add     ax,offset TablaBrk
        mov     si,ax
        cmp     byte ptr [si],0
        jne     rdb11a
        sub     rupact,8
        jmp     rdb11
rdb132: cmp     rupact,8        ;PGUP
        jb      rdb11
        sub     rupact,8
        jmp     rdb11a
rdb14:  mov     cl,1            ;acceso  a bkpts solo por nombre
        mov     si,offset nombrk+2
        mov     ai,bruo
rdrb2:  cmp     al,1            ;activo/desactivo brkpt?
        jne     rdrb3
        call    TogBrkPt        ;si, toggle
        jc      rdrb21
        jmp     rdrbf1
rdrb21: limpln  23,revers
        print   1,23,noesbrk,revers
        jmp     rdrbf
rdrb3:  cmp     al,2            ;creo brkpt?
        je      rdrb31
        jmp     rdrb4
rdrb31: limpln  23,revers       ;limpio lineas 23 y 24
        mov     ax,RegCs        ;bkpt en cierto punto
        mov     word ptr dhex1+6,ax
        xchg    al,ah           ;Hago que CS usuario sea seg. predef.
        push    ax              ;para lo que simulo condiciones
        hexasc
        pop     ax
        xchg    bl,bh
        mov     word ptr dhex1+2,bx
        mov     bufcorr,bx
        mov     al,ah
        hexasc
        xchg    bl,bh
        mov     word ptr dhex1+4,bx
        mov     bufcorr+2,bx
        mov     byte ptr dhex1+1,4
rup1:   mov     bx,offset vrdir ;ventana para recibir direccion
        mov     ax,ds
        mov     si,offset bufcorr
        vnt     'a'
        print   1,23,123cd,revers
```

```
        mov     gdreset,1
        cmp     byte ptr bufcorr,'_'
        je      rup11
        mov     ax,'__'
        mov     bufcorr,ax
        mov     bufcorr+2,ax
        jmp     rup2
rup11:  call    rup0            ;resalto campo de segmento
        getdata 29,21,dhex1,'h' ;recibo segmento
        mov     al,atrprev      ;quito resaltado
        mov     di,dirprev
        call    rdcd01
        mov     gdreset,0
        cmp     gccod,0         ;dato valido?
        je      rup2
        cmp     gccod,7         ;ESC para abortar
        jne     rup12
        jmp     rdrbf2
rup12:  cmp     gccod,5         ;TAB para pasar
        jne     rup11
rup2:   mov     gdreset,1
rup21:  call    rup00           ;resalto campo de offset
        getdata 36,21,dhex2,'h' ;recibo offset
        mov     al,atrprev      ;quito resaltado
        mov     di,dirprev
        call    rdcd01
        mov     gdreset,0
        cmp     gccod,0         ;dato valido?
        je      rup3
        cmp     gccod,7         ;ESC para abortar
        jne     rup22
        jmp     rdrbf2
rup22:  cmp     gccod,5         ;TAB para pasar
        jne     rup21
        jmp     rup1
rup3:   mov     si,offset noesbrk+2 ;todo listo para crear bkpt
        mov     dx,word ptr dhex1+6
        mov     bx,word ptr dhex2+6
        call    AddBrkPt
        jmp     rdrbf1
rdrb4:  call    DelBrkPt        ;borro brkpt
        jnc     rdrbf1
        lineln  23,revers
        print   1,23,noesbrk,revers
        jmp     rdrbf
rdrbf1: call    rdr0            ;actualizo ventana de rupturas
        call    ecod
        lineln  23,revers       ;salida OK con espera
        print   1,23,listo,revers
rdrbf:  getchar                 ;salida con espera
rdrbf2: call    pdebug          ;salida. Actualizo pantalla
        jmp     salida
```

```
rup0:   posdir  1dh,15h         ;resalto campo de segmento
rup01:  mov     di,pdoir
        inc     di              ;calculo posicion
        mov     ah,es:[di]      ;atributo en ese lugar
        mov     atrprev,ah
        mov     dirprev,di
        mov     al,70h          ;reverso
        jmp     rdr01           ;aqui esta el ret incluido
rup00:  posdir  24h,15h
        jmp     rup01           ;aqui incluigo el ret
rdr0:   ;aca llega RUPACT con el # de ruptura (-1) desde el que debe
        ;empezar la ventana. Crea BUFRUP y muestra la ventana.
        mov     dl,rupact       ;DL=# de rup. -1 a mostrarse
        mov     di,offset BufRup ;primero borro buffer
        mov     cx,168
        mov     al,0
        push    es
        push    ds
        pop     es
        rep     stosb
        pop     es
        mov     si,offset TablaBrk
        mov     di,offset BufRup
        mov     nlinea,0        ;para cada linea...
rdr01:  mov     al,16
        mul     dl
        mov     bp,ax           ;SI+BP apunta a la ruptura a mostrarse
        cmp     byte ptr ds:[si+bp],0 ;entrada vacia?
        jne     rdr02
        jmp     rdr03           ;si ,fue la ultima ruptura
rdr02:  mov     al,ds:[si+bp]   ;pongo # de brkpt
        hexasc
        mov     [di],bh
        mov     [di+1],bl
        mov     al,ds:[si+bp+1] ;marco con * si es activo
        cmp     al,0
        je      rdr021
        mov     byte ptr[di+2],'*'
rdr021: mov     ax,ds:[si+bp+2] ;copio 10 bytes del nombre
        mov     [di+3],ax
        mov     ax,ds:[si+bp+4]
        mov     [di+5],ax
        mov     ax,ds:[si+bp+6]
        mov     [di+7],ax
        mov     ax,ds:[si+bp+8]
        mov     [di+9],ax
        mov     ax,ds:[si+bp+10]
        mov     [di+11],ax
        mov     al,ds:[si+bp+15]        ;nuestro segmento
        hexasc
        xchg    bl,bh
        mov     [di+13],bx
```

```
        mov     al,ds:[si+bp+14]
        nexasc
        xchg    bl,bh
        mov     [di+15],bx
        mov     al,ds:[si+bp+13]        ;nuestro offset
        nexasc
        xchg    bl,bh
        mov     [di+17],bx
        mov     al,ds:[si+bp+12]
        nexasc
        xchg    bl,bh
        mov     [di+19],bx
        inc     nlinea          ;veo si termina la ventana
        inc     dl
        add     di,21
        cmp     nlinea,8        ;llego a ultima linea?
        je      rdr03
        cmp     dl,0            ;ultima ruptura?
        je      rdr03
        jmp     rdr01
rdr03:  mov     bx,offset vrup  ;nuestro la ventana
        mov     ax,ds
        mov     si,offset outrup
        int     'a'
        retn

rdrm:                           ;muestra ventana de bkpts
        mov     rupact,0
        linepln 23,revers
        linepln 24,normal
        print   1,23,123bkp,revers
rdrmlla:call    rdr0            ;muestra ventana de bkpts
rdrmll: getchar                 ;recibo orden del teclado
        cmp     gccod,4         ;ESC?
        jne     rdrm12
        jmp     rdrmf
rdrm12: cmp     gccod,5         ;especial?
        jne     rdrm11
rdrm13: cmp     gcascii,73      ;PGUP?
        je      rdrm132
        cmp     gcascii,81      ;PGDN?
        jne     rdrm11          ;no vale
        cmp     rupact,247      ;PGDN
        ja      rdrm11
        add     rupact,8
        mov     al,16
        mov     cl,rupact
        mul     cl
        add     ax,offset TablaBrk
        mov     si,ax
        cmp     byte ptr [si],0
        jne     rdrmlla
```

```
        sub     rucact,9
        jae     rdrm11
rdrm132:cmp     rucact,8        ;PEQUE
        jc      rdrm11
        sub     rucact,8
        jcc     rdrm132
rdrm1:  call    gpem            ;restauro pantalla
        call    greg
        call    gpsw
        jmp     salida

rdi:    limpln  23,revers
        limpln  24,normal
        print   1,23,instruc,revers
        mov     gdreset,1
rdi1:   getline 1,24,insbuff    ;recibo nemotecnico
        mov     goreset,0
        cmp     gccod,4
        jne     rdi2
        jmp     rdif            ;ESC
rdi2:   cmp     gccod,1
        jne     rdi1
        mov     bl,insbuff+1    ;ENTER, dato valido
        mov     bh,0
        mov     si,offset insbuff+2
        mov     byte ptr[bx+si],0
        mov     word ptr XferAddr,offset rdi3   ;direccion de retorno
        mov     word ptr XferAddr+2,cs
        jmp     ExIns           ;ejecuto
rdi3:   mov     ax,@data
        mov     ds,ax
        mov     es,videoseg
        jnc     rdi4
        limpln  23,revers       ;nemotecnico incorrecto
        print   1,23,errens,revers
        jmp     rdif
rdi4:   call    pdebug          ;todo bien
        limpln  23,revers
        print   1,23,listo,revers
rdif:   getchar
        jmp     salida

rdee:                           ;ensamble una instruccion
        call    check2
        mov     banens,1        ;pongo bandera
        limpln  23,revers       ;limpio lineas 23 y 24
        limpln  24,normal
        mov     ax,RegCs        ;ensamble en cierto punto
        mov     word ptr dhex1+6,ax
        xchg    al,ah           ;Hago que CS usuario sea seg. predef.
        push    ax              ;para lo que simulo condiciones
        hexasc
```

```
        pop     ax
        xchg    bl,bh
        mov     word ptr dhex1+2,bx
        mov     bufcorr,bx
        mov     al,ah
        hexasc
        xchg    bl,bh
        mov     word ptr dhex1+4,bx
        mov     bufcorr+2,bx
        mov     byte ptr dhex1+1,4
rdee1:  mov     bx,offset vens  ;ventana para recibir direccion
        mov     ax,ds
        mov     si,offset bufcorr
        vnt     'a'
        print   1,23,123cd,revers
        mov     gdreset,1
        cmp     byte ptr bufcorr, '_'
        je      rdee11
        mov     ax,'__'
        mov     bufcorr,ax
        mov     bufcorr+2,ax
        jmp     rdee2
rdee11: call    rdcd0           ;resalto campo de segmento
        getdata 10,20,dhex1,'h' ;recibo segmento
        mov     al,atrprev      ;quito resaltado
        mov     di,dirprev
        call    rdcd01
        mov     gdreset,0
        cmp     gccod,0         ;dato valido?
        je      rdee2
        cmp     gccod,7         ;ESC para abortar
        jne     rdee12
        jmp     rdeef
rdee12: cmp     gccod,5         ;TAB para pasar
        jne     rdee11
rdee2:  mov     gdreset,1
rdee21: call    rdcd00          ;resalto campo de offset
        getdata 17,20,dhex2,'h' ;recibo offset
        mov     al,atrprev      ;quito resaltado
        mov     di,dirprev
        call    rdcd01
        mov     gdreset,0
        cmp     gccod,0         ;dato valido?
        je      rdee3
        cmp     gccod,7         ;ESC para abortar
        jne     rdee22
        jmp     rdeef
rdee22: cmp     gccod,5         ;TAB para pasar
        jne     rdee21
        jmp     rdee1
rdee3:  mov     ax,word ptr dhex1+6 ;direccion para mostrar codigo
        mov     segcod,ax
```

```
        mov     ax,word ptr dhex2+6
        mov     offcod,ax
        call    mcod
rdee31: print   1,23,123ens,revers ;indicaciones
        mov     gdreset,1
        call    mexpc
        call    mstk
rdeea:  limpln  24,normal
        call    rdee0           ;actualizo direcciones
rdeeal: getline 1,24,insbuff    ;recibo nemotecnico
        mov     gdreset,0
        cmc     gccod,4         ;ESC?
        je      rdeef
        cmp     gccod,1         ;Vale?
        jne     rdeeal
        mov     bl,insbuff[1]   ;BX = # de caracteres
        cmp     bl,0
        je      rdeef
        xor     bh,bh
        mov     si,offset insbuff+2
        mov     byte ptr [si+bx],0
        mov     bx,es
        mov     es,word ptr dhex1+6
        mov     di,word ptr dhex2+6
        call    assem           ;ensamblo
        jnc     rdeex
        mov     es,bx           ;nemotecnico incorrecto
        limpln  23,revers
        print   1,23,errens,revers
        getchar
        limpln  23,revers
        jmp     rdee31
rdeex:  add     word ptr dhex2+6,cx ;preparo prox. ensamblado
        mov     es,bx
        call    mcod
        call    pdebug
        mov     gdreset,1
        jmp     rdeea
rdeef:  mov     banens,0        ;quito bandera
        call    pdebug          ;actualizo pantalla
        jmp     salida
rdee0:  posdir  12,23
        mov     di,pddir
        mov     ax,word ptr dhex1+6 ;pone en linea 23 la direccion
        call    rdee00
        add     di,10
        mov     ax,word ptr dhex2+6
        call    rdee00
        retn
rdee00: xchg    al,ah
        push    ax
        hexasc
```

```
        dec     al
        mov     es:[di].ch
        mov     es:[di+2].bl
        mov     al,ah
        keyasc
        mov     es:[di+4],ch
        mov     es:[di+6],bl
        retn

rdeb:                   ;ensamabla y busca en el segmento de codigo
        limpln  0,revers
        limpln  1,revers
        print   10,0,ensw1,revers
        print   10,1,ensw2,revers
        limpln  23,revers
        limpln  24,normal
        print   1,23,ensdusc,revers ;indicaciones
        mov     gdreset,1
rdeba:  limpln  24,normal
rdeba1: getline 1,24,insbuff    ;recibe nemotecnico
        mov     gcreset,0
        cmp     gccod,4         ;ESC?
        jne     rdeba2
        jmp     rdebf
rdeba2: cmp     gccod,1         ;Vale?
        jne     rdeba1
        mov     bl,insbuff[1]   ;BX = # de caracteres
        cmp     bl,0
        jne     rdeba3
        jmp     rdebf
rdeba3: xor     bh,bh
        mov     si,offset insbuff+2
        mov     byte ptr [si+bx],0 ;termino buffer con 0
        mov     bx,es
        mov     ax,ds
        mov     es,ax
        mov     di,offset InsEns
        call    assem
        mov     es,bx
        jnc     rdebx
        limpln  23,revers
        print   1,23,errens,revers
        getchar
        jmp     rdebf
rdebx:  push    RegIp
        mov     bx,cx           ;busco en el codigo
rdebx1: mov     es,RegCs        ;inicio de busqueda
        mov     di,RegIp        ;en CS:IP del usuario.
        mov     si,offset InsEns
        mov     cx,0ffffh
        call    search
        cmp     cx,0            ;CX=0 si no encontro
```

```
        jne     rdeb1
        pop     RegIp           ;no encontro
        mov     es,videoseg
        limpln  22,revers
        print   1,23,NoEnsB,revers
        getchar
        jmp     rdeb1
rdeb1:  mov     es,videoseg
        limpln  23,revers       ;si encontro,
        mov     offcod,di
        push    bx
        push    di
        call    mcod
        pop     di
        pop     bx
        print   1,23,sigo?,revers ;busco proxima?
rdeb11: getchar
        mov     ax,getword
        cmp     al,1bh
        jne     rdeb2
        jmp     rdeb3           ;ESC
rdeb2:  cmp     al,0dh
        jne     rdeb11
        inc     di              ;ENTER, sigo buscando
        mov     RegIp,di
        jmp     rdeox1
rdeb3:  pop     RegIp
rdebf:  mov     si,offset l0ini ;preparo y muestro linea 0
        call    mlin0
        mov     bx,offset vIin1
        mov     ax,ds
        mov     si,offset liini
        mov     t'a'
        call    pdebug          ;actualizo pantalla
        jmp     salida

rdm1:                           ;lleno un area de memoria
        limpln  23,revers
        limpln  24,normal
        mov     bx,offset vl1   ;ventana para pedir datos
        mov     ax,dir1
        xchg    al,ah
        mov     bufm1+2,ax
        mov     ax,dir1+2
        xchg    al,ah
        mov     bufm1,ax
        mov     ax,ds
        mov     si,offset bufm1
        mov     t'h'
        print   1,23,bytes?,revers
        mov     gdreset,1       ;recibo # de bytes
rdm11:  call    rdm10           ;resalto campo de #bytes
```

```
        getdata 20,19,dhex1,'h'
        mov     gdreset,0
        mov     al,atrprev
        mov     di,dirprev
        call    rdm101
        cmp     gccod,7
        jne     rdm12
        jmp     rdm1f           ;ESC
rdm12:  cmp     gccod,0
        jne     rdm11
        limpln  23,revers
        print   1,23,valor?,revers
        mov     gdreset,1       ;aqui ya hay el #bytes en DHEX1+6
rdm13:  call    rdm100
        getdata 22,19,dbyte,'h'
        mov     gdreset,0
        mov     al,atrprev
        mov     di,dirprev
        call    rdm1001
        cmp     gccod,7
        je      rdm1f           ;ESC
        cmp     gccod,0
        jne     rdm13
rdm14:  push    es              ;aqui ya tengo el valor
        mov     es,dir1+2
        mov     di,dir1         ;ES:DI apunta al inicio del area
        mov     cx,word ptr dhex1+6
        mov     al,dbyte+4
        call    fill            ;lleno el area con el valor de AL
        pop     es
        limpln  23,revers
        print   1,23,listo,revers
        call    pdebug
        getchar
        jmp     rdm1f1
rdm1f:  call    pdebug
rdm1f1: jmp     salida
rdm10:  posdir  20,18           ;resalto campo de #bytes
        mov     di,pddir
        inc     di
        mov     al,es:[di]
        mov     atrprev,al
        mov     dirprev,di
        mov     al,70h
rdm101: mov     cx,4
rdm102: stosb
        inc     di
        loop    rdm102
        retn
rdm100: posdir  22,19
        mov     di,pddir
        inc     di
```

```
        mov     al,es:[di]
        mov     atrprev,al
        mov     dirprev,di
        mov     al,70h
rdb100:mov      cx,2
        jmp     rdb102          ;esto incluye el retn

rdb:                            ;busca un string en segmento actual
        limpln  23,revers
        limpln  24,normal
        print   1,23,buscasc?,revers
rdbb1:  getchar                 ;espero respuesta H, A o ESC
        mov     ax,getword
        cmp      al,1bh
        jne     rdbb12
        jmp     rdbbf2          ;ESC
rdbb12: cmp     al,'A'
        je      rdbb2
        cmp     al,'a'
        je      rdbb2
        cmp     al,'H'
        je      rdbb13
        cmp     al,'h'
        jne     rdbb1           ;respuesta incorrecta
rdbb13: jmp     rdbb4
rdbb2:  mov     gdreset,1       ;Secuencia en ASCII
        limpln  23,revers
        print   1,23,busca?,revers
rdbb21: getline 1,24,insbuff    ;leo secuencia
        mov     gdreset,0
        cmp     gccod,4
        jne     rdbb22
        jmp     rdbbf2          ;ESC
rdbb22: cmp     gccod,1
        jne     rdbb21
        mov     bl,insbuff+1    ;vale, pero si esta vacio salgo
        cmp     bl,0
        jne     rdbb23
        jmp     rdbbf2
rdbb23: mov     bh,0            ;preparo BX para llamar a rutina de busqueda
        jmp     rdbb6
rdbb4:  limpln  23,revers       ;Secuencia en HEX
        print   1,23,busca?,revers
        mov     bx,0
rdbb41: mov     cl,bl
        mov     al,3
        mul     cl
        inc     al
        mov     cl,al
        mov     gdreset,1
rdbb42: call    rdbb0           ;resalto sitio actual
        getdata cl,24,dbyte,'h'
```

```
        mov     qdreset,0
        mov     al,atrprev      ;quito resaltado
        mov     di,dirprev
        call    rdmb01
        cmp     qcmod,7
        jne     rdmb43
        jmp     rdmbf2          ;ESC
rdmb43: cmp     qcmod,9
        jne     rdmb42
        cmp     dbyte+1,0       ;0 datos?
        je      rdmb5           ;si, pasar a buscar
        mov     si,offset insbuff+2 ;vale,guardo en INSBUFF+2
        mov     al,dbyte+4
        mov     [si+bx],al
        inc     bl
        cmp     bl,24
        jne     rdmb41
rdmb5:  cmp     bl,0
        jne     rdmb51
        jmp     rdmbf2
rdmb51: mov     insbuff+1,bl
rdmb6:  mov     cx,0ffffh       ;preparo llamada
        mov     bl,insbuff+1
        mov     bh,0
        mov     si,offset insbuff+2
        mov     es,dir1+2
        mov     di,dir1
        call    search
        mov     dir1+2,es
        mov     dir1,di
        mov     es,videoseg
        cmp     cx,0            ;CX=0 si no encontro
        jne     rdmb7
        liapln  23,revers       ;no encontro
        print   1,23,NoHayStr,revers
        inc     dir1
        jmp     rdmbf
rdmb7:  call    preopc  ;encontro, muestro en ventana
        call    mmem
        liapln  23,revers
        print   1,23,sigo?,revers ;busco proxima?
rdmb72: getchar
        mov     ax,getword
        cmp     al,1bh
        je      rdmbf2          ;ESC, salgo
        cmp     al,0dh
        jne     rdmb72
        inc     dir1
        jmp     rdmb6           ;enter, busco otra vez
rdmbf:  getchar
rdmbf2: jmp     salida
rdmb0:  posdir  cl,24           ;resalto campo para byte actual
```

```
        mov     di,pddir
        inc     di
        mov     al,es:[di]
        mov     atrprev,al
        mov     dirprev,di
        mov     al,70h
rcmd01: stosb
        inc     di
        stosb
        retn
rdmc:   ;copia desde la ventana principal a otra area
        limpln  23,revers       ;limpio lineas 23 y 24
        limpln  24,normal
        mov     ax,dir1
        xchg    al,ah
        mov     bufmi+2,ax
        mov     ax,dir1+2
        xchg    al,ah
        mov     bufmi,ax
        mov     ax,RegDs
        mov     word ptr dhex1+6,ax
        xchg    al,ah           ;Hago que DS usuario sea seg. predet.
        push    ax              ;para lo que simulo condiciones
        mov     bufmi+4,ax
        hexasc
        pop     ax
        xchg    bl,bh
        mov     word ptr dhex1+2,bx
        mov     bufcorr,bx
        mov     al,an
        hexasc
        xchg    bl,bh
        mov     word ptr dhex1+4,bx
        mov     bufcorr+2,bx
        mov     byte ptr dhex1+1,4
rdmc1:  mov     bx,offset vcopia ;ventana para recibir direccion
        mov     ax,ds
        mov     si,offset bufmi
        vnt     'h'
        print   1,23,123cd,revers
        mov     gdreset,1
        cmp     byte ptr bufcorr,'_'
        je      rdmc10
        mov     ax,'__'
        mov     bufcorr,ax
        mov     bufcorr+2,ax
        mov     ax,0
        mov     bufmi+4,ax
        jmp     rdmc2
rdmc10: print   15,18,ulinmi,revers
rdmc11: call    rdmc0           ;resalto campo de segmento
        getdata 15,18,dhex1,'h' ;recibo segmento
```

```
        mov     al,atrprev      ;quito resaltado
        mov     di,dirprev
        call    rdmc01
        mov     gdreset,0
        cmp     gccod,0         ;dato valido?
        je      rdmc2
        cmp     gccod,7         ;ESC para abortar
        jne     rdmc12
        jmp     rdmcf
rdmc12: cmp     gccod,5         ;TAB para pasar
        jne     rdmc11
rdmc2:  mov     gdreset,1
rdmc21: call    rdmc00          ;resalto campo de offset
        getdata 20,18,dhex2,'h' ;recibo offset
        mov     al,atrprev      ;quito resaltado
        mov     di,dirprev
        call    rdmc01
        mov     gdreset,0
        cmp     gccod,0         ;dato valido?
        je      rdmc3
        cmp     gccod,7         ;ESC para abortar
        jne     rdmc22
        jmp     rdmcf
rdmc22: cmp     gccod,5         ;TAB para pasar
        jne     rdmc21
        jmp     rdmc1
rdmc3:  print   1,23,bytes?,revers ;recibo el # de bytes
        mov     bp,word ptr dhex1+6 ;libero dhex1
        mov     gdreset,1
        call    rdmc30          ;resalto campo de #bytes
rdmc31: getdata 20,19,dhex1,'h'
        mov     gdreset,0
        cmp     gccod,7
        je      rdmcf           ;ESC
        cmp     gccod,0
        jne     rdmc31          ;no vale
rdmc4:  mov     es,bp           ;aqui tengo todo listo, copio
        mov     di,word ptr dhex2+6 ;ES:DI-->destino
        mov     cx,word ptr dhex1+6 ;#bytes
        mov     si,dir1
        mov     ds,dir1+2       ;DS:SI-->origen
        call    copy
        mov     ax,@data
        mov     ds,ax
        mov     es,videoseg
        limpln  23,revers
        print   1,23,listo,revers
        call    pdebug
        getchar
        jmp     rdmcf1
rdmcf:  call    pdebug
rdmcf1: jmp     salida
```

```
rdmc0:  posdir  15,18           ;resalto campo de segmento
        mov     di,pddir
        inc     di
        mov     al,es:[di]
        mov     atrprev,al
        mov     dirprev,di
        mov     al,70h
rdmc01: jmp     rdml01          ;Esto incluye el RETN
rdmc00: posdir  20,18           ;resalto campo de offset
        mov     di,pddir
        inc     di
        mov     al,es:[di]
        mov     atrprev,al
        mov     dirprev,di
        mov     al,70h
        jmp     rdml01          ;Esto incluye el RETN
rdmc30: posdir  20,19           ;resalto campo de #bytes
        mov     di,pddir
        inc     di
        mov     al,es:[di]
        mov     atrprev,al
        mov     dirprev,di
        mov     al,70h
        jmp     rdml01          ;Esto incluye el RETN

rdfun:  mov     dinfo,49h
        cmp     otroaux,0       ;si se digito ENTER, mostrar funciones
        jne     rdfun1
        jmp     rdfm
rdfun1: cmp     mopaux,0        ;ASCII normal si AL=0
        je      rdf11
        jmp     rdf6
rdf11:  cmp     mopauxi,59      ;F1?
        jne     rdf12
        call    rd_             ;editor en pantalla
        jmp     rdff
rdf12:  cmp     mopaux1,60      ;F2?
        jne     rdf13
        jmp     rdff
rdf13:  cmp     mopaux1,61      ;F3?
        jne     rdf14
        call    rdf3m
        jmp     rdff
rdf14:  cmp     mopaux1,62      ;F4?
        jne     rdf15
        usrpro                  ;Muestro pantalla de usuario
rdu1:   getchar                 ;espero ESC para regresar
        mov     ax,getword
        cmp     al,27           ;ESC?
        jne     rdu1
        usrpro                  ;vuelvo a pantalla de PRO
```

```
        jmp     rdff
rdf15:  cmp     mopaux1,83      ;F5?
        jne     rdf16
        jmp     rdff
rdf16:  cmp     mopaux1,64      ;F6?
        jne     rdf17
        jmp     rdff
rdf17:  cmp     mopaux1,65      ;F7?
        jne     rdf18
        xor     toggcorre,1     ;corre nesta direccion/condicion.
        mov     al,6            ;indico en pantalla:
        mov     cl,toggcorre    ;Direcc o Condic
        mul     ci
        mov     si,ax
        add     si,offset opccorr
        mov     di,offset llini+49
        mov     ax,[si]
        mov     [di],ax
        mov     ax,[si+2]
        mov     [di+2],ax
        mov     ax,[si+4]
        mov     [di+4],ax
        mov     bx,offset vlin1
        mov     ax,ds
        mov     si,offset llini
        int     'a'
        jmp     rdff
rdf18:  cmp     mopaux1,66      ;F8?
        jne     rdf19
        xor     toggsub,1       ;permite/impide ingreso en
        mov     al,2            ;subrutinas, reps o loops.
        mov     cl,toggsub      ;indico en pantalla:
        mul     cl              ;Si o No Subr.
        mov     si,ax
        add     si,offset opcpaso
        mov     di,offset llini+57
        mov     ax,[si]
        mov     [di],ax
        mov     bx,offset vlin1
        mov     ax,ds
        mov     si,offset llini
        int     'a'
        jmp     rdff
rdf19:  cmp     mopaux1,67      ;F9?
        je      rdf19a
        jmp     rdf1a
rdf19a: mov     bx,offset v19
        mov     ax,ds
        mov     si,offset uline
        int     'a'
        mov     gdreset,1       ;recibo un decimal
rdf191: getdata 41h,4,ddec,'d'
```

```
        mov     goreset,0
        cmp     gccoc,7
        jne     rdf192
        jmp     rdf19f          ;ESC
rdf192: cmp     gccoc,0
        jne     rdf191
        posdir  4ah,4
        mov     di,posdir
        mov     ax,word ptr ddec+7 ;vale
        xchg    al,ah           ;nuestro equivalencia hex
        push    ax              ;en la pantalla
        hexasc
        mov     es:[di],bh
        mov     es:[di+2],bl
        pop     ax
        mov     al,ah
        hexasc
        mov     es:[di+4],bh
        mov     es:[di+6],bl
        limpln  23,revers
        limpln  24,normal
        print   1,23,teccont,revers
        getchar
rdf19f: call    mreg
        call    mpsw
        jmp     rdff
rdf1a:  cmp     mopaux1,68      ;F10?
        je      rdfa
        jmp     rdff
rdfa:   limpln  23,revers       ;si
        limpln  24,normal
        mov     bx,offset vasc1 ;nuestro ventana de codigo ASCII
        mov     ax,ds
        mov     si,offset uno
        vnt     'a'
        mov     bx,offset vasc2
        vnt     'a'
        print   1,23,TecCont,revers
        getchar
        call    pdebug
        jmp     rdff
rdf6:   cmp     mopaux,'F'      ;ASCII normales
        je      rdfm            ;si es F o f nuestro explico funciones
        cmp     mopaux,'f'
        jne     rdff
rdfm:   mov     bx,offset vfunc ;nuestro ventana
        vnt     'a'
        limpln  23,revers
        limpln  24,normal
        print   1,23,TecCont,revers
        getchar
        call    pdebug
```

```
rd44:   mov     modaux,0
        mov     modaux1,0
        jmp     salida

rd_1:                           ;edicion en pantalla.
;Orden en que se atendera a las ventanas: 1 Registros, 2 Banderas
;3 Codigo, 4 direccion memorial, 5 direcciones memoria2, 6 memoria1
;hex, 7 memoria1 ascii, 8 memoria2 hex, 9 memoria2 ascii.
;Se usa para leer datos dos rutinas: GETCHAR y GETDATA, que al salir
;dan codigos sobre la forma en que se produjo la salida. En esta
;parte del programa el usuario tendra opciones generales como:
;       TAB             pasa a proxima ventana.
;       SHIFT TAB       pasa a ventana anterior.
;       ESC             sale del editor al menu.
;Existen opciones aplicables de forma diferente en cada rutina.
rd_1:   ;Cambios en ventana que muestra registros.
        ;Usar Flechas para moverse o para no alterar un registro.
        mov     dl,0            ;reg. actual = 0P
        mov     gdreset,1
        mov     si,offset l0reg ;LINEA0
        call    mln0
        limpln  1,revers        ;lineas 1, 23 y 24
        limpln  23,revers
        limpln  24,normal
        print   15,1,11reg.revers
        print   3,23,123d_,revers
        print   3,24,124d_,normal
rd_a:   mov     bx,offset vr    ;ventana para cambios
        mov     ax,ds
        mov     si,offset uline
        vnt     'a'
rd_11:  call    rd_10           ;resalta reg. actual
        getdata 1fh,6,dhex1,'h'
        mov     al,atrprev
        mov     di,dirprev
        mov     gdreset,0
        call    rd_102
        cmp     gccod,5
        jb      rd_12
        cmp     gccod,8
        jae     rd_11
        jmp     rd_1f
rd_12:  cmp     gccod,3         ;-->?
        jne     rd_13
rd_121: inc     dl              ;siguiente registro
        cmp     dl,13
        jne     rd_122
        mov     dl,0
rd_122: jmp     rd_11
rd_13:  cmp     gccod,4         ;<--?
        jne     rd_14
        dec     dl              ;registro anterior
```

```
        cmp     al,0ffh
        jne     rd_131
        mov     dl,12
rd_131: jmp     rd_11
rd_14:  cmp     qccod,1         ;arriba?
        jne     rd_15
        cmp     dl,4
        jc      rd_141
        inc     dl
rd_141: sub     dl,4
        jnb     rd_142
        add     dl,4
rd_142: jmp     rd_11
rd_15:  cmp     qccod,2         ;abajo?
        jne     rd_16
        cmp     dl,0
        jne     rd_151
        inc     dl
rd_151: add     dl,3
        cmp     dl,12
        jbe     rd_152
        sub     dl,3
rd_152: jmp     rd_11
rd_16:  mov     qdreset,1
        mov     bx,word ptr dhex1+6     ;dato valido.
        mov     al,dl
        actuar
        dw      offset rd_bp,offset rd_ds,offset rd_si,offset rd_ax
        dw      offset rd_es,offset rd_di,offset rd_bx
        dw      offset rd_cs,offset rd_ip,offset rd_cx
        dw      offset rd_ss,offset rd_sp,offset rd_dx
rd_bp:  mov     regbp,bx
        jmp     rd_161
rd_ds:  mov     regds,bx
        jmp     rd_161
rd_si:  mov     regsi,bx
        jmp     rd_161
rd_ax:  mov     regax,bx
        jmp     rd_161
rd_es:  mov     reges,bx
        jmp     rd_161
rd_di:  mov     regdi,bx
        jmp     rd_161
rd_bx:  mov     regbx,bx
        jmp     rd_161
rd_cs:  mov     regcs,bx
        jmp     rd_161
rd_ip:  mov     regip,bx
        jmp     rd_161
rd_cx:  mov     regcx,bx
        jmp     rd_161
rd_ss:  mov     regss,bx
```

```
        jmp     rd_161
rd_sp:  mov     regsp,bx
        jmp     rd_161
rd_dx:  mov     regdx,bx
rd_161: push    dx
        call    pdebug
        pop     dx
        jmp     rd_a
rd_1f:  mov     ax,regcs
        mov     segcod,ax
        mov     ax,regip
        mov     offcod,ax
        call    mcod
        mov     bx,offset vpro
        mvt     'a'
        call    mpsw
        cmp     gccod,5         ;TAB?
        jne     rd_1f1
        jmp     rd_2
rd_1f1: cmp     gccod,6         ;SHIFT TAB?
        jne     rd_1f2
        jmp     rd_7
rd_1f2: jmp     rd_f            ;ESC.

rd_10:  mov     bl,dl           ;muestro registro en reverso
        mov     cl,33h          ;CL,CH tendra la posicion del
        mov     ch,3            ;registro que debe resaltarse.
        cmp     dl,4
        jb      rd_101          ;primera fila
        inc     ch
        sub     bl,3
        cmp     dl,7
        jb      rd_101          ;segunda fila
        inc     ch
        sub     bl,3
        cmp     dl,10
        jb      rd_101          ;tercera fila
        inc     ch              ;cuarta fila
        sub     bl,3
rd_101: mov     al,8
        mul     bl
        add     cl,al           ;CL,CH listos.
        posdir  cl,ch
        mov     di,pddir
        inc     di
        mov     al,70h          ;video reverso.
        mov     ah,es:[di]      ;atributo que habia antes
        mov     atrprev,ah
        mov     dirprev,di
rd_102: stosb                   ;uso esto como otra subrutina
        inc     di
        stosb
```

```
        inc     di
        stosb
        inc     di
        stosb
        retn

rd_2:   ;Cambios en banderas del microprocesador.
        ;Flechas --) y (-- para ir a otra bandera.
        mov     dl,0            ;bandera actual OF
        mov     gdreset,1
        mov     si,offset l0ban ;linea 0
        call    mlin0
        limpln  1,revers        ;linea 1
        print   16,1,l1ban,revers
rd_2a:  mov     bx,offset vb    ;ventana para cambios
        mov     ax,ds
        mov     si,offset uline
        mnt     'a'
rd_21:  call    rd_20           ;resalta banoera actual
        getdata 23h,12,dbin,'b'
        mov     al,atrprev
        mov     di,dirprev
        mov     gdreset,0
        call    rd_201          ;quito resaltado
        cmp     gccod,0
        jne     rd_211
        mov     gdreset,1       ;dato valido.
        call    rd_210          ;altera variable Flags
        push    dx
        call    mpsm            ;actualiza pantalla
        pop     dx
        jmp     rd_2a
rd_211: cmp     gccod,3
        jb      rd_21
rd_22:  cmp     gccod,3
        jne     rd_23
        inc     dl              ;siguiente bandera
        cmp     dl,8
        jne     rd_221
        mov     dl,0
rd_221: jmp     rd_21
rd_23:  cmp     gccod,4
        jne     rd_24
        dec     dl              ;bandera anterior
        cmp     dl,0ffh
        jne     rd_231
        mov     dl,7
rd_231: jmp     rd_221
rd_24:  cmp     gccod,8
        jb      rd_241
        jmp     rd_21
rd_241: mov     ax,regcs
```

```
        mov     segcod,ax
        mov     ax,regip
        mov     offcod,ax
        call    mcod
        call    mseg
        call    mpsw
        cmp     gccod,5
        jne     rd_25
        jmp     rd_3            ;TAB
rd_25:  cmp     gccod,6
        jne     rd_26
        jmp     rd_1            ;SHIFT TAB
rd_26:  jmp     rd_f            ;ESC

rd_20:  mov     cl,38           ;resalto bandera actual
        add     cl,dl
        add     cl,dl
        mov     ch,5
        posdir  cl,ch
        mov     di,pddir
        inc     di
        mov     al,70h          ;video reverso
        mov     ah,es:[di]      ;atributo previo
        mov     atrprev,ah
        mov     dirprev,di
rd_201: stosb                   ;pone atributo (AL) en bandera
        add     di,159
        stosb
        retn
rd_210: mov     dh,dl           ;en CL obtengo la posicion
        add     dh,4            ;del bit a alterarse
        mov     cl,dh           ;empezando por el fin.
        cmp     dl,3
        jb      rd_2101
        inc     cl
        cmp     dl,5
        jb      rd_2101
        mov     cl,dl
        sub     cl,3
        add     cl,dh
rd_2101:mov     ax,8000h        ;un uno y quince ceros
        clc                     ;CF=0
        ror     ax,cl           ;roto CL veces
        cmp     dbin+2,'0'      ;altero Flags
        jne     rd_212
        xor     ax,0ffffh       ;saco el compl. a 1
        and     flags,ax
        retn
rd_212: or      flags,ax
        retn

rd_3:   ;Recibir una direccion y desensamblar desde ella.
```

```
        ;PGUP y PGDN para cambiar de paginas.
        ;Flechas --> y <-- para pasar del un campo al otro sin alterarlo.
        mov     dl,1            ;campo para el offset
        mov     dh,0            ;pagina 0
        mov     ax,offcod
        mov     tabcod,ax
        mov     ax,proxins
        mov     tabcod+2,ax
        mov     gdreset,1
        mov     si,offset i0cod ;linea 0
        call    siin0
rd_3a:  limpln  1,revers        ;linea 1
        cmp     dh,10
        jae     rd_3a1
        print   9,1,11cod,revers
        jmp     rd_3a2
rd_3a1: print   13,1,11cod1,revers
rd_3a2: mov     bx,offset vcc   ;ventana para dar la direccion
        mov     ax,ds
        mov     si,offset uline
        vnt     'a'
rd_31:  call    rd_30           ;resalta campo actual, muestra # pagina
        getdata 1ch,13h,dhex1,'h'
        mov     al,atrprev
        mov     di,dirprev
        mov     gdreset,0
        call    rd_301          ;quito resaltado
        cmp     gccod,0
        jne     rd_311
        mov     gdreset,1       ;dato valido
        mov     dh,0
        push    dx
        call    rd_310          ;altera variables y desensamble
        pop     dx
        jmp     rd_3a
rd_311: cmp     gccod,3
        jb      rd_31
rd_32:  cmp     gccod,3
        jne     rd_33
        xor     dl,1            ;otro campo
        jmp     rd_31
rd_33:  cmp     gccod,4
        jne     rd_34
        xor     dl,1            ;otro campo
        jmp     rd_31
rd_34:  mov     gdreset,1
        cmp     gccod,9
        jne     rd_35
        mov     ax,proxins      ;PGDN
        mov     offcod,ax
        push    dx
        call    gcod
```

```
        pop     dx
        cmp     dh,9            ;A la pagina 9 se regresa desde
        je      rd_34!          ;todas las paginas posteriores.
        inc     dx              ;no dejo que DH pase de 10.
rd_34!: mov     al,2
        mul     dh
        mov     si,offset tabcod
        add     si,ax           ;SI acuenta a la DH-esima pos. en TABCOD
        mov     ax,proxins
        mov     [si+2],ax
        jmp     rd_3a
rd_35:  cmp     gcccod,8
        jne     rd_36
        cmp     dh,0            ;PGUP. No bajo de la pagina 0.
        jne     rd_351
        jmp     rd_3a
rd_351: dec     dh              ;pongo pagina de arriba
        mov     al,2
        mul     dh
        mov     si,offset tabcod
        add     si,ax
        mov     ax,[si]
        mov     offcod,ax
        push    dx
        call    mcod
        pop     dx
        jmp     rd_3a
rd_36:  mov     ax,ds           ;ventana Opcmem2
        mov     bx,offset vopm2
        mov     si,offset memo2
        sub     si,8
        mvi     'm'
        call    mstk            ;ventana Stack
        mov     ax,regcs
        mov     segcod,ax
        mov     ax,regip
        mov     offcod,ax
        call    mcod            ;ventana Codigo
        cmp     gcccod,5
        jne     rd_37
        jmp     rd_4            ;TAB
rd_37:  cmp     gcccod,6
        jne     rd_38
        jmp     rd_2            ;SHIFT TAB
rd_38:  jmp     rd_f            ;ESC

rd_30:  mov     al,dh           ;resalto area de segmento o de offset
        add     al,'0'          ;primero muestro # de pagina
        cmp     dh,10
        jb      rd_302
        mov     al,'X'          ;pagina 10 o superior
rd_302: mov     di,46
```

```
        stosb
        mov     al,5
        mul     di
        mov     cl,al
        mov     ch,3
        posdir  cl,ch
        mov     di,oddir
        inc     di
        mov     al,70h          ;video reverso
        mov     ah,es:[di]      ;atributo previo
        mov     atrprev,ah
        mov     dirprev,di
rd_301: stosb                   ;Atributo AL en segmento u offset
        inc     di
        stosb
        inc     di
        stosb
        inc     di
        stosb
        retn
rd_310: mov     ax,word ptr dhex1+6     ;desensamblo desde dir. dada
        cmp     dl,0            ;Antes veo si lo alterado
        jne     rd_3101         ;fue el segmento o el offset
        mov     segcod,ax
        jmp     rd_312
rd_3101:mov     offcod,ax
        mov     tabcod,ax
rd_312: call    mcod
        mov     ax,proxins
        mov     tabcod+2,ax
        retn

rd_4:   ;Recibe una direccion y muestra la memoria desde ella.
        ;PGUP y PGDN para cambiar de paginas.
        ;Flechas --> y <-- para pasar del un campo al otro sin alterarlo.
        mov     dl,1            ;campo para el offset
        mov     gdreset,1
        mov     si,offset l0op1 ;linea 0
        call    mlin0
        liapln  1,revers        ;linea 1
        print   9,1,llcod,revers
rd_4a:  mov     bx,offset vc1   ;ventana para dar la direccion
        mov     ax,ds
        mov     si,offset uline
        mvt     'a'
rd_41:  call    rd_40           ;resalta campo actual
        getdata 17h,0dh,dhex1,'h'
        mov     al,atrprev
        mov     di,dirprev
        mov     gdreset,0
        call    rd_401          ;quito resaltado
        cmp     gccod,0
```

```
        mov     dirprev,di
rd_401: stosb                   ;Atributo AL en segmento u offset
        inc     di
        stosb
        inc     di
        stosb
        inc     di
        stosb
        retn
rd_410: mov     ax,word ptr dhex1+6     ;altero DIR1 y muestro memoria
        cmp     di,0
        jne     rd_4102
        mov     dir1+2,ax       ;cambio segmento
        jmp     rd_4101
rd_4102:mov     dir1,ax         ;cambio offset
rd_4101:call    prepopc         ;prepara buffer de opciones
        call    mem             ;prepara y muestra memoria
        retn

rd_5:   ;Cambios en las opciones de ventanas de memoria relativa.
        ;Flechas --> y <-- para pasar a otra de las opciones; arriba
        ;y abajo para ir a otra linea de la ventana; y PGUP y PGDN
        ;para pasar de Segmento a Offset o a la inversa.
        mov     di,0            ;linea actual
        limpln  1,revers        ;linea1
        print   2,1,llop2,revers
rd51a:  mov     dh,0            ;opcion actual
        mov     si,offset l0op21        ;linea 0
        call    mlin0
        mov     bx,offset vc2   ;recepcion de segmento
        vnt     'a'
        mov     bx,offset vc2s
        vnt     'a'
rd511:  call    rd510           ;resalta linea actual
        call    rd5100          ;resalta opcion actual de segmento
        getchar
        mov     al,atrprev      ;quito resaltados de linea y opcion
        mov     di,dirprev
        call    rd5101
        mov     al,atrprev1
        mov     di,dirprev1
        call    rd51001
        cmp     gccod,0
        je      rd511
        cmp     gccod,1         ;ENTER?
        je      rd5111a
        jmp     rd512
rd5111a:cmp     dh,11           ;opcion XXXX?
        jne     rd5112
        limpln  1,revers
        print   30,1,llop2s,revers
        mov     bx,offset vc2h
```

```
        mov     ax,ds
        mov     es,offset ulins
        mov     'e
        mov     goreset,1
rd5111: call    rd51%           ;resalto linea (y opcion)
:       call    rd51w0
        getdata lan,0eh,dtext,'h  ;recibo XXXX
        mov     al,atrprev      ;quito resaltados de linea (y opcion)
        mov     di,dirprev
        call    rd5101
;       mov     al,atrprev1
;       mov     di,dirprev1
;       call    rd51001
        mov     goreset,0
        cmp     gccod,0         ;dato valido?
        jne     rd5113
rd5112: call    rd5110          ;si, altera variables y actualiza
        print   2,1,,lop2,revers
        jmp     rd52a           ;pantalla. Paso a recibir offset
rd5113: cmp     gccod,5         ;no acepto opciones invalidas
        jb      rd5111          ;solo acepto TAB, SHIFT TAB y ESC
        cmp     gccod,7
        ja      rd5111
        sub     gccod,3         ;cambio GCCOD para coincidir con los
        jmp     rd517           ;de GETCHAR y usar la misma rutina
rd512:  cmp     gccod,5
        jne     rd517
        cmp     gcascii,77      ;-->?
        jne     rd513
        inc     dh              ;proxima opcion
        cmp     dh,12
        jne     rd5121
        mov     dh,0
rd5121: jmp     rd511
rd513:  cmp     gcascii,75      ;<--?
        jne     rd514
        dec     dh              ;opcion anterior
        cmp     dh,0ffh
        jne     rd5131
        mov     dh,11
rd5131: jmp     rd511
rd514:  cmp     gcascii,72      ;arriba?
        jne     rd515
        dec     dl
        cmp     dl,0ffh
        jne     rd5141
        mov     dl,4
rd5141: jmp     rd511
rd515:  cmp     gcascii,80      ;abajo?
        jne     rd516
        inc     dl
        cmp     dl,5
```

```
        jne     rd5151
        mov     dl,0
rd5151: jmp     rd511
rd516:  cmp     gcascii,81      ;PGDN o PGUP, pasa a recibir offset
        je      rd52a
        cmp     gcascii,73
        je      rd52a
        jmp     rd511           ;no es tecla valida
rd517:  mov     al,atrprev      ;quito resaltados de linea y opcion
        mov     di,dirprev
        call    rd5101
        mov     al,atrprev1
        mov     di,dirprev1
        call    rd51001
        call    mcod
        call    mdem
        call    mexpc
        call    esth
        cmp     gccod,2         ;TAB?
        jne     rd518
        jmp     rd_6
rd518:  cmp     gccod,3         ;SHIFT TAB?
        jne     rd519
        jmp     rd_4
rd519:  jmp     rd_f            ;ESC
rd52a:  mov     dh,0            ;opcion actual
        mov     si,offset lOopz2        ;linea 0
        call    slin0
        mov     bx,offset vc2   ;recepcion de offset
        int     'a'
        mov     bx,offset vc2o
        int     'a'
        mov     bl,dl           ;si la opcion de segmento fue XXXX solo
        mov     bh,0            ;se permite la opcion XXXX para el offset
        cmp     opcseg[bx],12
        jne     rd521
        mov     dh,16-
        jmp     rd52a1
rd521:  call    rd510           ;resalta linea actual
        call    rd5200          ;resalta opcion actual de segmento
        getchar
        mov     al,atrprev      ;quito resaltados de linea y opcion
        mov     di,dirprev
        call    rd5101
        mov     al,atrprev1
        mov     di,dirprev1
        call    rd52001
        cmp     gccod,0
        je      rd521
        cmp     gccod,1         ;ENTER?
        jne     rd522
rd52a1: call    rd530           ;altera variables, recibe despl., actualiza
```

```
        cmp     al,0            ;datos validos en subrutina?
        jne     rd5211
        jmp     rd51a
rd5211: cmp     al,1            ;tecla de salida en GETCHAR
        jne     rd5212
        jmp     rd517
rd5212: sub     gcscod,1        ;salida en GETDATA, resto 3 para ove
        jnz     rd517           ;ESCOD coincida con el de GETCHAR.
rd522:  cmp     gcscod,0
        je      rd5222          ;salida
        jmp     rd517
rd5222: cmp     gcascii,77      ;-->?
        jne     rd523
        inc     dh              ;proxima opcion
        cmp     dh,17
        jne     rd5223
        mov     dh,0
rd5223: jmp     rd521
rd523:  cmp     gcascii,75      ;<--?
        jne     rd524
        dec     dh              ;opcion anterior
        cmp     dh,0ffh
        jne     rd5231
        mov     dh,16
rd5231: jmp     rd521
rd524:  cmp     gcascii,72      ;arriba?
        jne     rd525
        dec     dl
        cmp     dl,0ffh
        jne     rd5241
        mov     dl,6
rd5241: jmp     rd521
rd525:  cmp     gcascii,80      ;abajo?
        jne     rd526
        inc     dl
        cmp     dl,5
        jne     rd5251
        mov     dl,0
rd5251: jmp     rd521
rd526:  cmp     gcascii,81      ;PGDN o PGUP? pasa a recibir segmento
        je      rd5261
        cmp     gcascii,73
        je      rd5261
        jmp     rd521           ;no es tecla valida
rd5261: call    mstk
        call    mexpc
        jmp     rd51a

rd510:  mov     cl,1fh          ;Resalto linea actual
        mov     ch,11h
        add     ch,dl
        posdir  cl,ch
```

```
        mov     di,pddir
        inc     di
        mov     ah,es:[di]      ;atributo previo
        mov     atrprev,ah
        mov     dirprev,di
        mov     al,70h
rd5101: mov     cx,15
rd5102: stosb
        inc di
        loop    rd5102
        retn
rd5100: mov     cl,0ch          ;Resalto opcion de segmento
        mov     ch,dh           ;primero busco la posicion en pantalla
        mov     bl,0ffh         ;en CL,CH
rd51002:inc     bl
        sub     ch,4
        jnb     rd51002
        add     ch,4
        mov     al,5
        mul     bl
        add     cl,al
        add     ch,0ch
        posdir  cl,ch
        mov     di,pddir
        inc     di
        mov     ah,es:[di]      ;atrib. previo
        mov     atrprev1,ah
        mov     dirprev1,di
        mov     al,70h
rd51001:mov     cx,4
rd51003:stosb
        inc     di
        loop    rd51003
        retn
rd5110: push    dx              ;altero variables y actualizo pantalla
        mov     bl,dl
        inc     dh
        mov     bh,0
        mov     opcseg[bx],dh
        cmp     dh,12           ;es XXXX?
        jne     rd51101
        mov     opcoff[bx],17
        add     bx,bx           ;si, guardo en arreglo XXXXSEG
        mov     si,word ptr dhex1+6
        mov     xxxxseg[bx],si
rd51101:call    preopc          ;actualizo pantalla
        call    mmem
        call    mcod
        pop     dx
        retn
rd5200: mov     cl,0ch          ;Resalto opcion de offset
        mov     ch,0ch
```

```
        mov     al,dh
        clc
        shr     al,1
        add     ch,al
        mposdir cl,ch
        mov     di,posdir
        inc     di
        mov     ah,es:[di]
        mov     atrprev1,ah
        mov     dirprev1,di
        mov     al,70h
rd52001:mov     cx,5
        test    dh,1
        jz      rd52002
        mov     cx,14
rd52002:stosb
        inc     di
        loop    rd52002
        retn
rd5300: push    dx              ;recibe XXXX, altera registros y pantalla
        call    rd5310          ;resalto linea y opcion
        call    rd5200
        cmp     ch,12           ;opcion XXXX?
        je      rd5301          ;En AL sale : 0 -->datos validos
        jmp     rd5304          ;1, salida por orden a GETCHAR
rd5301: limpln  1,revers        ;2, salida por orden a GETDATA.
        print   31,1,llop2o,revers
        mov     bx,offset vc2h
        mov     ax,ds
        mov     si,offset uline
        mov     'a'
        mov     gdreset,1
rd5302: getdata 1ah,0eh,dhext,'h' ;recibo XXXX
        mov     gdreset,0
        cmp     gccod,0         ;dato valido?
        jne     rd5303
        call    rd5300          ;si, altera variables y actualizo pantalla.
        mov     al,0            ;dato valido en subrutina.
        jmp     rd5304
rd5303: cmp     gccod,5         ;no acepto opciones invalidas
        jb      rd5302          ;solo acepto TAB, SHIFT TAB y ESC
        cmp     gccod,7
        ja      rd5302
        mov     al,2            ;salida en GETDATA, AL=2
        jmp     rd5304
rd5304: test    dh,1            ;pares necesitan despl.
        jnz     rd5305
        call    rd5300          ;no es par, altero y salgo con AL=0
        mov     al,0
        jmp     rd5304
rd5305: mov     bx,offset vc2d
        mov     ax,ds
```

```
        mov     si,offset uline
        mov     'a'
rd53051:getchar                 ;leo el signo
        cmp     gccod,1         ;ENTER?
        je      rd53052
        cmp     gccod,0         ;ASCII?
        je      rd53053
        cmp     gccod,4
        je      rd53051         ;dato no valido
        mov     al,1            ;salida en GETCHAR, AL=1
        jmp     rd5304
rd53052:mov     gcascii,'+'     ;ENTER = +
rd53053:cmp     gcascii,'+'     ;solo acepto signos + y -
        je      rd5306
        cmp     gcascii,'-'
        je      rd5306
        jmp     rd5305          ;dato invalido
rd5306: posdir  19h,0eh         ;eco del signo en pantalla
        mov     di,pddir
        mov     al,gcascii
        stosb
        mov     gdreset,1
rd5307: getdata 1bh,0eh,3hex1,'b' ;recibo desplazamiento
        mov     gdreset,0
        cmp     gccod,0         ;dato valido?
        jne     rd53071
        call    rd5300          ;si, altero variables y pantalla
        mov     al,0            ;Correcto, AL=0
        jmp     rd5304
rd53071:cmp     gccod,4         ;flecha a la izquierda?
        jne     rd5308
        jmp     rd53051         ;si, regreso a recibir signo
rd5308: cmp     gccod,5         ;no acepto teclas invalidas
        jb      rd5307
        cmp     gccod,7
        je      rd5307
        mov     al,2            ;salida en GETDATA
        jmp     rd5304
rd5304: push    ax
        mov     al,atrprev      ;quito resaltados de linea y opcion
        mov     di,dirprev
        call    rd5101
        mov     al,atrprev1
        mov     di,dirprev1
        call    rd52001
        print   2,1,llop2,revers
        pop     ax
        pop     dx
        retn
rd5300: push    dx              ;altero variables, actualizo pantalla
        inc     dh
        mov     bl,dl
```

```
        mov     bh,0
        mov     opcoff[bx],dh
        cmp     dh,17           ;offset fue XXXX?
        je      rd53001
        test    ch,1            ;necesita desplazamiento?
        jnz     rd53002
        mov     al,gcascii      ;si,guardo el signo
        mov     signoff[bx],al
rd53001:add     bx,bx           ;guardo XXXX
        mov     si,word ptr dhex1+6
        mov     xxxxoff[bx],si
rd53002:call    mcoc
        call    mstk
        call    mexpc
        call    prepc
        call    mmem
        pop     dx
        retn

rd_6:   ;Cambios en memoria directamente desde teclado. Las flechas -->
        ;y <-- no hacen cambio de linea. Arriba y Abajo no cambian de
        ;pagina, son para pasar de linea o ventana. PGUP y PGDN para pasar
        ;a otra pagina (+-64 en la ventana absoluta, +-8 en las relativas)
        mov     rd6?,0          ;bandera para rd600 y rd700
        mov     dl,0            ;DL=# de linea (0-12)
        mov     dh,0            ;DH=# de columna (0-7)
        mov     si,offset dir2i ;primero salvo las direcciones de
        mov     di,offset dir2ir ;las ventanas relativas
        push    es
        push    ds
        pop     es
        mov     cx,10
        rep     movsw
        pop     es
        mov     si,offset l0hex ;linea 0
        call    mlin0
        liepln  1,revers        ;linea 1
        print   6,1,l1hex,revers
        mov     gdreset,1
rd6a:   mov     bx,offset vdh   ;dibujo la ventana
        mov     ax,ds
        mov     si,offset uline
        vnt     'a'
rd61:   call    rd60            ;resalto byte actual
        call    rd600           ;resalto si v. relativas en pagina 0
        getdata 20h,6,dbyte,'h' ;recibo entrada de teclado
        mov     al,atrprev      ;quito resaltados
        mov     di,dirprev
        mov     gdreset,0
        call    rd601
        cmp     gccod,0         ;dato valido?
        jne     rd611
```

```
        mov     gdreset,1       ;si
        call    rdo10           ;altero memoria, actualizo pantalla
        inc     dh              ;paso al proximo byte
        cmp     dh,8            ;hasta llegar al fin de la ventana
        jne     rd6a
        mov     dh,0
        inc     dl
        cmp     dl,8
        jb      rd6a
        dec     dl
        mov     dh,7
        jmp     rd6a
rd611:  cmp     gccod,1         ;arriba?
        jne     rd612
        dec     dl              ;si
        cmp     dl,0ffh
        jne     rd6111
        mov     dl,12
rd6111: jmp     rd61
rd612:  cmp     gccod,2         ;abajo?
        jne     rd613
        inc     dl              ;si
        cmp     dl,13
        jne     rd6121
        mov     dl,0
rd6121: jmp     rd61
rd613:  cmp     gccod,3         ;-->?
        jne     rd614
        inc     dh              ;si
        cmp     dh,8
        jne     rd6131
        mov     dh,0
rd6131: jmp     rd61
rd614:  cmp     gccod,4         ;<--?
        jne     rd615
        dec     dh              ;si
        cmp     dh,0ffh
        jne     rd6141
        mov     dh,7
rd6141: jmp     rd61
rd615:  cmp     gccod,8         ;PGUP?
        je      rd616
        cmp     gccod,9         ;o PGDN?
        jne     rd617
rd616:  call    rd6160          ;si
        jmp     rd61
rd617:  mov     di,offset dir21 ;salida de esta rutina, restauro
        mov     si,offset dir21r ;direcciones de ventanas
        mov     cx,10
        push    es
        push    ds
        pop     es
```

```
        rep     movsw
        pop     es
        call    mcod            ;restauro pantalla
        call    mstr
        call    mexoc
        call    mmep
        mov     dx,offset vere
        mvt     'e'
        call    mpsw
        cmp     gccoc,5         ;TAB?
        jne     rd62
        jmp     rc_7            ;si
rd62:   cmp     gccod,6         ;SHIFT TAB?
        jne     rd63
        jmp     rd_5            ;si
rd63:   jmp     rd_f            ;ESC
rd60:   mov     cl,dh           ;resalto byte actual. Primero
        mov     ch,dl           ;calculo posicion
        mov     al,3
        mul     cl
        add     al,2fh
        mov     cl,al
        add     ch,0
        oosdir  cl,ch
        mov     di,oddir
        inc     di
        mov     ah,es:[di]
        mov     atrprev,ah
        mov     dirprev,di
        mov     al,70h
rd601:  stosb                   ;cambio el atributo
        inc     di
        stosb
        retn
rd600:  mov     nlinea,0        ;para cada linea...
rd6001: mov     si,offset dir21 ;veo si coincide DIR2x con DIR2xR
        mov     di,offset dir21r
        mov     al,nlinea
        xor     ah,ah
        add     ax,ax
        add     ax,ax
        add     si,ax
        add     di,ax
        push    es
        push    ds
        pop     es
        mov     al,2bh          ;atributo que pondre si coincide
        cmpsw
        pop     es
        je      rd6002
        mov     al,atrprev1     ;atributo previo
rd6002: mov     cl,1fh          ;posicion a resaltarse o normalizarse
```

```
        mov     ch,nlinea
        add     ch,17
        posdir  cl,ch
        mov     ch,posdir
        inc     di
        cmp     rd67,0          ;leo atrprevi solo al primer caso aqui
        jne     rd6003
        mov     ah,es:[di]      ;atributo previo
        mov     atrprevi,ah
        mov     rd67,1          ;bandera indica que ya paso por aqui
rd6003: mov     cx,15
rd6004: stosb
        inc     di
        loop    rd6004
        inc     nlinea          ;fin del analisis de la linea
        cmp     nlinea,5        ;termino?
        je      rd6005
        jmp     rd6001          ;no
rd6005: retn                    ;si.
rd6010: push    dx              ;altera memoria
        cmp     dl,8            ;estoy en ventana absoluta?
        jb      rd6101
        jmp     rd6102
rd6101: mov     ch,dir1         ;si
        mov     es,dir1+2
        mov     al,8
        mul     dl
        add     al,dh
        mov     ah,0
        add     di,ax           ;ES:DI apunta a byte a cambiarse
        mov     al,dbyte+4
        stosb
        mov     es,videoseg
        push    dx
        call    mmemx           ;actualizo ventana de memoria
        pop     dx
        jmp     rd6104
rd6102: mov     cl,dl           ;ventanas relativas
        sub     cl,8
        mov     ch,0
        add     cx,cx
        add     cx,cx
        mov     si,offset dir21
        add     si,cx
        mov     di,[si]
        mov     al,dh
        mov     ah,0
        add     di,ax
        mov     es,[si+2]       ;ES:DI apunta a byte a cambiarse
        mov     al,dbyte+4
        stosb
        mov     es,videoseg
```

```
        call    mmemx           ;actualizo ventana de memoria
rd610f: call    mcod
        call    ret1
        call    mexpc
        pop     dx
        retn
rd61601: ;cambia de pagina, cambia DIR1 y DIR21...DIR25
        cmp     cl,8            ;ventana absoluta?
        jae     rd61601
        mov     si,offset dir1  ;si, SI apunta a direccion a alterarse
        mov     cx,64           ;CX tiene el valor a sumar o restar
        jmp     rd61602
rd61601:mov     si,offset dir21 ;ventana relativa
        mov     cl,dl
        sub     cl,8
        mov     ch,0
        add     cx,cx
        add     cx,cx
        add     si,cx           ;SI apunta a direccion a alterarse
        mov     cx,8            ;CX tiene el valor a sumar o restar
rd61602:cmp     gcod,8          ;PGUP?
        jne     rd61603
        sub     [si],cx         ;si
        jmp     rd61604
rd61603:add     [si],cx         ;PGDN
rd61604:push    dx
        call    preopc
        call    mmemx
        pop     dx
rd61605:retn

rd_7:   ;Cambios en memoria directamente desde teclado (ASCII). Las flechas
        ;--> y <-- no hacen cambio de linea. Arriba y Abajo no cambian de
        ;pagina, son para pasar de lines o ventana. PGUP y PGDN para pasar
        ;a otra pagina (+-64 en la ventana absoluta, +-8 en las relativas)
        mov     rd6?,0          ;bandera para rd600 y rd700
        mov     dl,0            ;DL=# de linea (0-12)
        mov     dh,0            ;DH=# de columna (0-7)
        mov     si,offset dir21 ;primero salvo las direcciones de
        mov     di,offset dir21r ;las ventanas relativas
        push    es
        push    ds
        pop     es
        mov     cx,10
        rep     movsw
        pop     es
        mov     si,offset l0asc ;linea 0
        call    mlin0
        limpln  1,revers        ;linea 1
        print   6,1,11hex,revers
rd7a:   mov     bx,offset vda   ;dibujo la ventana
        vnt     'a'
```

```
rd71:   call    rd70            ;resalto ascii actual
        call    rd700           ;resalto si v. relativas en pagina 0
        getchar                 ;recibo entrada de teclado
        mov     al,atrprev      ;quito resaltado
        mov     di,dirprev
        stosb
        cmp     gccod,1         ;ENTER?
        jne     rd71x
        mov     gccod,0         ;si, simulo GCCOD=0
        mov     gcascii,13      ;Enter=^M
rd71x:  cmp     gccod,0         ;dato valido?
        jne     rd711
        call    rd710           ;si, alt. memoria, act. pantalla, -->.
        inc     dh              ;paso al proximo byte
        cmp     dh,8            ;hasta llegar al fin de la ventana
        jne     rd71
        mov     dh,0
        inc     dl
        cmp     dl,8
        jb      rd71
        dec     dl
        mov     dh,7
        jmp     rd71
rd711:  cmp     gccod,5         ;especiales?
        je      rd711x
        jmp     rd717
rd711x: cmp     gcascii,72      ;arriba?
        jne     rd712
        dec     dl              ;si
        cmp     dl,0ffh
        jne     rd7111
        mov     dl,12
rd7111: jmp     rd71
rd712:  cmp     gcascii,80      ;abajo?
        jne     rd713
        inc     dl              ;si
        cmp     dl,13
        jne     rd7121
        mov     dl,0
rd7121: jmp     rd71
rd713:  cmp     gcascii,77      ;-->?
        jne     rd714
        inc     dh              ;si
        cmp     dh,8
        jne     rd7131
        mov     dh,0
rd7131: jmp     rd71
rd714:  cmp     gcascii,75      ;<--?
        jne     rd715
        dec     dh              ;si
        cmp     dh,0ffh
        jne     rd7141
```

```
        mov     dh,7
rd7141: jmp     rd71
rd715:  cmp     gcascii,73      ;PGUP?
        je      rd716
        cmp     gcascii,81      ;o PGDN?
        je      rd716
        jmp     rd71            ;tecla invalida
rd716:  call    rd716a          ;PGUP O PGDN
        jmp     rd71
rd717:  mov     di,offset dir21 ;salida de esta rutina, restauro
        mov     si,offset dir21r ;direcciones de ventanas
        mov     cx,10
        push    es
        push    ds
        pop     es
        rep     movsw
        pop     es
        call    pdebug          ;restauro pantalla
        cmp     gccod,2         ;TAB?
        jne     rd72
        jmp     rd_1            ;si
rd72:   cmp     gccod,3         ;SHIFT TAB?
        jne     rd73
        jmp     rd_6            ;si
rd73:   jne     rd_f            ;ESC
rd70:   mov     cl,dh           ;resalto byte actual. Primero
        mov     ch,dl           ;calculo posicion
        add     cl,47h
        add     ch,9
        posdir  cl,ch
        mov     di,pddir
        inc     di
        mov     ah,es:[di]
        mov     atrprev,ah
        mov     dirprev,di
        mov     al,70h
        stosb                   ;cambio el atributo
        retn
rd700:  mov     nlinea,0        ;para cada linea...
rd7001: mov     si,offset dir21 ;veo si coincide DIR2x con DIR2xR
        mov     di,offset dir21r
        mov     al,nlinea
        xor     ah,ah
        add     ax,ax
        add     ax,ax
        add     si,ax
        add     di,ax
        push    es
        push    ds
        pop     es
        mov     al,2bh          ;atributo que pondre si coincide
        cmpsw
```

```
        pop     es
        je      rd7002
        mov     al,atrprev1     ;atributo previo
rd7002: mov     cl,1+h          ;posicion a resaltarse o normalizarse
        mov     ch,nlinea
        add     ch,17
        posdir  cl,ch
        mov     di,oddir
        inc     di
        cmp     rd6?,0          ;lee atrprev1 solo al primer paso aqui
        jne     rd7003
        mov     ah,es:[di]      ;atributo previo
        mov     atrprev1,ah
        mov     rd6?,1          ;bandera indica que ya paso por aqui
rd7003: mov     cx,15
rd7004: stosb
        inc     di
        loop    rd7004
        inc     nlinea          ;fin del analisis de la linea
        cmp     nlinea,5        ;termino?
        je      rd7005
        jmp     rd7001          ;no
rd7005: retn                    ;si.
rd710:  push    dx              ;altera memoria
        cmp     dl,8            ;estoy en ventana absoluta?
        jb      rd7101
        jmp     rd7102
rd7101: mov     di,dir1         ;si
        mov     es,dir1+2
        mov     al,8
        mul     dl
        add     al,dh
        mov     ah,0
        add     di,ax           ;ES:DI apunta a byte a cambiarse
        mov     al,gcascii
        stosb
        mov     es,videoseg
        push    dx
        call    memmx           ;actualizo ventana de memoria
        pop     dx
        jmp     rd710f
rd7102: mov     cl,dl           ;ventanas relativas
        sub     cl,8
        mov     ch,0
        add     cx,cx
        add     cx,cx
        mov     si,offset dir21
        add     si,cx
        mov     di,[si]
        mov     al,dh
        mov     ah,0
        add     di,ax
```

```
        mov     es,[si+2]       ;ES:DI apunta a byte a cambiarse
        mov     al,gcascii
        stosb
        mov     es,videoseg
        call    mmem            ;actualizo ventana de memoria
rd7104: call    mcod
        call    mstk
        call    mexpc
        pop     dx
        retn
rd7160: ;cambia de pagina, cambia DIR1 y DIR21...DIR25
        cmp     cl,8            ;ventana absoluta?
        jae     rd71601
        mov     si,offset dir1  ;si, SI apunta a direccion a alterarse
        mov     cx,64           ;CX tiene el valor a sumar o restar
        jmp     rd71602
rd71601:mov     si,offset dir21 ;ventana relativa
        mov     cl,dl
        sub     cl,8
        mov     ch,0
        add     cx,cx
        add     cx,cx
        add     si,cx           ;SI apunta a direccion a alterarse
        mov     cx,6            ;CX tiene el valor a sumar o restar
rd71602:cmp     gcascii,73      ;PGUP?
        jne     rd71603
        sub     [si],cx         ;si
        jmp     rd71604
rd71603:add     [si],cx         ;PGDN
rd71604:push    dx
        call    preopc
        call    mmemx
        pop     dx
rd71605:retn

rd_f:   mov     si,offset l0ini
        call    mlin0
        call    pdebug
        mov     bx,offset vlin1
        mov     ax,ds
        mov     si,offset l1ini
        int     'a'
        ret

rd43m:  ;muestro las equivalencias de las direcciones relativas
        mov     nlinea,0
        mov     si,offset dir21 ;para cada linea...
rd43m1: mov     bl,nlinea
        mov     dh,nlinea
        add     dh,17
        posdir  1fh,dh
        mov     di,pddir
```

```
        mov     cx,15           ;primero borro la linea
        mov     al,20h
rdf3m2: stosb
        inc     di
        loop    rdf3m2
        mov     di,oddir
        add     di,12
        mov     bh,0
        and     bx,bx
        add     bx,bx
        mov     bp,bx
        mov     al,ds:[si+bx+3] ;MSB del segmento
        hexasc
        mov     es:[di],bh
        mov     es:[di+2],bl
        mov     al,ds:[si+bp+2] ;lsb del segmento
        hexasc
        mov     es:[di+4],bh
        mov     es:[di+6],bl
        mov     byte ptr es:[di+8],':'
        mov     al,ds:[si+bp+1] ;MSB del offset
        hexasc
        mov     es:[di+10],bh
        mov     es:[di+12],bl
        mov     al,ds:[si+bp]
        hexasc
        mov     es:[di+14],bh
        mov     es:[di+16],bl
        inc     nlinea          ;fin de la linea, veo si hago otra
        cmp     nlinea,5
        je      rdf3m3
        jmp     rdf3m1
rdf3m3: linepln 23,revers
        linepln 24,normal
        print   1,23,teccont,revers
        getchar
rdf3mf: call    preopc
        call    mmem
        retn

        end
```

```
        dosseg
        .model small
        .data
cont    db      ?               ;Contador de segundos
decena  db      ?               ;Decenas de segundos
unidad  db      ?               ;Segundos
cr      db      0dh             ;Paso a proxima linea
finstr  db      '$'             ;Fin del string

        .code
crono:  mov      cont,0         ;preparo contador
        mov     dl,7            ;hago un BEEP
        mov     ah,2
        int     21h
        call    pant            ;muestro inicio de cuenta
lazo:   mov     al,2
lazo1:  mov     cx,0ffffh       ;retardo para tener la
lazo2:  loop    lazo2           ;cuenta en segundos
        dec     al
        jnz     lazo1
        inc     cont            ;incremento contador
        call    pant            ;muestro cuenta actual
        cmp     cont,60
        jnz     lazo            ;hacer este trabajo 60 veces
        mov     dl,7            ;salgo con BEEP
        mov     ah,2
        int     21h
        mov     ah,4ch          ;termino
fin:    int     21h

pant:   mov     cl,cont         ;rutina que muestra numero en pantalla
        mov     decena,0
        mov     unidad,0
p1:     cmp     cl,10           ;si CL < 10
        jae     p2
        mov     unidad,cl       ;guardar CL en las unidades
        jmp     mostr           ;y pasar a mostrar el numero
p2:     inc     decena          ;si CL > 10, incremento decenas
        sub     cl,10           ;y resto 10 a CL
        jmp     p1
mostr:  mov     ah,9            ;funcion 9 del DOS: mostrar
        mov     dx,offset decena  ;un string terminado en '$'
        int     21h
        retn
        end     crono
        end
```

```
        dosseg
        .model small
        .data
cont    db      ?               ;Contador de segundos
decena  db      ?               ;Decenas de segundos
unidad  db      ?               ;Segundos
cr      db      0ah             ;Paso a proxima linea
finstr  db      '$'             ;Fin del string

        .code
crono:  mov     ax,@data
        mov     ds,ax
        mov     cont,0          ;preparo contador
        mov     dl,7            ;hago un BEEP
        mov     ah,2
        int     21h
        call    pant            ;muestro inicio de cuenta
lazo:   mov     al,8
lazo1:  mov     cx,0ffffh       ;retardo para tener la
lazo2:  loop    lazo2           ;cuenta en segundos
        dec     al
        jnz     lazo1
        inc     cont            ;incremento contador
        call    pant            ;muestro cuenta actual
        cmp     cont,60
        jnz     lazo            ;hacer este trabajo 60 veces
        mov     dl,7            ;salgo con BEEP
        mov     ah,2
        int     21h
        mov     ah,4ch          ;termino
fin:    int     21h

pant:   mov     cl,cont         ;rutina que muestra numero en pantalla
        mov     decena,0
        mov     unidad,0
p1:     cmp     cl,10           ;si CL < 10
        jae     p2
        mov     unidad,cl       ;guardar CL en las unidades
        jmp     mostr           ;y pasar a mostrar el numero
p2:     inc     decena          ;si CL > 10, incremento decenas
        sub     cl,10           ;y resto 10 a CL
        jmp     p1
mostr:  add     decena,30h      ;convierto en ASCII
        add     unidad,30h
        mov     ah,9            ;funcion 9 del DOS: mostrar
        mov     dx,offset decena  ;un string terminado en '$'
        int     21h
        retn
        end     crono
        end
```

```
                dosseg
                .model small
                .data
buffer          db      80 dup (0)
pregunta        db      "Nombre del archivo a procesar:"
handle          dw      0
cuenta          db      ?
tamano          equ     128
bloque          db      tamano dup(0)
                .code
inicio:         mov     ax,@data        ;puntero ds a datos
                mov     ds,ax           ;pregunta el nombre
                mov     ah,09h          ;del nombre del archivo
                mov     dx,offset pregunta
                int     21h
                mov     ah,0ah          ;ingresa a buffer el
                mov     dx,offset buffer ;nombre del archivo
                mov     buffer,12
                int     21h
                mov     ah,3dh          ;abre un archivo con el
                mov     dx,offset buffer ;nombre que esta en buffer
                mov     al,0
                int     21h
                jc      fin             ;CF=1 es error
                mov     handle,ax       ;lee el tamano de bytes
                mov     ah,3fh          ;del archivo
                mov     dx,offset bloque
                mov     cx,tamano
                mov     bx,handle
                int     21h
                mov     ah,3eh          ;cierra el archivo
                mov     bx,handle
                int     21h
                mov     cuenta,0        ;contador de bytes leidos
disp:           mov     bx,offset bloque ;despliega los "tamano"
                mov     al,cuenta       ;bytes leidos
                mov     ah,0
                add     bx,ax
                mov     dl,ds:[bx]
                mov     ah,2
                int     21h
                inc     cuenta          ;cuenta=cuenta+1
                cmp     cuenta,tamano   ;si cuenta < tamano sigue
                jne     disp            ;desplegando
fin:            mov     ah,4ch          ;termina
                int     21h
                end     inicio
                end
```

```
                dosseg
                .model small
                .data
buffer          db      80 dup (?)
pregunta        db      "Nombre del archivo a procesar?",0dh,0ah,'$'
handle          dw      0
cuenta          db      ?
tamano          equ     128
bloque          db      tamano dup(0)
                .code
inicio:         mov     ax,@data                ;puntero ds a datos
                mov     ds,ax                   ;pregunta el nombre
                mov     ah,09h                  ;del nombre del archivo
                mov     dx,offset pregunta
                int     21h
                mov     ah,0ah                  ;ingresa a buffer el
                mov     dx,offset buffer        ;nombre del archivo
                mov     buffer,78
                int     21h
                mov     si,offset buffer        ;pongo un 0 luego del
                mov     bl,buffer+1             ;nombre para que sea
                mov     bh,0                    ;un ASCIIZ
                add     bx,2
                mov     byte ptr[si+bx],0
                mov     ah,3dh                  ;abre un archivo con el
                mov     dx,offset buffer+2      ;nombre que esta en buffer,
                mov     al,0                    ;solo para lectura
                int     21h
                jc      fin                     ;CF=1 es error
                mov     handle,ax
                mov     ah,3fh                  ;lee 'tamano' bytes
                mov     dx,offset bloque        ;del archivo
                mov     cx,tamano
                mov     bx,handle
                int     21h
                mov     ah,3eh                  ;cierra el archivo
                mov     bx,handle
                int     21h
                mov     cuenta,0                ;contador de bytes leidos
disp:           mov     bx,offset bloque        ;despliega los "tamano"
                mov     al,cuenta               ;bytes leidos
                mov     ah,0
                add     bx,ax
                mov     dl,ds:[bx]
                mov     ah,2
                int     21h
                inc     cuenta                  ;cuenta=cuenta+1
                cmp     cuenta,tamano           ;si cuenta < tamano sigue
                jne     disp                    ;desplegando
fin:            mov     ah,4ch                  ;termina
                int     21h
                end     inicio
```

DIAGRAMA DE BLOQUES DEL CONTROLADOR DE FLUJO

Modulo Flujo.Asm

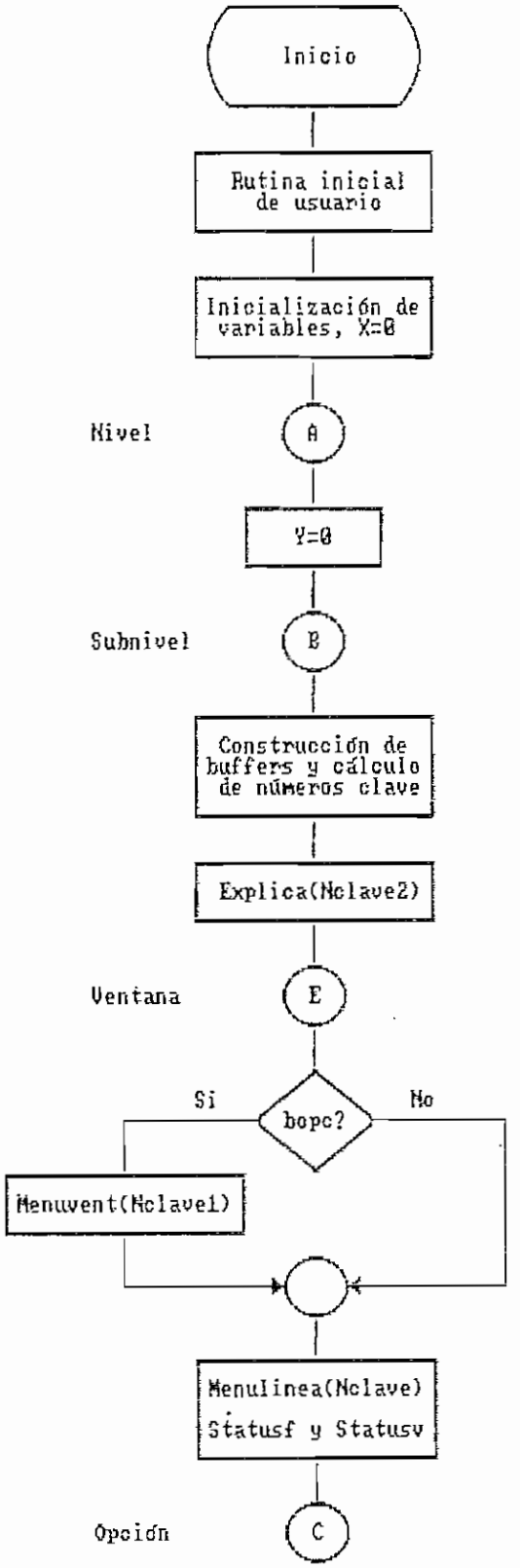
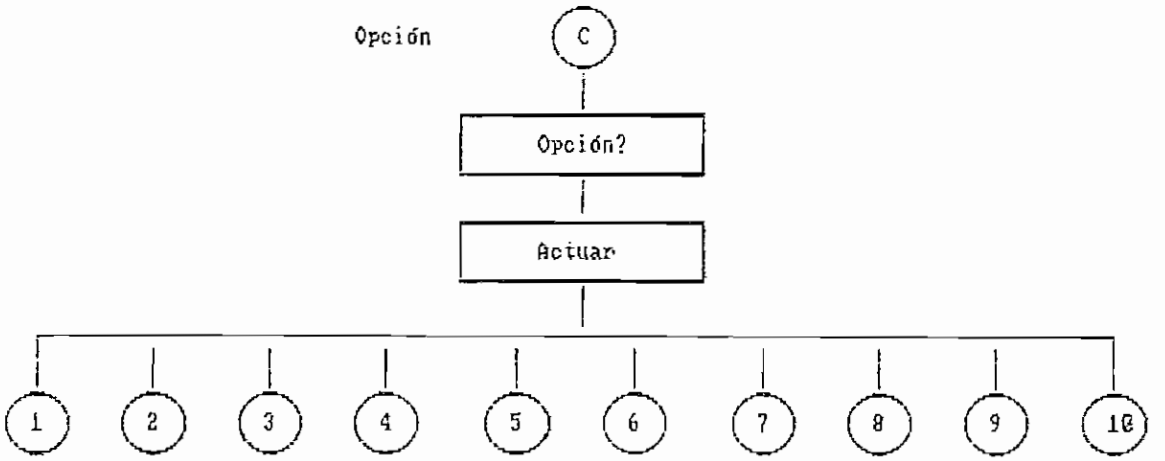
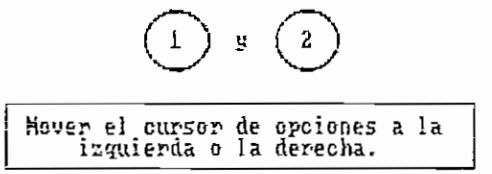
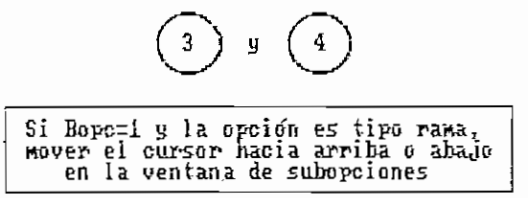


Diagrama A.1







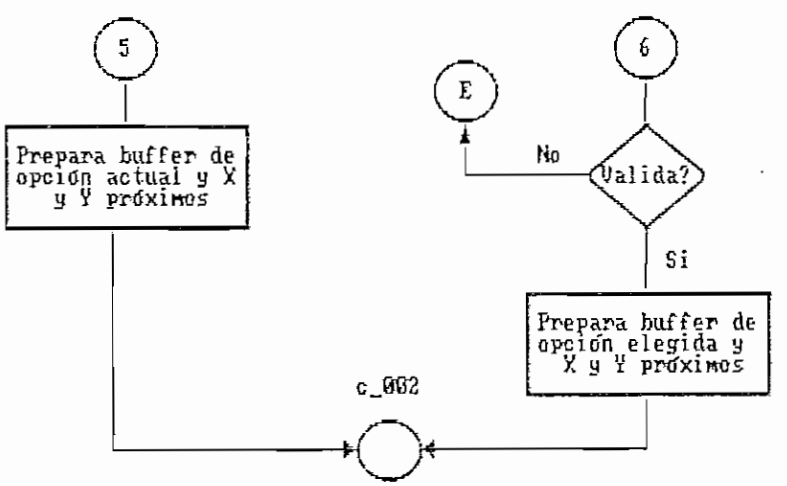


Diagrama A1 (cont...)

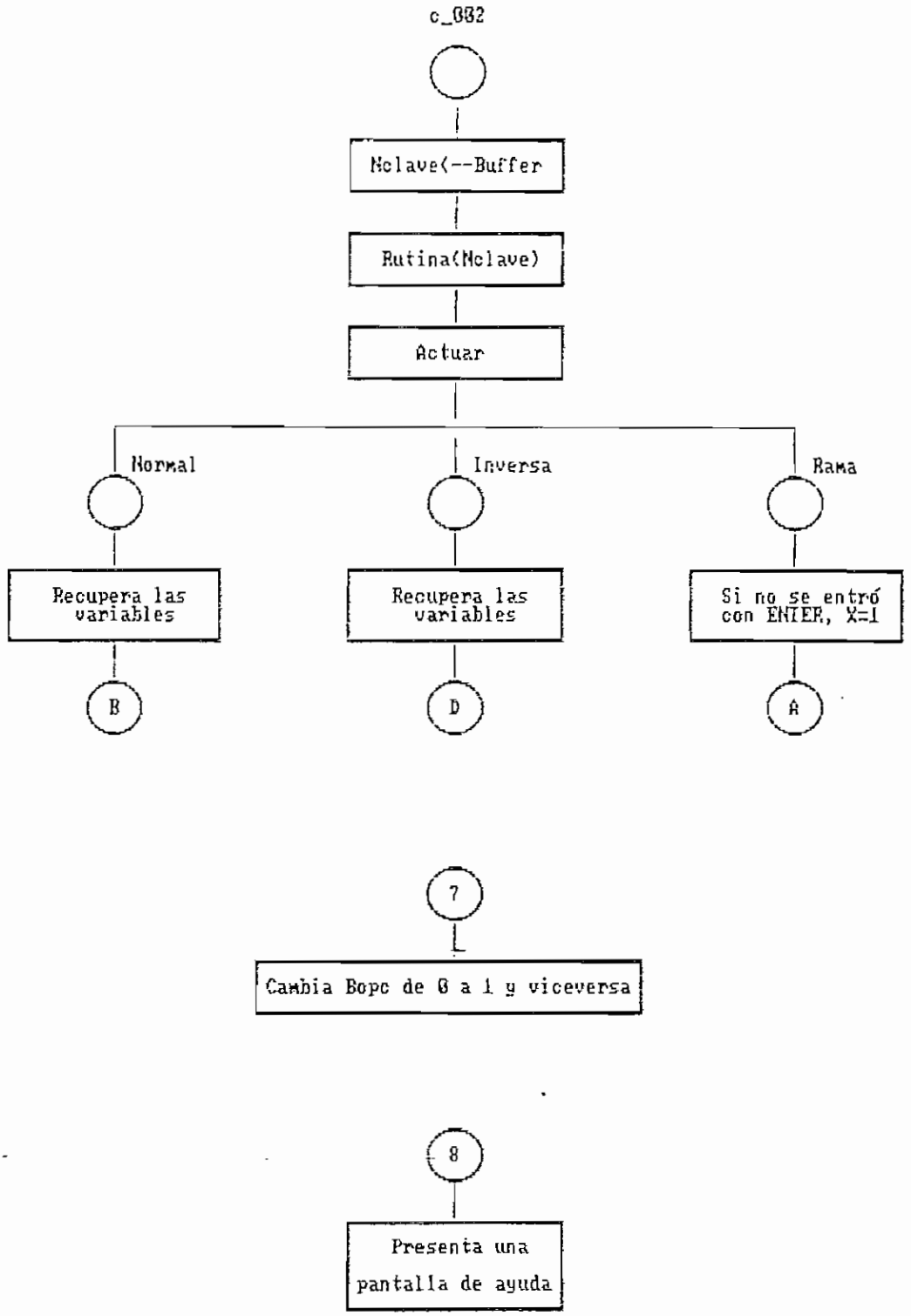


Diagrama A1 (cont...)

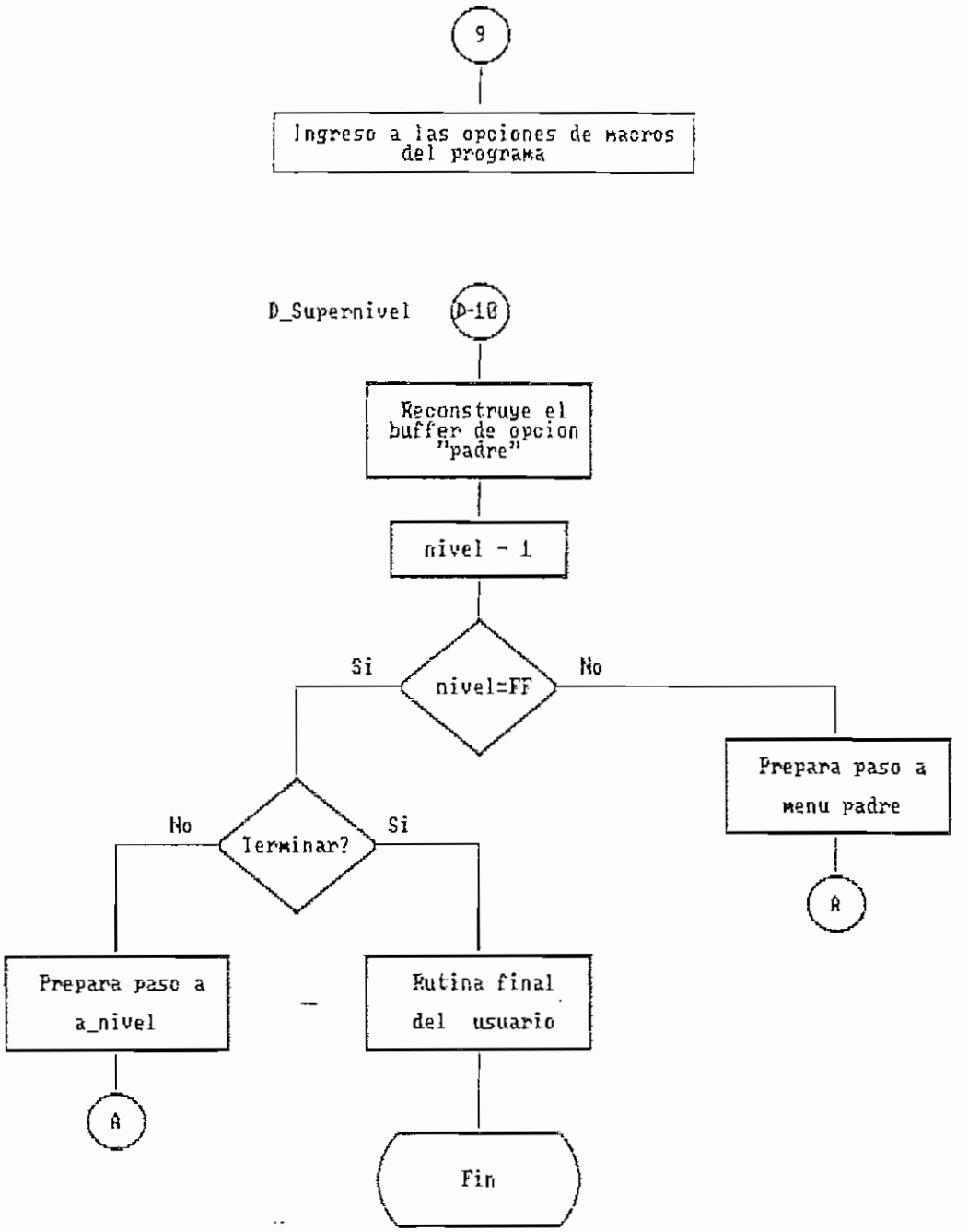
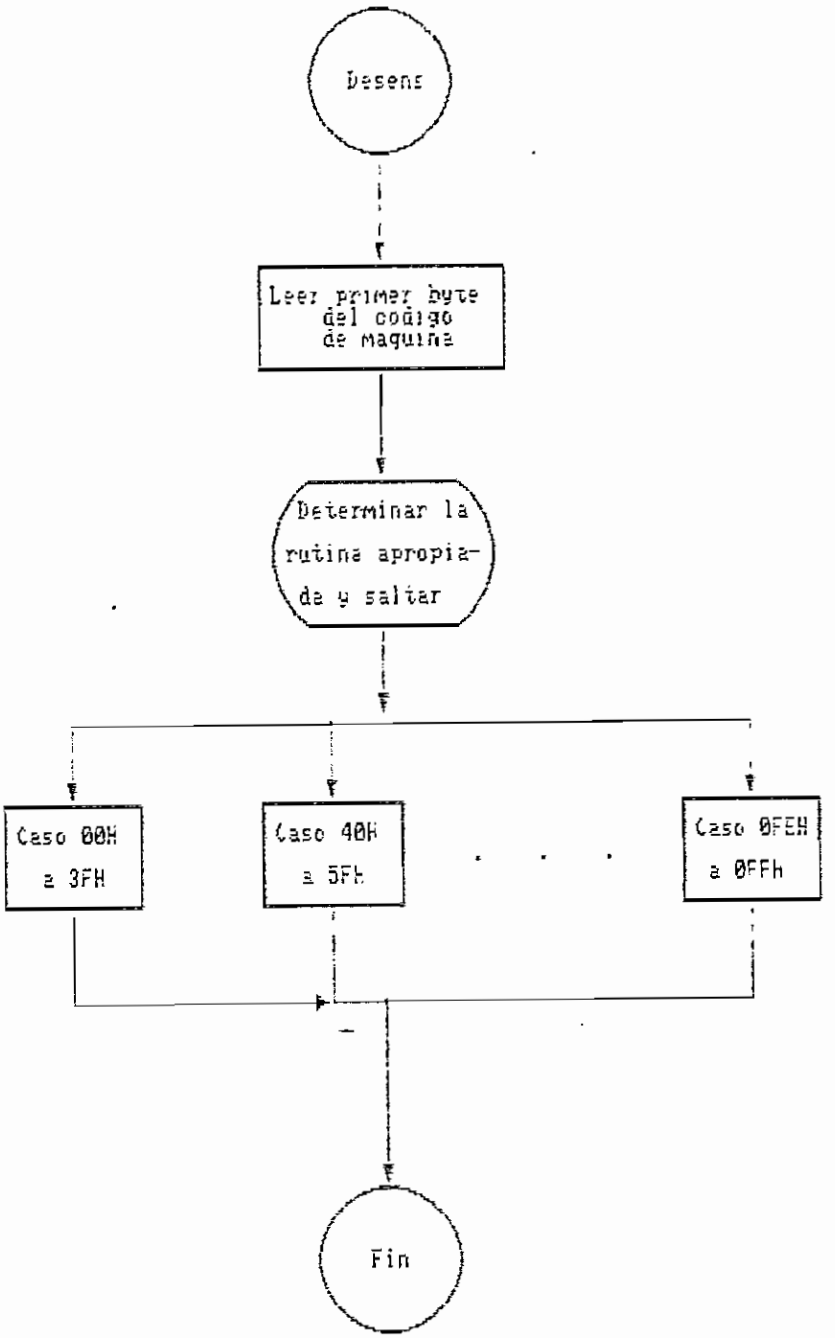


Diagrama A1 (cont...)

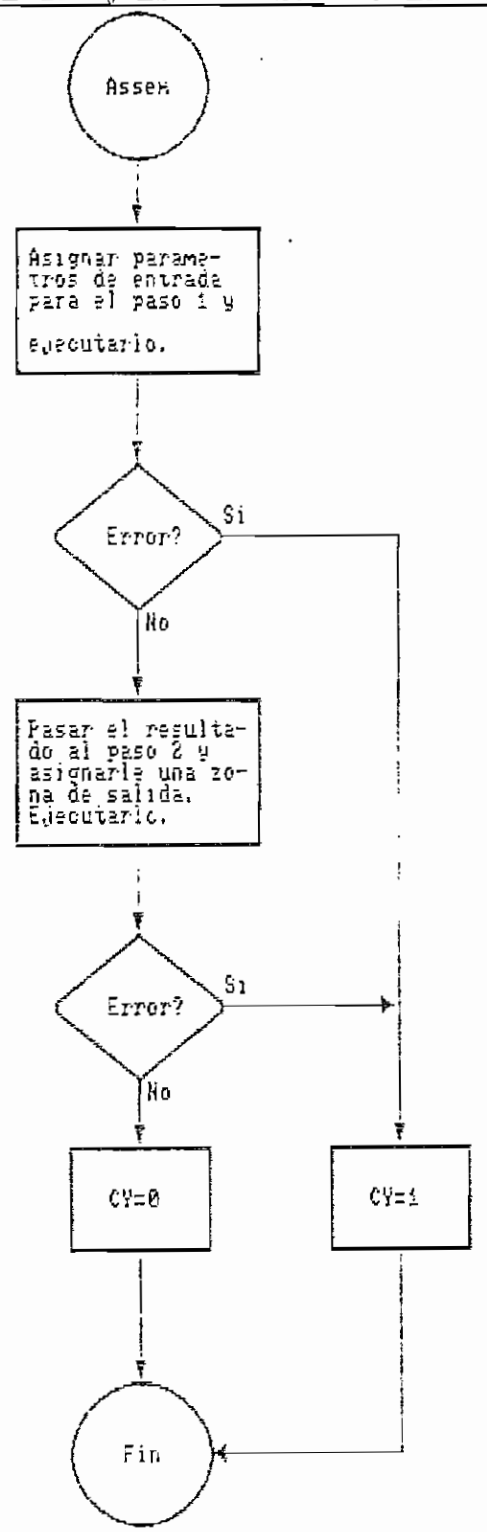
DIAGRAMA DE BLOQUES DEL MODULO DESENSAMBLADOR



Cuerpo principal del modulo Desens.Asm

Diagrama A.2

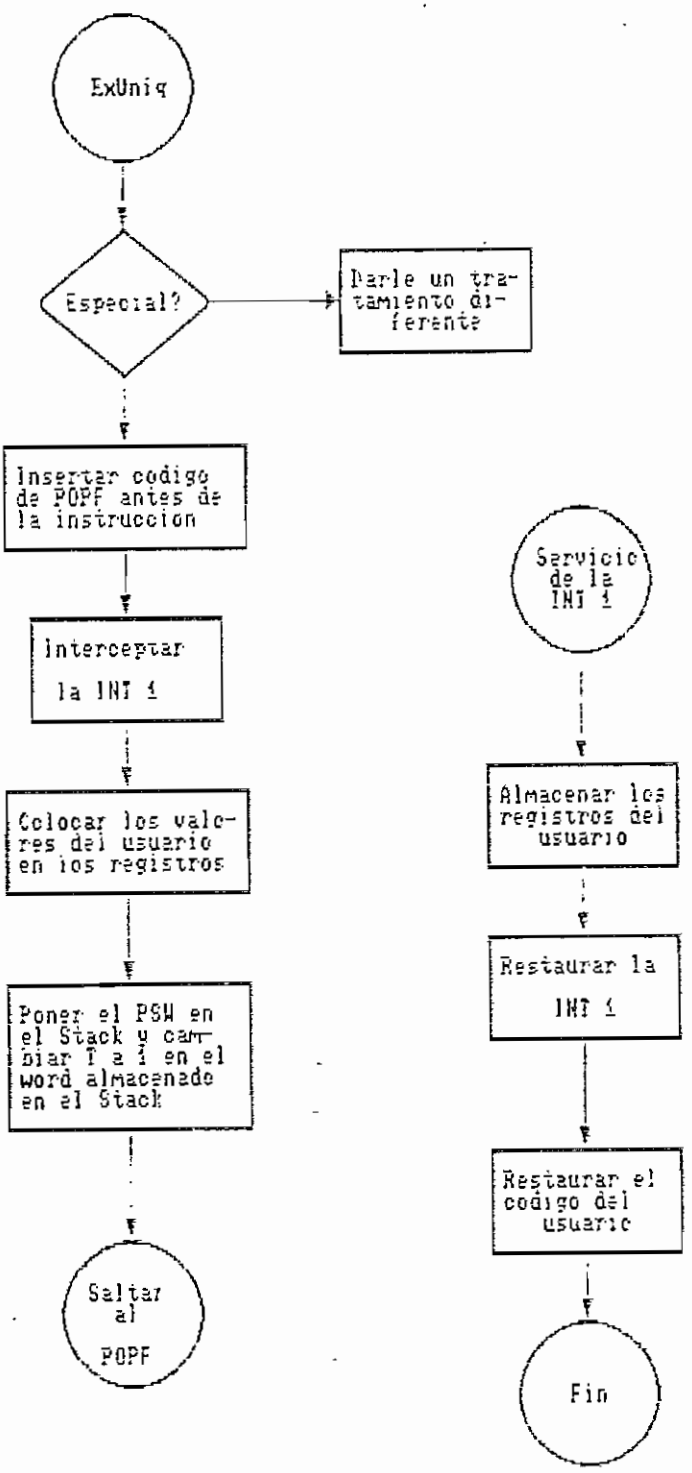
DIAGRAMA DE BLOQUES DEL MODULO ENSAMBLADOR



Cuerpo principal del modulo Assem.Asm

Diagrama A.3

DIAGRAMA DE BLOQUES DEL EJECUTOR DE UNICA INSTRUCCION



Cuerpo principal del procedimiento ExUniq

Diagrama A.4

DIAGRAMA DE BLOQUES DEL MODULO PRINCIPAL DEL PROGRAMA

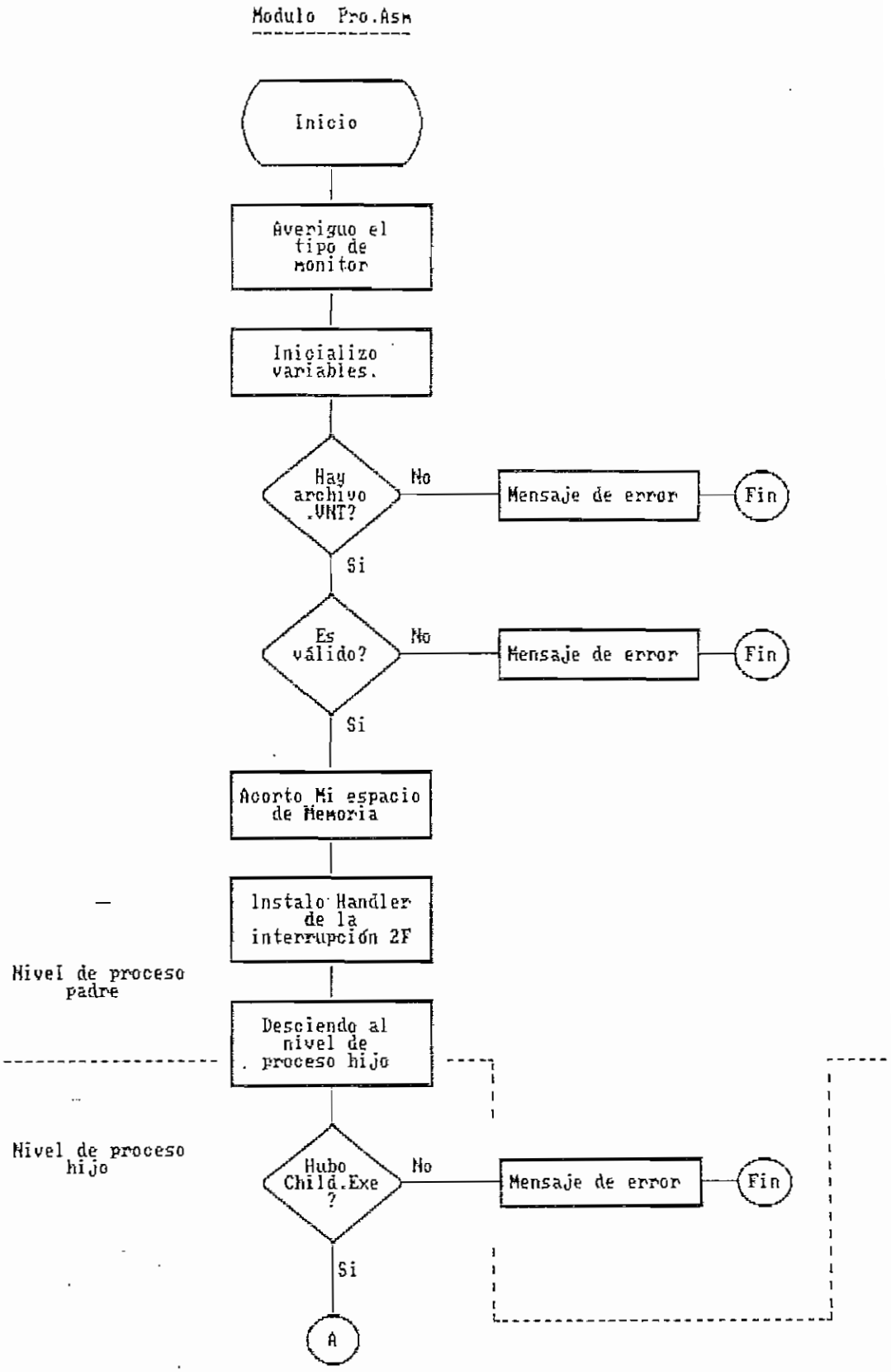


Diagrama A.5

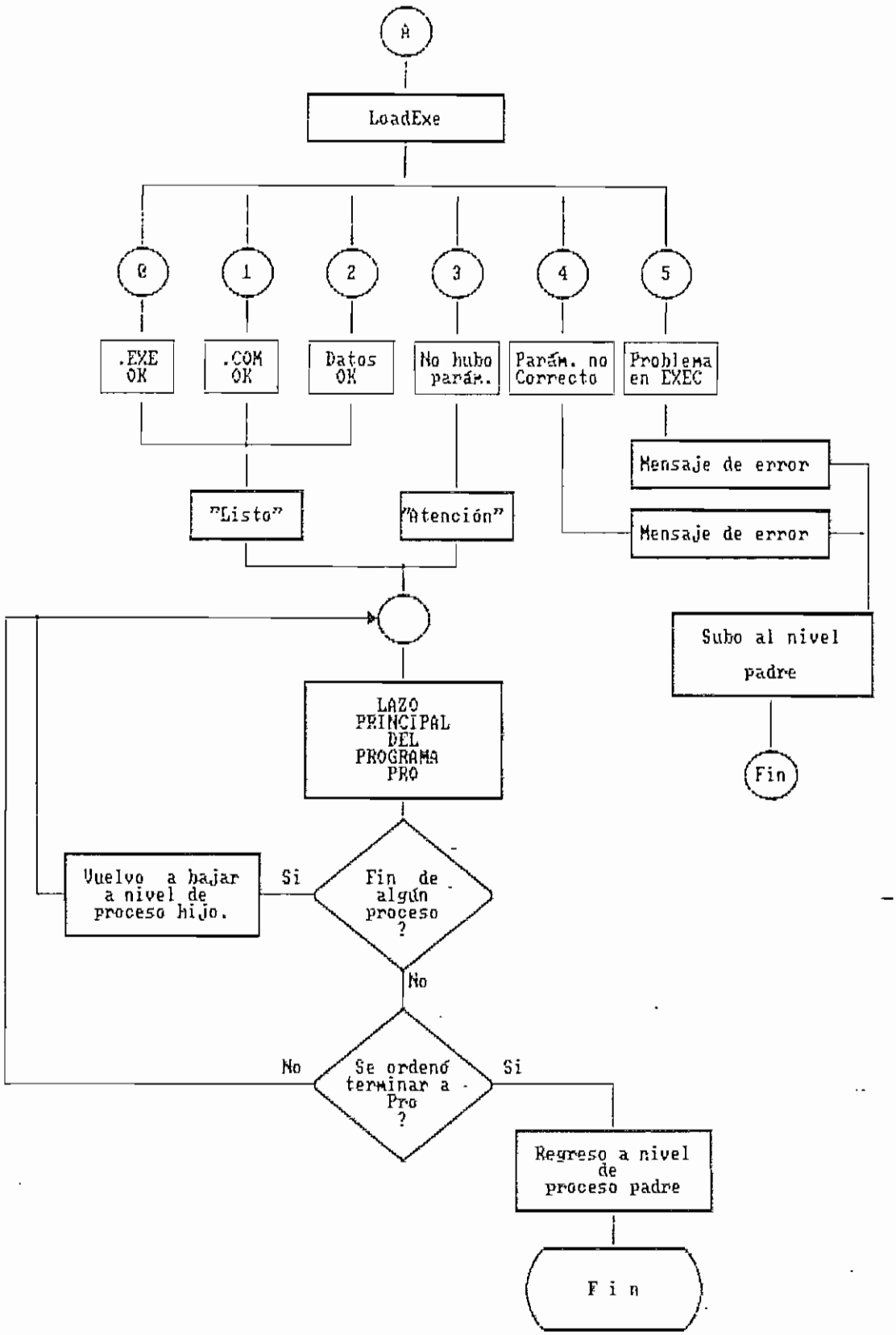


Diagrama A5 (cont...)

DIAGRAMA DE BLOQUES DEL PROGRAMA CRONO

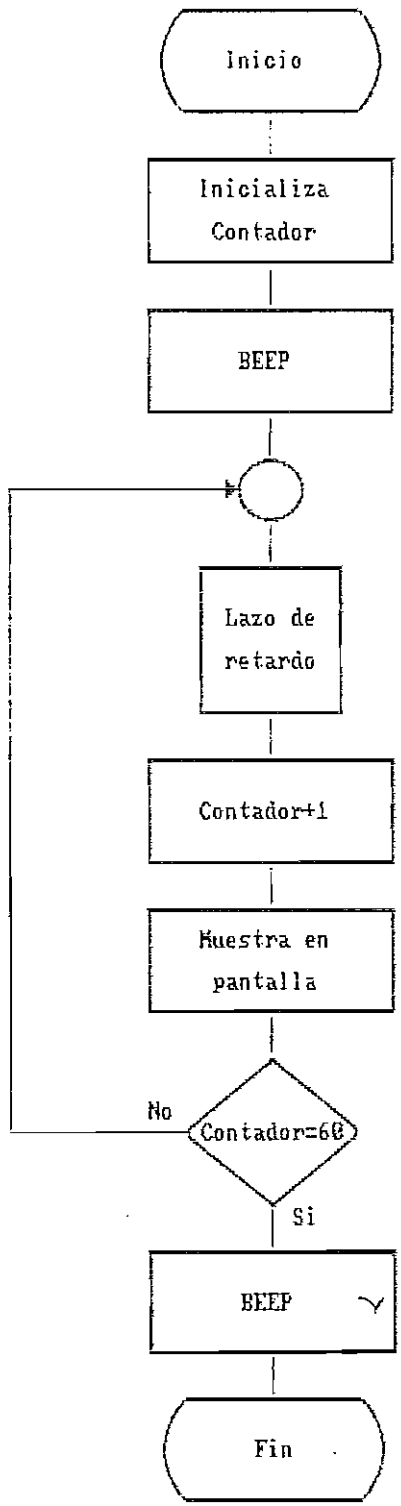


Diagrama A.6

DIAGRAMA DE BLOQUES DEL PROGRAMA TEST

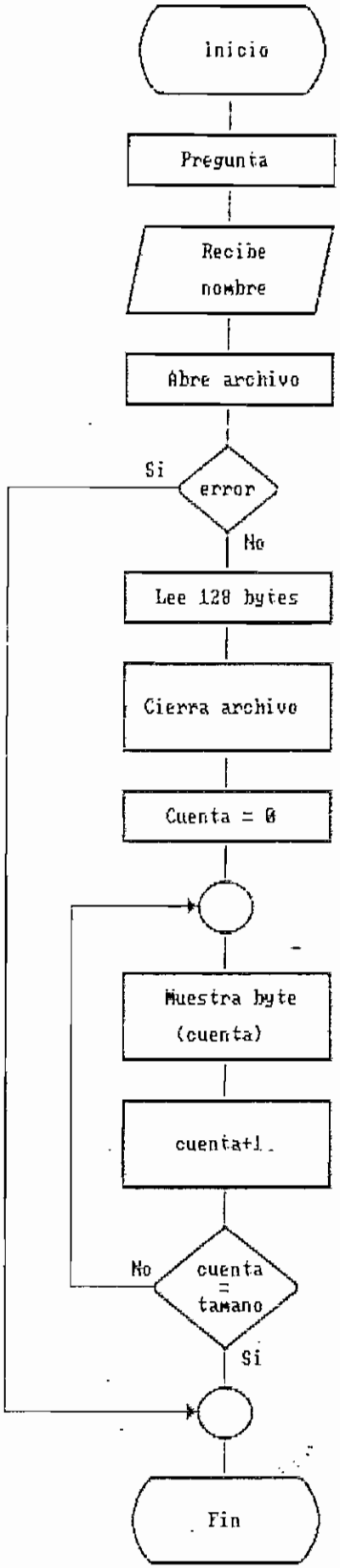


Diagrama A.7

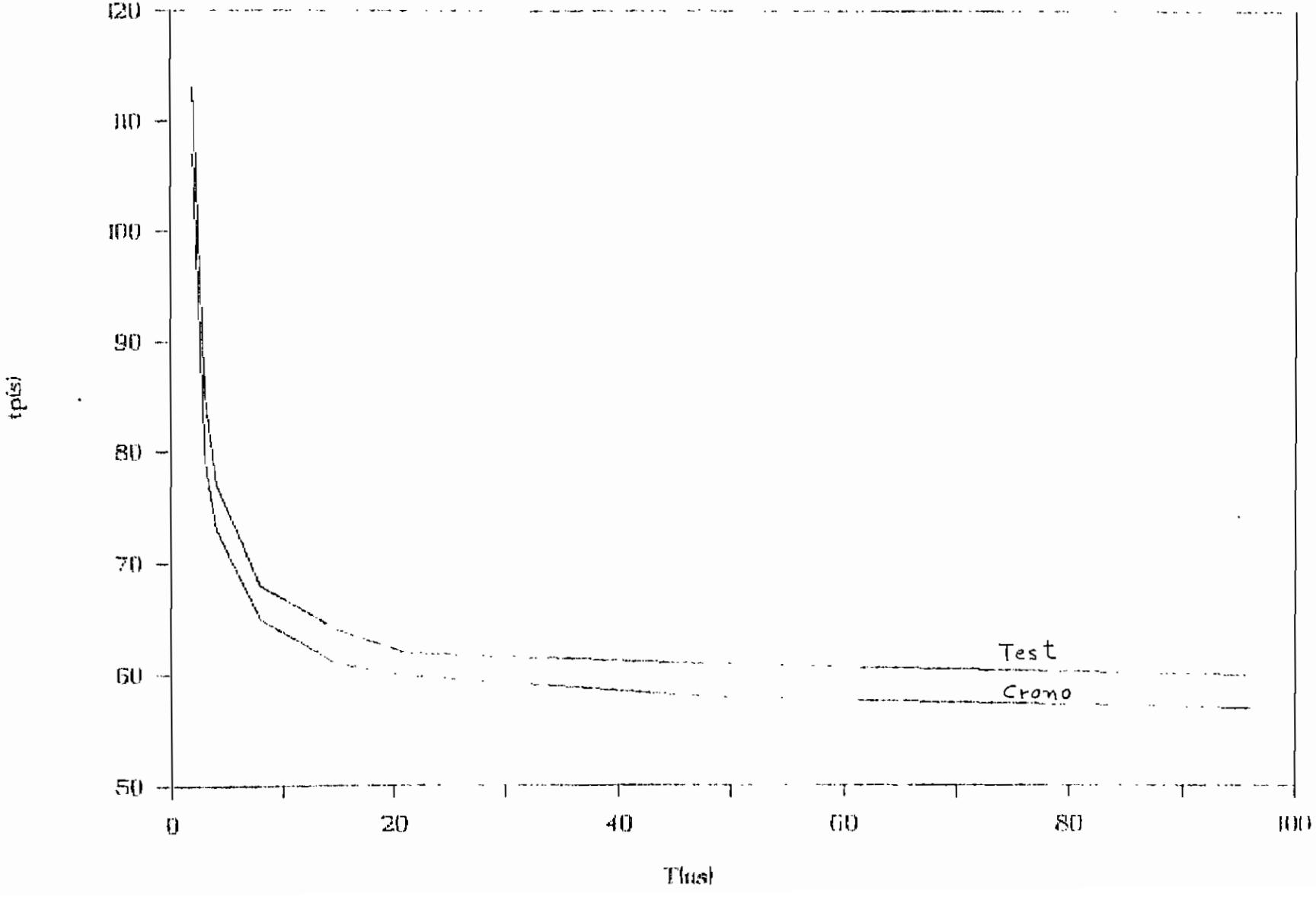
Tabla A.1

CABECERA DE LOS ARCHIVOS .EXE

Offset	Long	Descripción
00	2	Marca de Link para archivos .EXE (4DH,5AH)
02	2	Longitud del módulo imagen en páginas (bloques de 512 bytes)
04	2	Tamaño del archivo en páginas
06	2	Número de ítemes de relocalización
08	2	Tamaño de la cabecera en párrafos (incrementos de 16 bytes)
0A	2	Mínimo número de párrafos requerido sobre el final del programa cargado
0C	2	Máximo número de párrafos requerido sobre el final del programa cargado
0E	2	Desplazamiento en párrafos del segmento de Stack en el módulo cargado
10	2	Valor que se colocará en el registro SP cuando se entregue control al programa
12	2	Sumatoria negativa de todos los words en el archivo, ignorando sobreflujo
14	2	Valor que se colocará en el registro IP cuando se entregue control al programa
16	2	Desplazamiento en párrafos del segmento de código en el módulo cargado
18	2	Desplazamiento en bytes del primer ítem de relocalización en el archivo
1A	2	Número de overlay
??	?	Tabla de ítemes de relocalización

DURACION DE UNA PRUEBA tp

VS. PERIODO DE REFRESCAMIENTO T



Adaptación al tipo de monitor.-

PRO.EXE puede manejar tanto monitores monocromáticos como RGB; para éllo requiere dos archivos de ventanas diferentes. El primero, PROBN.VNT, es apropiado para monitores monocromáticos y el segundo, PROCOLOR.VNT, es el más apropiado para monitores RGB. El programa se encarga de determinar el tipo de monitor.

Ingreso al programa.-

Se invocará al depurador mediante la siguiente sintaxis:

PRO [Nombre de archivo] [Param1] ... [Param N]

(Los ítemes entre corchetes son opcionales)

El nombre de archivo se refiere al programa que se desea depurar (sea .COM o .EXE) o a un archivo de datos. Param1..ParamN son parámetros que se desee pasar al programa por depurar. Puede ingresarse sin necesidad de especificar un archivo, y luego leerlo desde el interior del programa.

Entonces el programa escogerá el tipo de monitor. De inmediato aparecerá un logotipo de introducción que además informará sobre el estado en que se ingresó (con o sin programa objetivo); si se ha producido un error fatal al ingreso (como la especificación de un nombre de archivo errada o la ausencia de uno de los archivos indispensables para que corra el programa), se informará al respecto y se abortará al pulsar una tecla. De no haber problemas, se presentará la pantalla del depurador con el menú principal.

Pantalla del Depurador.-

Incluimos un diagrama explicativo de la pantalla del depurador al final de este apéndice. Dicha pantalla consta de las siguientes ventanas:

a) Ventana de código.-

Contiene el código desensamblado de una parte de la memoria, normalmente aquel desde el cual apunta CS:IP. Para aprovechar mejor el espacio, no hemos colocado indicadores de tamaño de operador (como Byte Ptr), y los hemos remplazado por una letra al final de cada línea. Esta letra será una B para operadores de byte, una W para

f) Ventana de línea actual.-

Muestra el código de máquina de la instrucción a la que apunta CS:IP.

g) Ventana de Stack.-

Muestra los 8 words más superficiales en el Stack.

Opciones en los menúes.-

Varias de las opciones en los menúes tienen subopciones; estas se pueden ver mediante ^O. En cada menú hay siempre una opción resaltada, que se puede escoger mediante la tecla ENTER; se puede cambiar la opción resaltada mediante las flechas izquierda y derecha. También se puede escoger una opción simplemente oprimiendo la letra mayúscula de su nombre (por ejemplo, la J para eJecutar). Para ascender a un nivel superior se utilizará la tecla ESC. A continuación detallaremos las opciones del menú principal, con excepción de la opción "Funciones", que está en todos los menúes, y que lo único que hace es presentar un menú explicativo de lo que hace cada una de

b) Información.-

Muestra información sobre los autores y el lugar de origen del programa PRO.EXE.

c) Depurador.-

Contiene las subopciones de depuración de programas y de manipulación de datos que exponemos a continuación.

- Paso: Ejecuta la línea de código actual (aquella a la que apuntan CS:IP). Dependiendo de lo que se haya escogido mediante F8, se ingresará o no en subrutinas, repeticiones o interrupciones.

- Corre: Ejecuta el código del usuario desde la línea actual hasta que se cumpla un evento determinado; este evento será, según lo que se haya escogido con F7, el alcanzar una dirección determinada o el cumplir una cierta condición, que puede ser igualdad o desigualdad de un registro con un valor o una situación determinada del PSW. Si el programa del usuario finaliza sin haberse cumplido el evento dado, retornará el control al depurador.

- Ins: Permite ejecutar una instrucción fuera del código. Para éllo, se debe proporcionar el mnemotécnico de dicha instrucción, y ésta se ejecutará de inmediato.

- Ens: Tiene las siguientes subopciones:

  Ensambla: Permite alterar el código del usuario. Se proporcionará la dirección de memoria desde la cual se desea ensamblar, y luego una secuencia de mnemotécnicos.

  Busca: Explora el segmento de código del usuario para tratar de encontrar el código de máquina de un mnemotécnico específico.

- Memoria: Permite operar sobre la memoria. Tiene las siguientes subopciones:

  Llena: Coloca un valor de byte determinado en un bloque de memoria que empieza en el sitio apuntado por la ventana principal de memoria. Es necesario especificar la longitud del bloque y el byte que va a llenar el área.

Teclas de Función.-

Son accesibles desde cualquier nivel, y realizan los siguientes trabajos.

F1: Activa el editor interactivo. Este permite alterar cualquiera de las ventanas del depurador, sea el código, la dirección de desensamblado, el contenido de los registros, la dirección de la ventana de memoria absoluta, los apuntadores de la ventana relativa,o el contenido de la memoria (en hexadecimal o en ASCII). Para cambiar de ventana se utiliza TAB o Shift-Tab.

F2: Presenta una ventana de ayuda según el nivel en que se esté.

F3: Muestra las direcciones efectivas de los apuntadores de la ventana relativa de memoria.

F4: Muestra la pantalla del usuario. Esta es la pantalla que se asigna al programa del usuario cuando ejecuta. Para regresar al depurador, se debe oprimir ESC.

Se invoca las opciones de macros mediante ^K, lo que lleva al usuario a un menú en que se le presenta los nombres de los macros existentes en el archivo, y se le pide que proporcione el nombre de un macro. Si el usuario proporciona un nombre nuevo, el programa asume que el usuario desea grabar un nuevo macro, y se procederá en consecuencia. Entonces, se retornará al menú principal, y todo lo que haga entonces el usuario quedará entonces registrado en el macro, hasta el momento en que el usuario oprime ^K nuevamente, lo que indica fin de macro.

Si el usuario proporciona el nombre de un macro que ya existe, se le presentan 3 posibilidades: probarlo, correrlo o borrarlo. Probar un macro implica ejecutarlo paso a paso, oprimiendo la tecla ENTER cada vez. Correrlo implica ejecutarlo de una sola vez sin interrupciones, hasta su finalización. Y borrarlo implica eliminarlo del archivo.

Conjuntamente con el resto del paquete, entregamos un macro de ejemplo al que llamamos "SHELL". Lo que hace este macro es proporcionar el nombre C:\COMMAND.COM en Archivos, y luego leer dicho programa, que es el

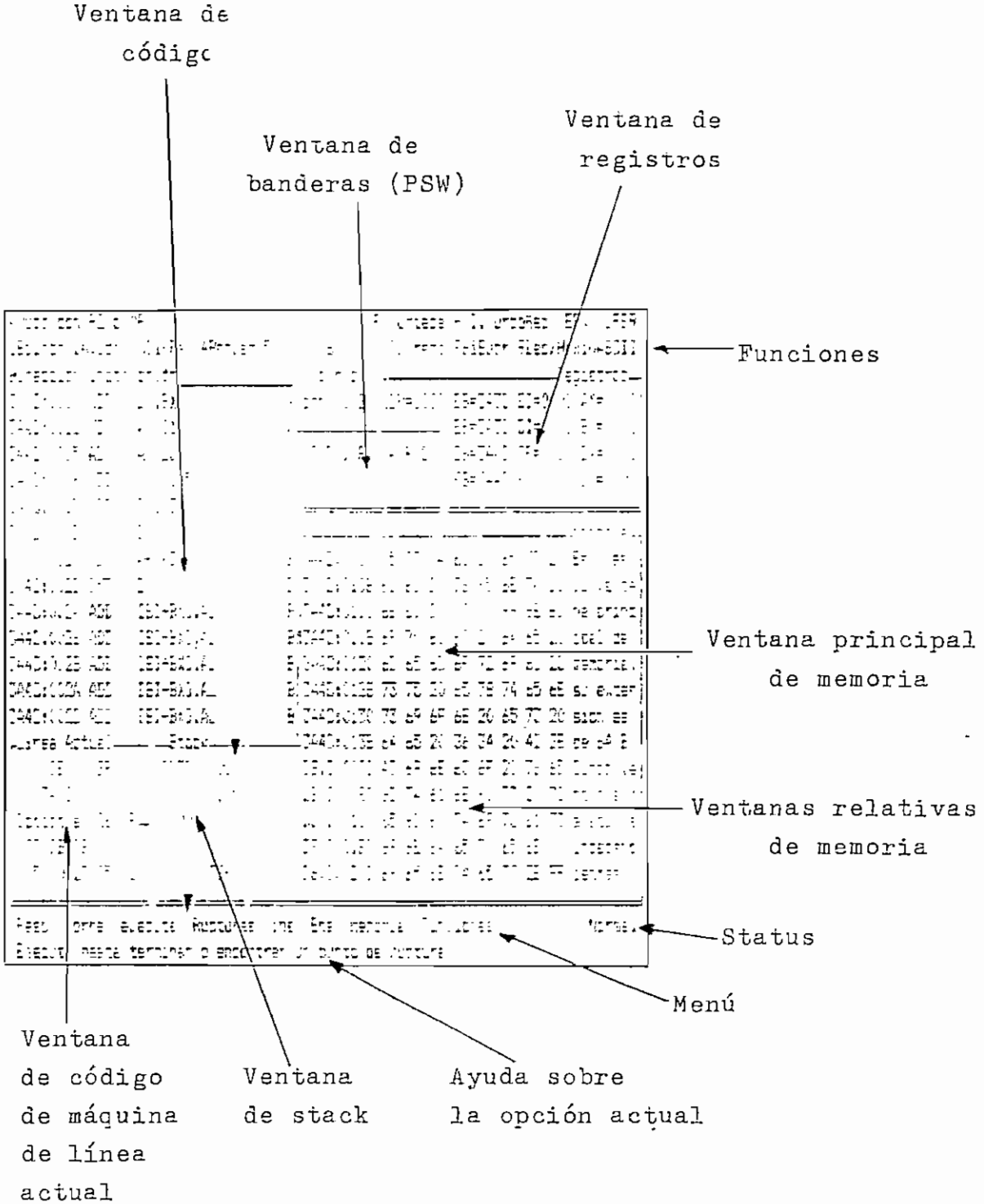
intérprete de comandos del DOS. Luego, ingresa a Depurador y eJecuta. Entonces, aparecerá el mensaje del DOS y se podrá ejecutar cualquier comando de este (por ejemplo DIR). Para retornar a PRO tipearemos EXIT, lo que continuará la ejecución del macro, ya que el depurador habrá recuperado el control; se ascenderá al menú principal y el macro terminará.

Limitaciones de PRO.EXE.-

Hemos encontrado algunos problemas con nuestro paquete de depuración, que escaparon a nuestra investigación durante el tiempo de desarrollo. Los exponemos a continuación a fin de prevenir al usuario, y consideraremos su corrección en una versión futura del programa.

- No es posible cargar en memoria un archivo completo que tenga más de 64 KBytes de extensión; si se intenta hacerlo, el programa no reportará ningún error, pero solo habrá cargado los primeros 64K.

P A N T A L L A D E L P R O G R A M A

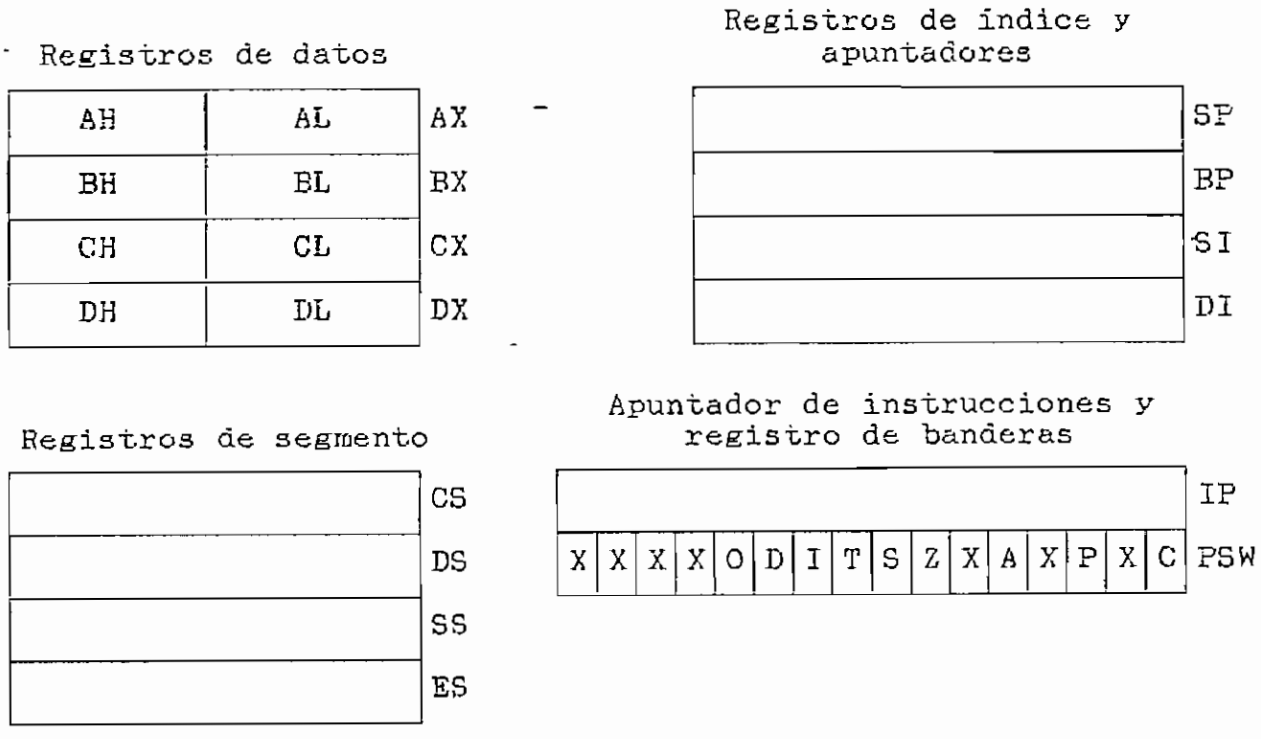


C. ESPECIFICACIONES DEL MICROPROCESADOR 8086

En este apéndice presentamos la estructura lógica que presenta este microprocesador. No pretendemos dar a conocer su set de instrucciones ni las técnicas de programación que se le aplican, sino hablar brevemente sobre sus registros y sus modos de direccionamiento.

Registros. El 8086 es un microprocesador de 16 bits, es decir que su mínima unidad lógica es el "word"; sin embargo, es posible tener acceso a bytes individuales, e incluso a bits aislados. Consta de 18 registros, a los que podemos clasificar en cuatro grupos:

Gráfico C.1: Registros de segmento



Los registros de datos tienen la característica de que pueden ser tratados como registros normales de 16 bits, o como registros dobles, compuestos por registros de 8 bits, a los que se puede tener acceso como si se tratara de registros independientes. Son de uso general, pero cada uno tiene características específicas. El registro AX es el acumulador (también lo es el AL en operaciones de 8 bits), y las operaciones que lo involucran suelen tener un tiempo de respuesta bastante pequeño. BX (base index) es un registro bastante especial, pues puede funcionar también como apuntador. CX es un contador que suele contener el número de iteraciones que se harán en procesos repetitivos. DX es un almacenador general de datos.

Los registros de índice pueden también ser utilizados como almacenadores de uso general, pero tienen los siguientes usos específicos. SP (Stack pointer) es un apuntador a una pila tipo FILO (first in - last out), y está implícitamente involucrado en operaciones como PUSH y POP. BP (Base pointer) es un apuntador normalmente asociado con la pila. SI (Source index) y DI (Destination Index) se involucran con operaciones de tiras de datos como MOVS.

Todos los apuntadores representan únicamente offsets, es decir, direcciones relativas respecto de otras absolutas. Tales direcciones absolutas se determinan mediante los llamados "registros de segmento", los cuales direccionan párrafos. Cada párrafo es un grupo de 16 bytes; así, si uno de los registros de segmento tiene un valor de 3, estará direccionando la memoria física 48. El registro CS (code segment) direcciona código; DS (data segment) direcciona datos, SS (stack segment) direcciona la pila, y ES (extra segment) suele también asociarse a datos.

Por último, IP (instruction pointer), apunta a la próxima instrucción que ejecutará el microprocesador dentro del segmento de código; ésto quiere decir que la dirección física de dicha instrucción será 16*CS+IP. PSW (program status word) es un registro en que cada bit es una "bandera", un indicador de estado de lo que viene ocurriendo en cada momento de la ejecución del código. Las banderas son las siguientes:

O: Overflow. Es 1 si una operación aritmética produjo un sobreflujo.

El contenido del PSW es el elemento de decisión en los saltos condicionales.

Modos de direccionamiento: El programador puede referirse a un lugar de la memoria mediante direccionamiento por índice proporcionando un registro de segmento y un offset (por ejemplo SS:[BX+DI+17]); los modos de direccionamiento posible son los mismos que se exponen en una tabla previa, en la explicación del módulo Assem.Asm. Sin embargo, es posible proporcionar únicamente el offset, y el microprocesador asumirá un registro de segmento por defecto, de la siguiente manera:

Tabla C.2: Registro de segmento predefinido

SI+BX	(+Desp)	DS
DI+BX	(+Desp)	DS
SI+BP	(+Desp)	SS
DI+BP	(+Desp)	SS
SI	(+Desp)	DS
DI	(+Desp)	DS
Desp.		DS
BX	(+Desp)	DS
BP	(+Desp)	SS

Finalmente, las operaciones de tiras de datos no aceptan parámetros, y siempre toman a DS:SI como fuente y a ES:DI como destino.

D. LLAMADAS A FUNCIONES DEL DOS Y DEL BIOS

En este apéndice listamos algunas funciones disponibles del BIOS y del DOS sin explicarlas en detalle. Todos los valores numéricos están en hexadecimal.

Tabla D.1

Funciones del BIOS.-

Tema	Int.	Servicio	Descripción
Impresión de pantalla	5	-----	Envia el contenido de la pantalla a la impresora
Video	10	0	Fija el modo de video
Video	10	1	Ajusta el tamaño del cursor
Video	10	2	Posiciona el cursor
Video	10	3	Lee la posición del cursor
Video	10	4	Lee la posición del lápiz óptico
Video	10	5	Elige la página activa
Video	10	5 (AL=80)	Obtiene los registros de la página de video
Video	10	5 (AL=81)	Da valores al registro de la página de video de la CPU
Video	10	5 (AL=82)	Da valores al registro de la página de video de la CRT

E. EL GENERADOR INTERACTIVO DE VENTANAS

Vent.Exe es un programa cuya misión es facilitar la creación de áreas de pantalla con un formato predefinido. Utiliza un controlador de flujo para evitar el tener que manejar esta labor.

Una ventana tan solo es un rectángulo de la pantalla. Puede contener información cuyo contenido varía a lo largo de la ejecución del programa o es constante. En el generador interactivo de ventanas cada posición de la pantalla se define por dos bytes: el carácter que la ocupa y el atributo que posee.

El programa se corre dando como parámetro el nombre del archivo que contiene las ventanas o se quiere_crear. Si el archivo es nuevo y el usuario confirma su creación, se inicia la corrida con un índice inicializado con valores que indican cero ventanas y cero grupos ocupados. Expliquemos aquí qué es un grupo. Entre los métodos usados para manejar informaciones que son creadas, editadas, borradas continuamente y con acceso aleatorio, uno muy práctico, y que lo emplean incluso los sistemas operativos es tener un índice en que se describe la ubicación de la información. Lo ideal sería tener todo el espacio ocupado, sin áreas desperdiciadas. Esto, sin embargo, no

es aplicable en la práctica. La cantidad de información necesaria para saber la ubicación de cada unidad elemental de memoria sería más grande que la información almacenada. Por éso se tiene que trabajar con áreas de memoria relativamente grandes, que se convierten en los elementos de la memoria. El área no debe ser muy pequeña por el problema descrito hace poco. Sin embargo, no debe ser muy grande porque el espacio no ocupado dentro pasa a ser inutilizable. El índice del programa de ventanas se inspira en el formato del directorio del CP/M, que es lo bastante sencillo como para adaptarse a nuestras necesidades sin dejar de ser efectivo. El tamaño del área es de 256 bytes, y es llamado un grupo en Vent.Exe.

Una ventana solo puede almacenarse en un número entero de grupos (en Vent.Exe el límite es cuatro grupos). La información de la ventana se almacena secuencialmente, y por cada ventana se mantiene un índice con datos importantes.

Las ventanas se crean dando un nombre, marcando dos esquinas suyas opuestas y grabando lo creado. Luego se define su contenido. En la opción "Caracteres" se definen los datos e informaciones que la ventana presentará. Si

alguna posición va a ser de valor variable, se la marca como "Variable". Luego se definen los "Atributos" de la ventana. Cada punto puede tener un diferente atributo. Aquí se define la presentación de la ventana.

Así se siguen creando las ventanas que algún programa necesita. Es conveniente que antes de empezar se tenga una planificación de la pantalla, al menos aproximada. En todo caso el programa permite eliminar ventanas, mostrar las existentes, alterar algunos de sus parámetros, etc..

Cuando una ventana es creada ocupa los primeros grupos vacíos que encuentre, aunque no sean consecutivos. Cuando una ventana es borrada deja sus grupos libres para que la próxima ventana creada los ocupe. Al cabo de cierto tiempo de creación y borrado de ventanas, es lo más normal encontrar un archivo completamente desordenado, con espacios vacios y que solo es manejable por el buen uso del índice. Esto se puede ver con las opciones Informa e íNdice del menú principal. Cada vez que el usuario necesite, y siempre que ya quiera usar el archivo de ventanas para su aplicación, deberá ordenar el archivo con la opción respectiva. El método de ordenación es este: No dejar grupos vacíos en el archivo, colocar primero las

ventanas principales, y luego las secundarias. El archivo se trunca siempre al final del último grupo válido.

En este momento hemos topado otro tópico interesante: Cuando el usuario planifica su programa sabe que existen ventanas que deben aparecer continuamente, pero otras que solo lo hacen de manera ocasional. Las ventanas que más aparezcan (principales), debieran ser instaladas en memoria y no retiradas, para que su lectura sea rápida. Las otras serían leídas de disco, a menos que hayan sido leídas recientemente y aún estén en memoria (concepto de residencia). Al crear una ventana, el usuario será interrogado por su categoría, que puede ser principal o secundaria. Podríamos pensar que lo óptimo es que todas las ventanas de un archivo sean principales, pero ésto lleva a tener que dejar muy extensas zonas de memoria dedicadas a contener las ventanas. Por éso Vent limita el número de grupos principales a 40, ésto es, unos 10 kB, que, bien utilizados podrían tener la definición de unas dos pantallas completas. Al referirnos a una buena utilización hablamos de que no haya demasiado desperdicio del espacio de los grupos. Este caso ocurre, por ejemplo, cuando se definen muchas ventanas de muy poca área.

Hasta este momento solo hemos hablado de la creación de ventanas. Cómo utilizarlas? Al final de este apéndice consta el esquema del índice y del archivo de ventanas. El usuario podría dar el uso que estime conveniente a esta información y crear sus rutinas de manejo de las ventanas creadas. Sin embargo, al escribir el programa Pro, hemos creado un archivo llamado Macvnt.Asm que podría ser incluido en cualquier programa en Assembler, para usar sus macros (Puede leerse el listado de Pro.Asm para observar la manera de incluir Macvnt y utilizar los macros del archivo).

Como se detalla en el listado de Macvnt, los macros principales son tres: El primero (IDVNT) sirve para indicar dónde el programa ha dejado un área para ventanas, cuál es el nombre del archivo de ventanas que se va a leer. y el tamaño del área de ventanas. Existe la necesidad de que las ventanas principales y el índice sean residentes (el índice ocupa 6 grupos), de que haya un área común de, al menos, 4 grupos para residencia temporal de las ventanas secundarias, y un grupo para uso del macro. Entonces el área debe ser de, al menos, el número de grupos principales más 11. Ese número de grupos principales es una de las informaciones dadas por Vent.

Tabla E.1: Esquema del archivo

Archivo de Ventanas	
Cada entrada del índice tiene 24 bytes.	
Índice: Grupos 0 a 5	Entrada 0.- Parametros Generales Entrada 1.- Mapas de ventanas y grupos Entrada 2.- Mapas de ventanas y grupos Entrada 3.- Parámetros ventana 3 ..... Entrada 63.- Parámetros ventana 63
Area de datos: Grupo 6 Grupo F9 (máximo)	

Tabla E.2: Parámetros generales

Parámetros Generales.		
Offset	Parámetro	Valor en índice: nuevo y (rango)
0	# ventanas válidas	0 (0-61)
1	# grupos válidos	0 (0-244)
2	Ultima ventana válida	2 (2-63)
3	Ultimo grupo válido	5 (5-249)
4	1a. ventana libre	3 (3-63)
5	1r. grupo libre	6 (6-249)
6	Ventana actual	3 (3-63)
7	Grupo actual	6 (6-249)
8	# ventanas principales	0 ------
9	# grupos principales	0 (0-40)
10-11	Dirección de residentes	--------
12	Tamaño área residentes	--------
13-14	Handle archivo ventanas	--------
15	Próx. grupo de área de residentes a ser usado	--------
16-24	Libres	--------

Tabla 3.1 Parámetros particulares

Parámetros de una ventana.		
Offset	Parámetro	Valores: Nuevo y (rango)
0 (8 bytes)	Nombre	-------
8 bit 0	Categoria	0 (0,1)
8 bit 1	Residencia	0 (0,1
9	# grupos en ventana	0 (0-4
10-11	# variables ventana	0 (0-512
12-13	Esquina sup. izq	0 (hasta 79,22)
14-15	Esquina inf. der.	0 (hasta 79,22)
16-19	#s de grupo	0 (6-F9 c/byte)
20-23	#s de grupo en área de residentes	(-------)

E. EMPLEO DEL CONTROLADOR DE FLUJO

En este apéndice presentamos un listado del esqueleto del programa controlado. Constan todas las declaraciones necesarias, se indica el sitio donde deben colocarse los buffers, la información de las opciones, las explicaciones, y los menúes y el macro Fetorno. indispensable para que el controlador funcione. Asi mismo, se indica la ubicación de los datos y rutinas del usuario. Base.Asm podría copiarse a otro archivo que se utilizaría como punto de partida.

Base.Asm

```
        .model  small
        .data
        public  tabbuf,tabinf,nmaxbuf
        extrn   dirret:near,mraux2:word,mraux3:word
        extrn   videoseg:word,valptr:word
        extrn   mopaux:byte,mopaux1:byte,tiprut:byte
        extrn   areamac:near,bgmac:byte,bmac:byte,bver:byte

nmaxbuf equ     ......                          ;# de buffers
tabbuf  db      01h,00h,00h,00h,00h,00h,00h,00h ;principal
        db      ......

tabinf  dw      offset exp0,offset menu0,offset rut0
        db      3
        dw      ......
        db      ...

exp0    db      ' ',0

menu0   db      'a    b    c    d',0,4,'ABCD',0
;************ datos del programa **********

        .code
        public  rinic,rfinal
```