UN DEPURADOR INTERACTIVO DE PROGRAMAS PARA EL MICROPROCESADOR 18086
P R O . E X E

ORDOÑEZ, IVAN ING.
ORTEGA, FERNANDO ING.
MONTALVO, LUIS ING.
VALENCIA, GALO ING.
ESCUELA POLITECNICA NACIONAL

RESUMEN

Se describe el diseño y desarrollo de un paquete de depuración de programas escritos en lenguaje de máquina del microprocesador i8086 en computadoras compatibles con la IBM PC bajo DOS, presentado como Tesis de Grado por los autores. El paquete está enteramente escrito en lenguaje Assembly en forma modular, y concebido para proporcionar una gran cantidad de información útil al usuario (estado de los registros del microprocesador, código desensamblado, áreas de memoria y stack), además de darle opciones de manejo de archivos y de depuración de su programa, es decir, de ejecución de su código en varias modalidades. La interfaz con el usuario es amigable y basada en menús.

ABSTRACT

Design and development of a program debugger for machine language code of the i8086 microprocessor in the IBM PC or compatible under DOS is described; it was developed by the authors as a Professional Degree's Thesis. This package was entirely written in Assembly Language by creating separate modules which were integrated afterwards, and was designed to give the user a great amount of useful information (state of the microprocessor's registers, disassembled code, memory and stack areas), and to give him several options of file management and program debugging, which implies execution of code in various different ways. The package is user friendly and menu driven.

INTRODUCCION

El paquete que se describe en este articulo fue desarrollado como Tesis de Grado de Ingeniero por los autores como respuesta a una necesidad concreta en el Laboratorio de Microcomputadores II de la Facultad de Ingeniería Eléctrica, que era la de contar con un depurador de programas para el microprocesador i8086 suficientemente amigable y didáctico para que pueda ser aprovechado por los estudiantes. Si bien existen otros depuradores en el mercado no cumplen con estos requerimientos. En Laboratorio de Microcomputadores I se cuenta con un programa simulador del microprocesador Z80 procurandose seguir un esquema similar en la elaboración de este programa, a fin de dar continuidad a la experiencia que los estudiantes adquieren con dicho simulador. El paquete no solo es un auxiliar de laboratorio sino también una herramienta bastante poderosa de desarrollo de software y de investigación pues está concebido para poder analizar con exhaustividad cualquier código en lenguaje de máquina, toda la memoria, y registros del microprocesador.

Antecedentes:

En el período Abril - Agosto de 1988 se orientó por primera vez en la Facultad de Ingeniería Eléctrica la materia de Microcomputadores II al estudio del microprocesador i8086. Dado el extendido uso de microcomputadores de la familia PC en el medio, se decidió que tanto esta familia como sus compatibles eran el sitio natural en el que se debían realizar los proyectos de programación. Las ventajas eran muchas: tener editores de texto adecuados, contar con un excelente ensamblador (el MASM), un "encadenador" de módulos (el LINK) y un programa depurador ya desarrollado (el DEBUG).

Se terminaron varios proyectos pero todos tropezaron, tarde o temprano, con las dificultades de la depuración con DEBUG. Algunos grupos de trabajo intentaron el uso de otro paquete, más poderoso pero menos difundido, llamado CODEVIEW, pero las dificultades fueron similares. Tanto DEBUG como CODEVIEW requieren la memorización de sus comandos y, si bien CODEVIEW muestra permanentemente una ventana de código, ni él ni DEBUG pueden presentar áreas de memoria actualizadas automáticamente. Estos factores motivaron al Ing. Luis Montalvo, así como a los autores del articulo, a planificar la elaboración de un depurador de programas escritos en Lenguaje Assembly, amistoso al usuario y que presente más información en pantalla que los otros programas.

El modelo obvio era el AVSIM Z80, simulador del microprocesador Z80 que posee las características indicadas y que ya era usado por los estudiantes en la materia Microcomputadores I. Esto permitiría una transición fácil del simulador del Z80 al depurador para el 8086.

Objetivos:

El depurador debe ser controlado en base a menús, evitando la necesidad de memorizar comandos.

Actualización de la información mostrada en la pantalla automática e interrelacionada.

Información a presentarse: contenido de los registros del microprocesador, estado de las banderas del mismo, código desensamblado, área de memoria con dirección absoluta y con direcciones relativas a registros, área apuntada por el puntero de instrucciones y del stack.

Debe poseer diferentes modos de ejecución del programa: paso a paso, corrida hasta el cumplimiento de un evento -que puede ser una dirección o una condición de registros o banderas-, lanzamiento del programa en tiempo real con la posibilidad de inclusión de puntos de ruptura y ejecución de instrucciones fuera del código ("parches").

Facilidades de manejo y alteración de áreas de memoria en ASCII o hexadecimal.

Un ensamblador debe ser parte principal del depurador, a fin de poder escribir programas o parches ingresando directamente los mnemotécnicos y no el código de máquina.

Permitir lectura y grabación de archivos con identificación del tipo de archivo.

DESCRIPCION DEL DEPURADOR

Dada la extensión del trabajo se decidió dividirlo en módulos encargados de funciones específicas. Un módulo puede contener varios procedimientos y algunos de éllos pueden agruparse para realizar una tarea compleja.

Los módulos que conforman a PRO son:

Fluio:

Este módulo es un controlador general de flujo para programas interactivos manejados por menúes. Se basa en la sistematización del proceso de presentación del menú, elección de una de sus alternativas y cumplimiento de la rutina respectiva. El ambiente que presenta este controlador es el mismo en cualquiera de los niveles del árbol de opciones y subopciones.

El controlador se adapta a las necesidades del "programa usuario" a través de tablas que ese programa debe contener. En la primera de esas tablas el usuario define el flujo que desea para su aplicación, indicando la forma de llegar a cada opción como una secuencia de números. La segunda tabla indica la ubicación de las informaciones asociadas a una opción: una breve explicación, el menú al que conduce -si existe-, la rutina que realiza su función y, por último, su tipo, que será definido más adelante. Luego se deben definir las explicaciones, los menúes y las rutinas asociadas.

Las opciones, en este contexto, pueden ser de tres tipos según el tratamiento que reciben luego de hacer su trabajo. Las opciones "normales" o de tipo 1 son aquellas que llevan al mismo menú al que pertenecen. Las "inversas" o de tipo 2 emergen al menú que les dio origen. Finalmente las opciones "rama" o de tipo 3 conducen a un nivel inferior presentando su menú asociado.

El módulo almacena las variables que definen la situación actual del programa: nivel y forma de llegar a la opción actual, y transfiere el control a la rutina elaborada por el usuario. Por medio de un macro de instrucciones de Lenguaje Assembly, que siempre debe ser usado, el flujo regresa al controlador. De esta forma el usuario no debe preocuparse de escribir código para manejar el flujo de su aplicación, sino solo las rutinas que hacen las funciones deseadas.

El esquema general de un programa interactivo controlado por este módulo es este: al comenzar se corre una rutina inicial creada por el usuario. Luego se entra al lazo principal del programa, en el cual, por medio de los menúes se llega a la realización de las tareas. Cuando se desea acabar, se corre una rutina de finalización, también creada por el usuario.

El controlador puede ser utilizado por cualquier aplicación para evitar el tener que desarrollar su propio manejo de flujo, con solo respetar la simple convención para la definición del árbol y el paso del control que fue descrita anteriormente.

Pro:

Pro es el módulo que invoca a los procedimientos contenidos en todos los otros módulos para cumplir las tareas de depuración. Se encarga de presentar la información en la pantalla y de inquirir al usuario del programa por la información que dichos procedimientos necesiten. Es aquí donde se deben encontrar las tablas usadas por el módulo de Flujo, tablas que definen el árbol de opciones y subopciones de esta aplicación. Una información más detallada sobre este módulo y sus funciones se encontrará en líneas posteriores, cuando sea referido como el módulo que integra las funciones de todos los demás.

Desens:

Este módulo es el desensamblador, y su función es la de convertir código en lenguaje de máquina a mnemotécnicos desensamblados. Se utiliza para generar la información de la ventana de código, que es necesaria para que el usuario pueda seguir el flujo de su programa. Contiene varios submódulos que interactúan; al de más alto nivel es DesensDet, que lo que hace es llamar a Desens, para luego reformatear la tira de caracteres que el último pasa a fin de prepararla para ser utilizada por PRO en la ventana de código.

El submódulo más importante es Desens, que es el que realmente genera la tira alfanumérica que representa al mnemotécnico, y además proporciona la longitud en bytes del código de máquina desensamblado, a fin de que el procedimiento invocante pueda localizar la siguiente instrucción.

Finalmente, hay varios otros submódulos utilizados por Desens, que realizan una gran cantidad de funciones, como conversión de datos y operaciones en tiras de caracteres.

Assem:

Este módulo cumple la función inversa del anterior: convierte cualquier mnemotécnico válido del 8086 en su código de máquina correspondiente, y además proporciona la longitud en bytes de este código. Su submódulo de más alto nivel es Assem, que invoca sucesivamente a los dos pasos de compilación.

En el primer paso se genera un "pseudocódigo", es decir, una representación numérica codificada del mnemotécnico que no llega a ser el código de máquina verdadero; simplemente se reduce cada símbolo a un número. El segundo paso parte del pseudocódigo para generar el verdadero código.

Este módulo es indispensable para que el usuario pueda hacer "parches" o "remiendos" a su código sin que deba tomarse el trabajo de ensamblar a mano la nueva instrucción que desee incluir. También permite escribir pequeños módulos ejecutables directamente en el depurador.

Ejecutor:

Este módulo no cumple una función específica, sino que contiene un grupo de procedimientos de control de flujo de subprogramas, a fin de posibilitar la depuración, y de manejo de archivos y de memoria. A continuación se describe los más importantes:

TestTrap, prueba el comportamiento de la interrupción 1, que se debe producir una vez que la bandera de trampa (Trap Flag) T se ponga en 1 y transcurra una instrucción más. Sin embargo, detectamos que en algunos microprocesadores esta interrupción demora dos instrucciones, y no una, en producirse; el procedimiento determina si este es el caso, y fija una bandera indicadora de memoria que le dice al procedimiento ExUniq cómo actuar.

ExUniq, ejecuta una sola instrucción del programa del usuario, para lo cual hace 1 a la bandera de trap T tras haber transferido el control a dicho programa y ejecutado un NOP si TestTrap le dijo que la INT 1 se retarda en el microprocesador. Recupera el control tras ejecutar la instrucción del usuario porque la interrupción está intervenida.

ExHasta, ejecuta el código del usuario desde una dirección inicial hasta otra de terminación. Para ello coloca un INT 3 en la dirección de terminación y recupera el control en ese punto, porque dicha interrupción está intervenida.

ExTotal, ejecuta el código del usuario hasta encontrar uno de los puntos de ruptura definidos en una tabla. Esta tabla se genera y modifica mediante tres procedimientos que también constan en el módulo Ejecutor y son: AddBrkPt, que añade un punto de ruptura activo a la tabla, DelBrkPt, que elimina uno, y TogBrkPt, que activa o desactiva un punto de ruptura.

ExIns, ejecuta una instrucción fuera del código del usuario, dado su mnemotécnico. Para ello, invoca a Assem y luego a ExUniq.

ReduceLevel, invoca a la función EXEC del DOS a fin de ejecutar el subprograma Child.Exe, que simplemente devuelve el control sin finalizar. De esta forma, se consigue "bajar de nivel", es decir, engañar al sistema operativo para que si se ejecuta 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 se asigna al programa cargado mediante LoadExe.

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 archivo.COM u otro cualquiera. Puesto que los archivos .EXE requieren de una cabecera con información normalmente generada por el linker y no se ha 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, modificarlos y grabarlos, siempre y cuando no contengan información de relocalización, pues ésta sería alterada.

Se recomienda 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.

Memory:

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.

Integración de todos los módulos:

Todos los módulos son llamados por el programa principal PRO, pero dependiendo de FLUJO para el manejo de menús. El funcionamiento se expone a continuación:

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 se crearon utilizando "vent.exe", que es un programa que permite definir ventanas en la pantalla con información fija y con campos que aceptan información dinámicamente del usuario o de un programa. Siguiendo la convención necesaria PRO 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; creandose el archivo de macros si no existe uno.

Si la apertura 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 una vez reducida ésta a los requerimientos del programa (1300H párrafos, es decir, 77824 bytes), el resto puede ser utilizado por el programa del usuario.

Se instala en memoria un servicio para la interrupción 2FH (llamada "multiplex"), que recibe como parámetro un número de función en AH, estando disponibles los números desde 80H hasta F0H. Se reconoce como perteneciente a PRO la función 80H; la cual responde restaurando el stack (alterado por efecto de la interrupción) y devolviendo el control al programa sin ejecutar jamás un IRET.

Una vèz instalado el servicio se invoca a la función EXEC para correr el programa Child.Exe que invoca 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 se sigue ejecutando un subproceso de PRO. El efecto de ésto es poder transferir temporalmente el control al programa del usuario sin temor de que este 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, PRO; que nuevamente ejecuta Child.Exe para 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 convierta en el PSP del programa contenido en dicho archivo. Con éllo se evita a PRO el tener que construir un PSP, ya que EXEC puede hacerlo.

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 una rutina que pone en el video todas 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 eventualmente abra no pueden pasarse al proceso invocante, de forma que si el programa que está siendo depurado termina, todos sus archivos quedan automáticamente cerrados.

Cuando el usuario decide terminar de utilizar el programa se invoca la rutina de finalización, que cierra los archivos de ventanas y de macros, retira el servicio de la interrupción multiplex, y ejecuta un EXIT para subir de nivel, desechando así el programa del usuario, y dejando todo listo para poder salir mediante otro EXIT.

La pantalla principal del depurador PRO, se presenta en la Fig. 1.1.

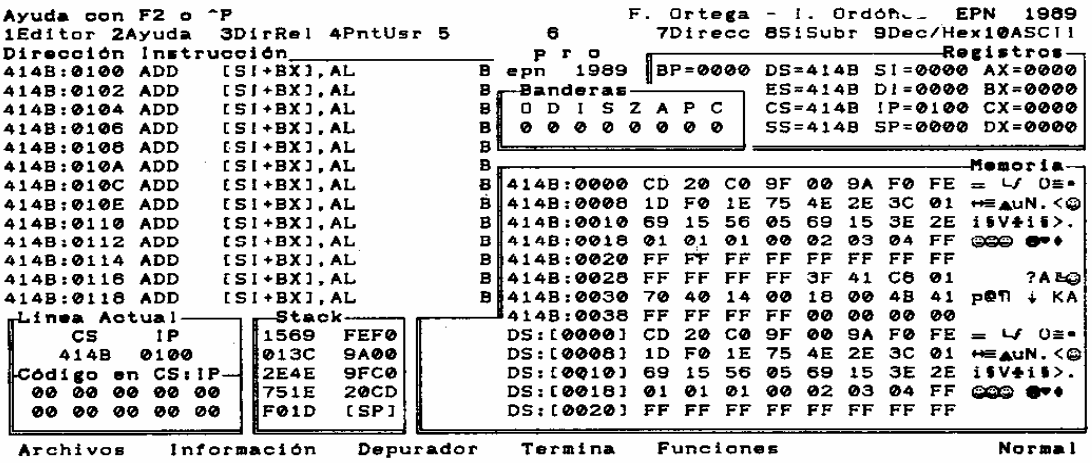


Fig. 1.1 Pantalla principal del depurador PRO

PROYECCIONES

PRO será utilizado por primera vez como herramienta de laboratorio en el semestre Abril Octubre de 1989 en la materia Microcomputadores II de la Facultad de Ingeniería Eléctrica de la EPN. Se espera que en este periodo los estudiantes colaboren con observaciones que permitan depurar el paquete. Debido a la gran ayuda que representa para el programador e investigador puede intentarse una comercialización del programa que beneficiaría a la Facultad y a la institución. Esperamos que esta iniciativa sea un ejemplo que motive a otros estudiantes y profesores a desarrollar instrumentos de investigación en los otros campos que la institución lo requiera.

CONCLUSIONES

De la investigación que fue necesaria para el desarrollo de PRO se llegó a las siguientes conclusiones:

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, lo cual 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 trap T ha sido seteada (la cual debería ser una sola instrucción, pero a veces son más). Esto condujo al 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. Debe señalarse 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.

PRO.EXE tiene una habilidad, que ningún otro depurador que los autores conozcan posee, la cual es 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). Se consigue 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. 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, más no es imperativa, debido al objetivo didáctico con el que fue concebido.

El módulo ensamblador representó un paso muy interesante en la elaboración de PRO ya que 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 indujo a los autores a trabajar en dos etapas independientes e interrelacionadas:

En la primera se chequea la validez de los elementos sintácticos del mnemotécnico y se lo reduce a una secuencia de bytes que representa a dichos elementos en forma condensada, llamada "pseudocódigo". En la segunda, se genera el código de máquina a partir del pseudocódigo. 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 se recomienda este estilo de programación a cualquier persona o grupo de personas que planeen desarrollar un paquete relativamente extenso.

Complementando este criterio debe decirse que los módulos de PRO 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 posterior integración a otros proyectos.

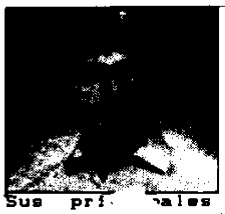
Por otro lado de quienes dirigen el desarrollo de Tesis o trabajos que involucren a la programación, se requiere que destinen un gran esfuerzo a la planificación, para lograr un avance coordinado y no disperso, sin duplicación de trabajo y sobre todo con conocimiento de causas y objetivos.

REFERENCIAS

1. AT&T, System Programmer's Guide, Agora Resources, Lexington, MA. 1984.
2. Epson, MS-DOS For the Equity + Systems, Seiko-Epson Corp., 1986.
3. IBM, Disk Operating System Technical Reference Manual (Programming Family), IBM Corp., 1985.
4. Microsoft, MS-DOS Programmer's Reference Manual, Microsoft Corp., 1983.
5. Microsoft, Macro Assembler 5.0 Reference, Microsoft Corp., 1987.
6. Microsoft, Macro Assembler Programmer's Guide, Microsoft Corp., 1987.
7. Miller Alan, Mastering CP/M, Sybex Inc, Berkeley, CA. 1983.
8. Norton Peter, El IBM PC a fondo, Anaya Multimedia, 1985.
9. Norton, Peter, Guía del Programador para el IBM PC, Anaya Multimedia, 1985.
10. Ordóñez, Iván; Ortega, Fernando, Depurador de programas para el microprocesador i8086, Tesis de Grado, EPN, Quito, 1989.

BIOGRAFIAS

ORDOÑEZ, IVAN.- Nació en Guayaquil el 29 de Marzo de 1965. Obtuvo el título de Ingeniero en Electrónica y Telecomunicaciones en la Escuela Politécnica Nacional, en Mayo de 1989. Sus principales áreas de interés son: desarrollo de software en diversos lenguajes de programación y especialmente en lenguaje Assembly.



ORTEGA, FERNANDO.- Nació en Quito el 30 de Junio de 1964. Obtuvo el título de Ingeniero en Electrónica y Telecomunicaciones en la Escuela Politécnica Nacional, en Mayo de 1989. Sus principales areas de interés son: Comunicaciones de datos, desarrollo de software, hardware de computadores.