ESCUELA POLITECNICA NACIONAL

FACULTAD DE INGENIERIA ELECTRICA

ESPECIALIZACION EN ELECTRONICA Y TELECOMUNICACIONES

DEPURADOR DE PROGRAMAS PARA EL MICROPROCESADOR 80486

JORGE BOLIVAR OSORIO HINOJOSA

ALEX JOHNINE TROYA ALDAZ

TESIS PREVIA A LA OBTENCION DEL TITULO DE INGENIERO EN LA ESPECIALIZACION DE INGENIERIA ELECTRONICA Y TELECOMUNICACIONES DE LA ESCUELA POLITECNICA NACIONAL

MAYO 1997

CERTIFICACION

Certifico que la presente tesis fue
realizada en su totalidad por los Srs.:
Jorge B. Osorio Hinojosa y Alex J. Troya Aldaz
bajo mi asesoramiento.

Ing. Jaime Velarde Guevara

Dedicatoria :

A Dios por la sabiduría que nos brinda para seguir el camino del bien
a mis padres Elva y Esteban que me han apoyado en todo momento
a mis hermanos Henry, Livy, Conzuelo, Pedro y Hugo que me incentivan a seguir adelante

Alex J. Troya Aldaz

Agradecimiento :

A las personas que colaboraron para que esta tesis pueda ser terminada con éxito a la Escuela Politécnica Nacional que nos da la oportunidad de superarnos

Alex J. Troya Aldaz

CONTENIDO

CAPITULO 3

HERRAMIENTAS DE PROGRAMACION DEL 80486

Capítulo 1

NECESIDAD DE UN DEPURADOR DE PROGRAMAS

1.1 Justificación del Tema

En los últimos años las computadoras personales han tenido un desarrollo acelerado, especialmente el Microprocesador constituido en el cerebro del computador y por consiguiente en su parte más importante, las nuevas generaciones de microprocesadores incorpora nuevas facilidades y ventajas. A partir de algunas versiones de los microprocesadores Intel 80486 se incluye en el mismo circuito integrado al Coprocesador Matemático que abarca su propio conjunto de instrucciones.

La meta es aumentar la capacidad, eficiencia, velocidad y los recursos de los microprocesadores; si comparamos un microprocesador 8086 con un microprocesador 80486 veremos que a pesar de haber total compatibilidad, éste ultimo incrementa el número de Instrucciones buscando facilitar el proceso de programación, el número de registros de segmento, el tamaño de los registros de propósito general, y la capacidad para direccionar memoria de los microprocesadores es cada vez mayor. Ventajas que le han permitido incursionar con gran éxito en nuevos campos de aplicación : Trabajo en ambientes gráficos que permiten el desarrollo de programas más amistosos para el usuario, manejo de audio y vídeo, sistemas multiusuario, sistemas multitarea, adquisición de datos a mayor velocidad, etc.

Considerando que los microprocesadores son objeto de estudio en la materia de Microcomputadoras a surgido la necesidad de disponer de un Depurador de Programas en Lenguaje Ensamblador (lo nombraremos como Depurador) acorde a las características de los microprocesadores Intel 80486. Este programa didáctico además de permitir controlar el flujo de ejecución del Programa desarrollado por el usuario, facilitará el entendimiento del funcionamiento del microprocesador por su característica de mostrar explícitamente como varía el estado de registros, banderas y localidades de memoria controladas por el microprocesador y que han sido afectadas con la ejecución de una instrucción. Si consideramos que el ensamblador es el lenguaje más básico pero a la vez el más complejo en el que resulta critico el control adecuado de los registros del microprocesador y la memoria direccionada, se justifica la importancia de un Depurador para facilitar el desarrollo de los programas de aplicación.

Debido a que los microprocesadores Intel y compatibles son los de mayor uso en las computadoras a nivel mundial, se plantea prácticamente como un obligación que el Depurador desarrollado sea diseñado para funcionar en éstas computadoras. La programación en lenguaje ensamblador permite conocer con mayor profundidad el funcionamiento básico de la máquina, ganar rapidez en la ejecución de los programas, tener acceso a posibilidades de la máquina que son inaccesibles desde otros lenguajes, poder utilizar las posibilidades de la máquina desde los propios lenguajes de alto nivel mediante la realización de módulos.

1.2 Presentación de Objetivos

Para el desarrollo de la presente tesis se han planteado los siguientes objetivos, basados en el análisis que se hace más adelante acerca de los depuradores :

Desarrollar un Depurador amigable y de fácil uso para el usuario, que mantenga disponible a primera mano la mayor parte de la información importante durante la depuración. Poder acceder fácilmente a la información que no se encuentra en la pantalla principal, en base a menúes sencillos y cuyas opciones se encuentren siempre a disposición. Implementar el uso del ratón para agilitar el acceso a las diferentes opciones del programa.

Incluir una ventana opcional para el Coprocesador Matemático que pueda ser habilitada o Inhabilitada por el usuario. La ventana contendrá toda la información de las banderas y registros internos del coprocesador matemático.

Permitir que el usuario tenga acceso a las ventanas de banderas, registros y memoria para realizar cambios directamente de tal manera que pueda realizar pruebas de funcionamiento del programa bajo diversos valores de datos. Esto es importante especialmente para el caso de saltos condicionales o para comparaciones.

Ofrecer al usuario herramientas adicionales para facilitar el proceso de depuración, como por ejemplo: Permitir la salida al entorno del sistema sin abandonar el depurador, poder ver y editar el archivo fuente (.asm) durante la depuración, depuración paso a paso, depuración hasta una cierta condición o dirección, depuración usando puntos de parada, ingresar o no ingresar a las subrutinas, entre otras.

1.3 Introducción a Depuradores de Programas

El depurador es una herramienta muy útil para el programador, permite seguir la ejecución de programa verificando el estado de los datos y la información que presenta el programa en pantalla. Mediante la depuración se pueden detectar y corregir errores de operación, hacer más eficiente el funcionamiento del programa, y probar varias opciones de funcionamiento.

El proceso de programación se puede considerar como un proceso cíclico (ver Figura 1.1). El paso de depuración no es necesariamente obligatorio como los demás pasos pero si de mucha utilidad y que en ciertos casos puede ahorrarnos mucho tiempo.

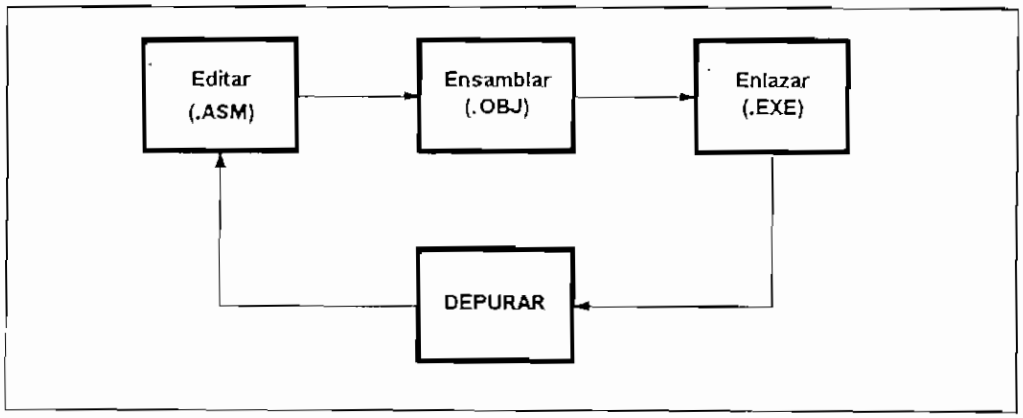


FIGURA 1.1 Pasos del Proceso de programación

En los Lenguajes de Alto Nivel todos los pasos del proceso de programación están incluidos como opciones en un solo programa, dependiendo de la versión del lenguaje, el depurador será más avanzado. Algunos depuradores solo permiten correr completamente el programa sin dar la opción de verificar el valor

de las variables ; en estos casos se hace necesario insertar en el mismo código puntos de parada e instrucciones que presenten los valores de las variables por pantalla. Otras versiones más avanzadas ofrecen mayores facilidades como: ejecución paso a paso, ejecución completa, inserción de puntos de parada, ingreso o no a subrutinas, ventanas que muestran y permiten modificar las variables seleccionadas.

En el caso del lenguaje Ensamblador el enfoque es diferente, cada paso del proceso de programación es realizado por un programa independiente. En el mercado actual se dispone de varios programas para realizar depuración en lenguaje ensamblador, cada uno de los cuales poseen sus ventajas y desventajas. Una desventaja común de todos estos depuradores es que no disponen de una ventana de memoria dinámica y de fácil acceso que muestre la variación de los datos a cada paso y de la facilidad de modificar directamente los contenidos de memoria para probar el funcionamiento del programa bajo diversas posibilidades de datos. Algunos depuradores no dispone de una ventana para los datos del coprocesador matemático, no admiten el acceso para modificar directamente el valor de los registros y de las banderas, entre otras. El uso de la mayoría de los depuradores es difícil por el trabajo para acceder a las diversas opciones debido a la gran cantidad de comandos existentes que generalmente no se conocen en su totalidad.

Debug uno de los depuradores comerciales más elementales que viene incluido entre los archivos del DOS, requiere del usuario un conocimiento profundo de la ubicación en memoria del archivo a depurar, y de las localidades de datos. El programa no diferencia entre su pantalla y la pantalla de usuario motivo por el que existe una mezcla de información, muestra en pantalla únicamente la

información que le sea requerida mediante comandos. Entre otros depuradores disponibles tenemos a Code View de Microsoft, TD286 que viene incluido con Borland -C. Son depuradores más amigables y con mayores facilidades que Debug pero con las desventajas que se mencionaron anteriormente.

En esta tesis se pretende crear un depurador que conserve las ventajas y corregir las desventajas que ofrecen los depuradores comerciales, para lo cual hemos tomado como base al depurador PRO para los microprocesadores 8086 desarrollado como tesis de grado en nuestra facultad.

CAPITULO 2

MICROPROCESADORES

2.1 Evolución de los Microprocesadores

A fin de lograr un enfoque claro, rápido y sencillo de la familia de microprocesadores Intel, a estos se los analizará independientemente y de una manera breve iniciando por el 8086 hasta el 80686, que son una evolución de los primeros microprocesadores de 4 y 8 bits.

Antes de comenzar los microprocesadores modernos, se debe primero entender qué fue lo que puso a estos dispositivos en primer plano. La historia dice que los antiguos babilonios empezaron a usar el ábaco, (calculadora primitiva hecha con cuentas o esferas ahuecadas), alrededor del año 500 a. de C. Con el tiempo, esta sencilla calculadora, estimuló a la humanidad para perfeccionar una maquinaria calculadora en la que se utilizaban engranes y ruedas (Blas Pascal en 1642). Se continuaron los progresos con las gigantescas máquinas computadoras de las décadas de 1940 y 1950, construidas con relevadores y tubos de vacio (bulbos). más adelante, se utilizaron los transistores y los componentes electrónicos de estado sólido para construir las poderosas computadoras de la década de 1960. Con el advenimiento de los circuitos integrados se llegó al perfeccionamiento del microprocesador y de los sistemas de microcomputadoras.

En 1971, Intel Corporation y Marcian Hoff lanzaron el primer microprocesador: el 4004, de 4 bits. Este controlador integrado, programable en un solo encapsulado era insuficiente, según las normas actuales , porque solo direccionaba 4096 localidades de 4 bits en la memoria. El 4004 contenía un conjunto de instrucciones que ofrecían sólo 45 instrucciones diferentes. Como consecuencia, el 4004 sólo se podía emplear en aplicaciones limitadas, como en los primeros juegos de vídeo y en controladores pequeños basados en

microprocesadores. Cuando surgieron aplicaciones más complejas para el microprocesador , el 4004 resultó inadecuado.

Más tarde, en 1972 Intel Corporation produjo el 8008, el primer microprocesador de 8 bits. El tamaño ampliado de la memoria (16k) y las instrucciones adicionales (un total de 48) en este nuevo microprocesador brindaron la oportunidad de muchas aplicaciones más avanzadas. Posteriormente, en 1973, Intel Corporation introdujo el 8080, el primero de los microprocesadores modernos de 8 bits el cual no sólo direccionaba más en la memoria sino que también ejecutaba más instrucciones, y con diez veces más rapidez que el 8008. Para 1977 Intel Corporation introdujo una nueva versión del 8080: el 8085. Aunque sólo ligeramente más avanzado que el 8080, el 8085 direcciona la misma cantidad de memoria, y ejecuta más o menos el mismo número de instrucciones se diferencia por ser más rápido además de que el generador de reloj y el controlador del sistema eran integrados en el mismo chip, convirtiéndolo en uno de los microprocesadores Intel más conocidos.

2.1.1 Microprocesador 8086

Uno de los acontecimientos más significativos del año 1978 en el campo de los microprocesadores fué cuando Intel Corporation comenzó la producción de la primera CPU con unas prestaciones similares a las de un minicomputador de propósito general y de alto rendimiento.

El 8086 es un microprocesador fabricado con tecnología HMOS (MOS de alta velocidad), que posibilitó la implementación en una área muy pequeña del equivalente a casi 30000 elementos activos. El mayor acierto del diseño del 8086 consistió en producir una máquina con 16 bits de resolución, pero sin las restricciones clásicas que aparecían en dichas máquinas cuando se manejaban datos de 8 bits o cadenas de datos de 8 bits. El 8086 fue diseñado para trabajar con lenguajes de alto nivel y dispone de un soporte de hardware con el que los

programas escritos en lenguaje de alto nivel ocupan un pequeño espacio de código y pueden ejecutarse con alta velocidad. Un año mas tarde, Intel Corporation lanzó el microprocesador 8088, siendo este también un microprocesador de 16 bits, que ejecutan instrucciones en escasos 400 ns; una gran mejoría en relación con la velocidad de ejecución del 8085. El 8086 y el 8088, tienen también capacidad para direccionar a 1 M byte. Las velocidades más altas de ejecución y el tamaño de memoria más grande, permitieron al 8086 y 8088 sustituir a mini computadoras pequeñas en muchas aplicaciones.

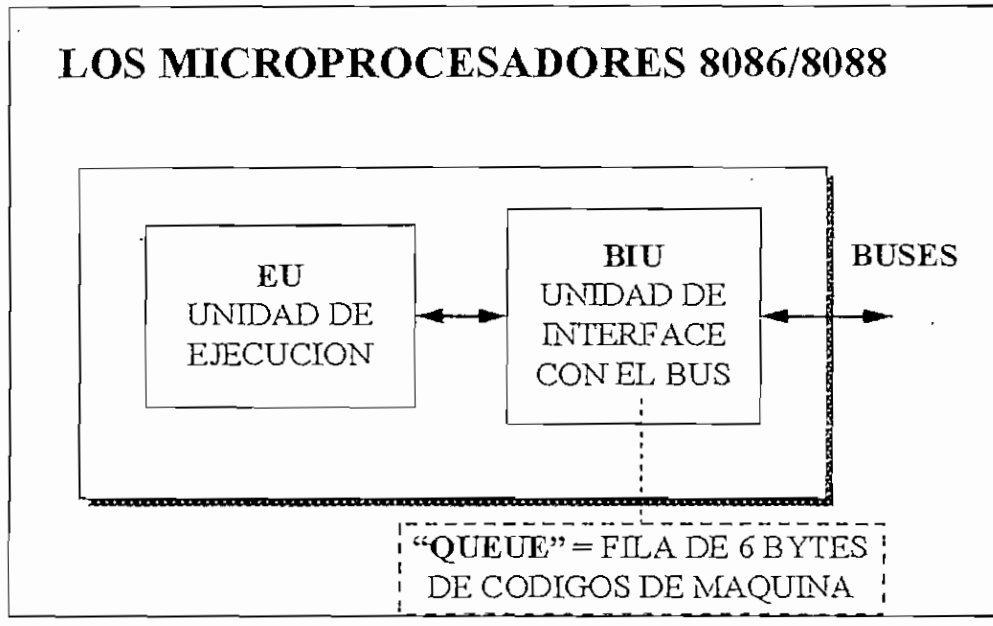


FIGURA 2.1 Diagrama de bloques del microprocesador 8086/8088

2.1.2 Microprocesador 80186

La evolución del microprocesador de 16 bits no terminó en el 8086 y el 8088, sino que continuó con la introducción del 80186, una versión altamente integrada del 8086. El 80186 se utiliza en muchas aplicaciones de sistemas de control, pero no como el microprocesador principal en los sistemas de

computadoras personales. Este tiene las mismas funciones que el CPU 8086, añade generación de reloj, dos canales DMA, controlador de interrupciones, tres temporizadores de 16 bits, memoria y periférico lógico de selección de chip y generador de tiempos de espera, todo en un solo chip. El objetivo es reducir el costo y tiempo de ejecución.

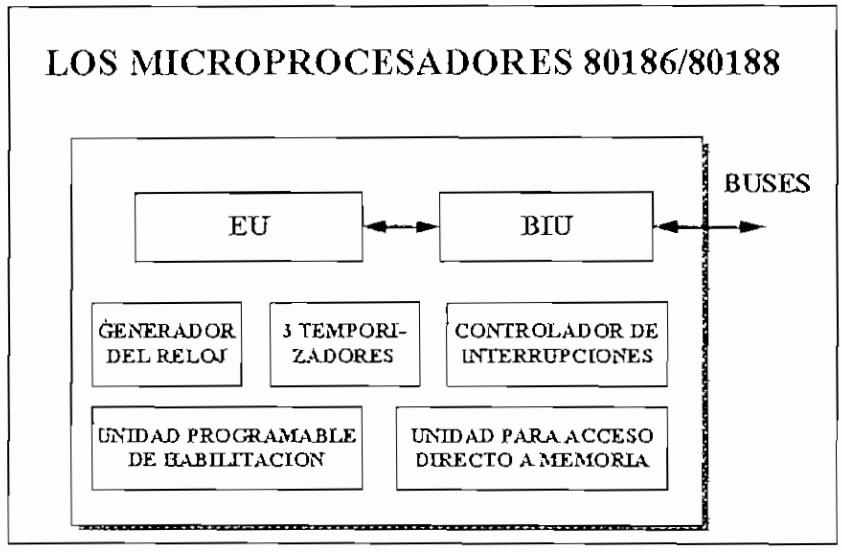


FIGURA 2.2 Diagrama de bloques del microprocesador 80186/80188

2.1.3 Microprocesador 80286

El más reciente microprocesador de 16 bits producido por Intel, es el 80286, una versión mejorada del 8086, que contiene una unidad de administración de memoria y direcciona a una memoria de 16 Mbyte. La velocidad del reloj del 80286 se ha aumentado también a 16 MHz, en las últimas versiones producidas por Intel. La versión básica del 8086 y del 8088 ejecutaba hasta 2.5 MIP (Millones de instrucciones por segundo), en tanto la versión básica del 80286 ejecuta hasta 8 MIP.

Este microprocesador es una extensión evolucionaría del 8086 con capacidades especiales para multiusuario y sistemas multitarea. Tiene en un solo

chip gerenciamiento de memoria y protección de funciones que soportan aislamiento de multitarea, seguridad de datos y programas, cuatro niveles de privilegio dentro de una misma tarea. El gerenciamiento de memoria soporta tanto como 1 G byte de dirección virtual por tarea, mapeada dentro de 16 M bytes de memoria física.

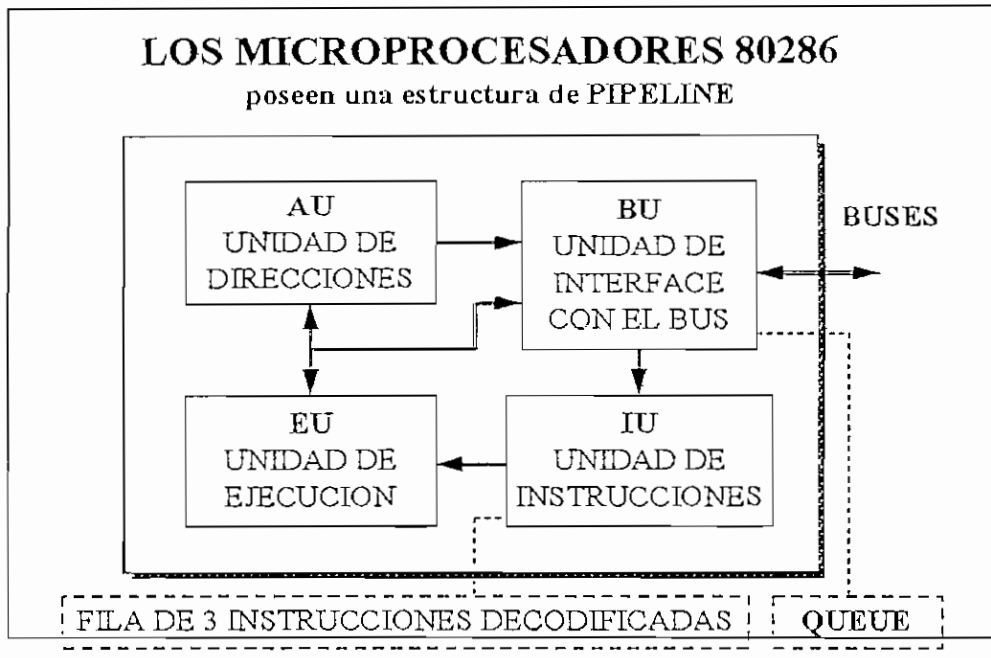


FIGURA 2.3 Diagrama a bloques de la arquitectura del microprocesador 80286.

2.1.4 Microprocesador 80386

El microprocesador 80386 es el primer miembro de 32 bits de la popular familia 8086 de Intel y representa un notable avance frente a sus predecesores. Su arquitectura de 32 bits le permite manejar simultáneamente el doble de datos que los procesadores de 16 bits. Los ordenadores basados en él, son mucho más potentes que los sistemas tipo IBM PC y sus clónicos (basados en el 8088). El 80386 es algo más que un 8088 o un 80286 expandido. Entre sus características claves se encuentra:

- Capacidad de memoria de 4 G bytes. Esto representa 256 veces la capacidad del 80286 y 4926 veces la del 8088.
- Capacidad para ejecutar programas diseñados para el 8086 en sus propios entornos. El 80386 puede conmutar entre el modo de trabajo similar al del 8086 y su propio modo de trabajo. Un ordenador basado en el 80386 puede por tanto, ejecutar los programas más populares diseñados para el MS-DOS, sin perder acceso a las características más avanzadas. Esta es una gran mejora sobre el 80286, que no podía ejecutar programas MS-DOS en su modo nativo (modo protegido). Muchas de las características del 80286, como el bus de direcciones de 16 bits, son por tanto inaccesibles a los usuarios de MS-DOS.

- Capacidad para soportar sistemas que ejecutan varias tareas a la vez (denominados sistemas en *multitasking*). Tanto el 80286 como el 80386 disponen de instrucciones especiales, estructuras de datos y otras características especialmente pensadas para esos sistemas. El multitasking es importante en aplicaciones como diseño asistido por ordenador (CAD), robótica, inteligencia artificial, sistemas militares y aeroespaciales entre otros.
- Posibilidad para direccionar unidades o bloques de memoria de hasta 4 Gb. Estas unidades de datos o programas direccionables independientemente se denominan segmentos. En el 80386, incluso los programas más grandes caben en un único segmento. Esto elimina la necesidad de realizar una programación extra y emplear más tiempo de máquina comprobando lo límites de los segmentos y cambiando de un segmento a otro.
- Soporte para memoria virtual. Es decir, el 80386 puede acceder a más memoria de la realmente presente. El sistema operativo puede mover los

datos y los programas hacia o desde el disco según lo necesite. El programador no tiene por qué gestionar la memoria física de los sistemas, que puede variar entre modelos de ordenador o expandirse con el tiempo.

- Hardware especial incluido en el chip para realizar operaciones de desplazamiento, multiplicación y división, así como de generación de direcciones. Este nuevo hardware permite una ejecución más rápida de las instrucciones.

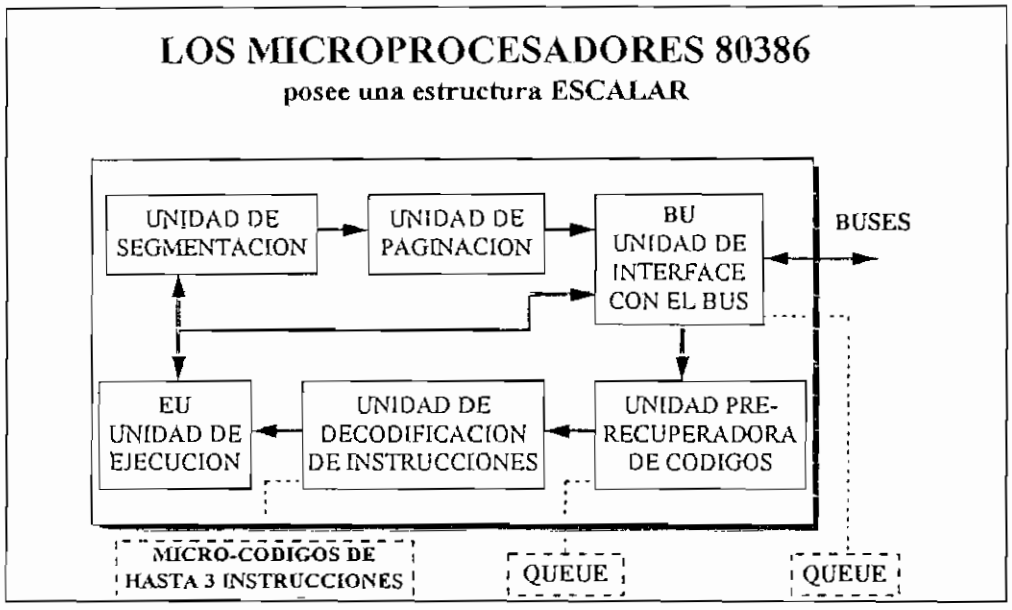


FIGURA 2.5 Diagrama a bloques de la arquitectura del microprocesador 80386

2.1.5 Microprocesador 80486

El 80486 contiene básicamente un 80386 mejorado, un coprocesador aritmético (para la versión DX del 80486) y una memoria caché interna de 8K bytes. El 80386 ejecuta muchas instrucciones en dos ciclos de reloj, mientras que el 80486 ejecuta muchas instrucciones en un solo ciclo de reloj. Estas mejoras, combinadas con un reloj de 66 MHz (80486 DX2) permiten que las instrucciones

se ejecuten a 54 MIP, de acuerdo con Intel Corporation. Esto puede compararse con el 8085, presentado 12 años antes que el 80486, que ejecutaba las instrucciones a una velocidad de alrededor de 0.5 MIP. Estas mejoras en la velocidad continuaron con las versiones más nuevas del microprocesador de 32 bits conforme sea disponible. La generación (Pentium), logra velocidades de 100 MIP.

El microprocesador 80486 es un dispositivo altamente integrado que contiene más de 1'200.000 transistores. Localizados dentro de este poderosos circuito integrado hay una unidad para la administración de la memoria (MMU); un coprocesador numérico completo que es compatible con el 80387; una memoria caché de alta velocidad que contiene 8 Kbytes de espacio; y un microprocesador completo de 32 bits que es compatible hacia arriba con el microprocesador 80386. El 80486 está disponible actualmente en versiones de 25 MHZ, 33 MHZ y 50 MHz. Intel ha mostrado una versión del 80486 de 100 MHz., la que actualmente es muy acogida en el mercado. El 80486 viene como un 80486DX o como un 80486SX. La única diferencia entre estos dispositivos es que el 80486SX no contiene un coprocesador numérico, lo cual reduce el precio. El coprocesador numérico 80487SX está disponible como un componente separado para el microprocesador 80486SX. También están disponibles las versiones de doble reloj como el 80486DX2 (versiones de 50 MHZ y 66MHz.). Las versiones de doble reloj operan internamente a 50 MHz o a 66MHz, sin embargo utilizan una velocidad de canal de 25MHz o 33MHz para facilitar los requerimientos al sistema de la memoria. La versión de doble reloj de 50MHz ejecuta los programas a una velocidad promedio entre las versiones de 33MHz y 50MHz. La versión de doble reloj de 66MHz opera a una velocidad ligeramente mejor que la versión de 50MHz. Observe que usualmente no se requiere de ningún cambio en el sistema para escalar a una versión de doble reloj en la mayoría de las tarjetas madre.

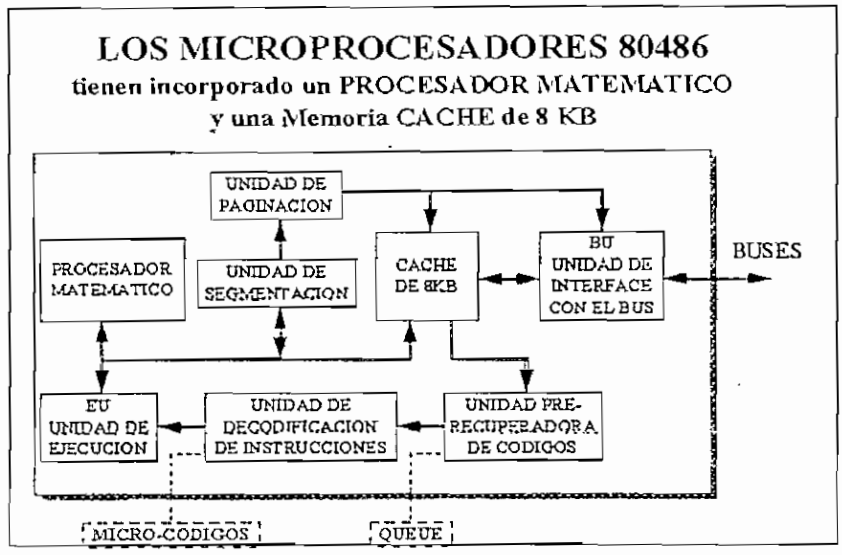


FIGURA 2.6 La estructura interna del microprocesador 80486

En un 80486 aparecen apenas 6 nuevas instrucciones respecto al 80386. Este CPU no ha traído grandes novedades respecto al 80386, la gran alteración es el increíble aumento de velocidad y el alto índice de integración. A continuación se presenta un resumen de los ítems más importantes en el 80486:

- Adición de un coprocesador matemático (80387)
- Adición de un caché de 8k para instrucción y datos
- Modificación del sistema de interfase con el bus, de manera que en un único ciclo de reloj se puede transferir datos en modo ráfaga (16 bytes en 5 ciclos de reloj)
- Soporte para un caché de segundo nivel
- Dramático aumento en la velocidad, cerca de 2 veces la velocidad de un 386 con el mismo reloj. Este aumento de desempeño se debe a:
    - Operación del bus en modo ráfaga con un ciclo de reloj único
    - Caché integrado
    - Filas de instrucciones de 32 bytes

- Reducción significativa del número de ciclos de reloj por instrucción, muchas instrucciones son ejecutadas con un solo ciclo de reloj
- Relojes más rápidos (25, 33, 50mhz)

La habilidad de Intel de integrar una unidad matemática 80387 es la responsable de la gran velocidad del 80486. Para que un 80387 pueda realizar operaciones de "Load/store", son necesarios de 15 a 20 períodos de reloj; para un 486 son necesarios apenas 3 períodos de reloj. Además existe un par de caminos de 32 bits que forman un canal de datos de 64 bits, permitiendo un rápido tráfico de datos.

2.1.6 Microprocesador 80586 (PENTIUM P5)

Los procesadores Pentium de 60 y 66 MHz, presentados en mayo de 1993, más que doblaron la velocidad de un 486DX2/33(66) MHz. Con un alto índice de integración (3.1 millones de transistores) y registros de 32 bits, fue el gran lanzamiento de la industria de las PCs durante el año 1993. Además de los 64 bits del bus interno y del reloj más rápido, muchas otras innovaciones aumentan la velocidad del P5.

- Un producto super escalar y una vía de acceso doble permite que se pueda ejecutar más de una instrucción en un ciclo de reloj.
- Unidad de punto flotante (procesador matemático) con mejor desempeño.
- Un par de "caches" internos más versátiles y eficientes, uno para datos y otro para códigos, ayudan a mantener el procesador trabajando a gran velocidad.

- Una opción de predicción , que determina anticipadamente los desvíos en los saltos.
- Un chequeo interno de paridad permite detectar errores internos de procesamiento.
- Recursos para administración de energía permite obtener un básico consumo de potencia.

El conjunto de instrucciones del P5, sus registros de uso general y de segmentación, los tipos de datos y los modos de acceso a memoria, son similares a la de los modelos 80386/80486. Los modos real y protegido poco han cambiado. Ofrece internamente registros de propósito general de 32 bits, acceso a memoria y operaciones de enteros de ese tamaño. Las opciones de segmentación también siguen los patrones 80386/80486 y usan en modo real registros de segmento de 16 bits y en modo protegido seleccionadores de segmentos de 16 bits. El mecanismo de paginación permite tamaños de página de 4 Kbytes (como en 80386/80486) y también de 4 M bytes.

Las técnicas de vías de acceso múltiples (pipelines), que empezó con los 80386 y 80486, están muy refinadas en el P5, ya que usan una vía de acceso doble. El objetivo de esto es procesar simultáneamente múltiples instrucciones en varias etapas de ejecución, para obtener la ejecución de una instrucción por período de reloj. El resultado final en la P5 fue un producto super escalar que permite ejecutar más de una instrucción por período de reloj.

Las características super escalares del P5 están ligadas a las vías de acceso doble. Los procesadores super escalares permiten la ejecución de más de una instrucción por vez. El P5 tiene dos vías de acceso para enteros, llamadas U y V y automáticamente emparejan las instrucciones para aumentar el número de ejecuciones por período de reloj.

Puede parecer que esas dos vías de acceso para enteros provocan mucha confusión (choques); para evitar esto; existen reglas y restricciones que evitan las colisiones y atrasos. Por ejemplo, si una instrucción genera un dato que será usado por la siguiente instrucción, se dice que entre ellas existe una dependencia de información; otro ejemplo es cuando dos instrucciones operan sobre un mismo registro, entre ellas existe una dependencia de recursos. Cuando se detecta una dependencia, el CPU garantiza que las dos instrucciones serán ejecutadas en secuencia.

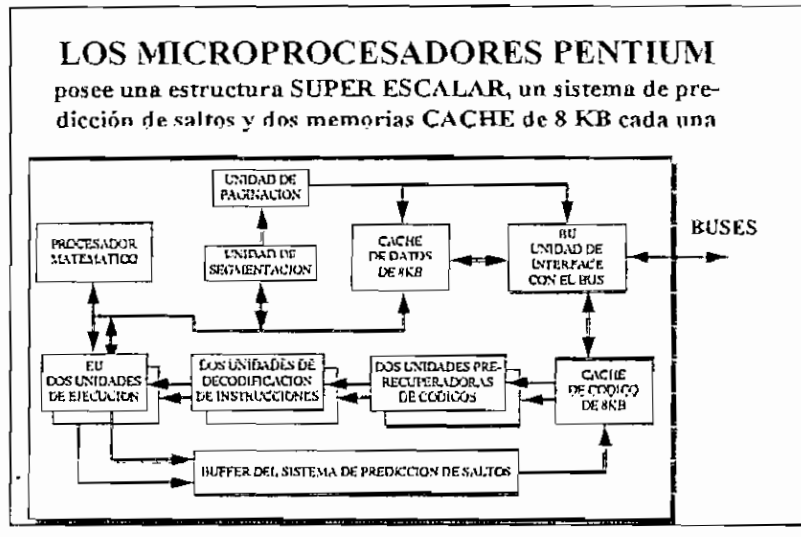


FIGURA 2.7 Diagrama de bloques de P5

Las dos vías U y V no son intercambiables. la vía U ejecuta instrucciones con datos enteros y punto flotante. En cuanto a la vía V sólo puede trabajar con datos enteros.

El orden en que viajan las instrucciones por la doble vía nunca es diferente de la secuencia que se tenía en el programa. Ellas también entran al unísono (juntas) por cada etapa de las vías U y V. Si una instrucción termina la etapa antes que la otra la más adelantada espera por la atrasada. Esto es lo que Intel llama modelo altamente estructurado.

Las instrucciones de desvío, pueden provocar una gran pérdida de velocidad de los procesadores super escalares. Por ejemplo, este puede estar procesando instrucciones en ambas vías de acceso cuando se encuentra en una instrucción de desvío que va a sacarlo de la secuencia de ejecución, esto lleva a que ambas vías y la etapa de pre-fetch deban ser abandonadas por ser una arquitectura "pipeline", cuando se detecta un desvío, todo lo que fue procesado por anticipado es perdido. Además de perderse lo que fue hecho, también hay que esperar la lectura de las nuevas instrucciones. Para evitar esta pérdida de desempeño, fue colocado en el P5 un predictor de desvío. El procesador pre carga las instrucciones de dirección de desvío que fue pronosticado, en un juego de buffers alternativo. Esto da al procesador condiciones para reducir los conflictos y las demoras como resultado se obtiene una mejor utilización de los recursos del procesador.

Un proyecto complejo, como del P5, también tiene innovaciones en otras áreas. La unidad de interface con el bus (BIU), que administra la interface externa del "chip", actúa como un policía de tráfico electrónico, controlando el flujo de información entre los dispositivos externos y la arquitectura interna del P5. Esto es hecho con un bus de datos de 64 bits y un bus de direcciones de 32 bits. Internamente el BIU se conecta con los "caches" de códigos y datos y con la unidad de paginación. Además controla la verificación de datos y direcciones y también las transferencias en la memoria en modo ráfaga.

El P5 también mejora de forma significativa las operaciones de lectura y escritura en modo ráfaga. El flujo máximo es de 528 Mbytes/s, cuando se opera a 66 MHZ. Es más que el doble de lo que se lograba obtener con el 80486DX2/66. Es bueno recordar que el 80486 tiene un bus de datos de 32 bits y que externamente opera a 33 MHZ. La cantidad de bytes que se transfiere en modo ráfaga es igual a la línea interna del "caché": 256 bits (32 bytes).

La unidad matemática, o unidad de punto flotante (FPU), fue totalmente diseñada a partir de los 80386 y 80486 y ahora presenta características de

arquitectura RISC (Reduced Instruction Set Computers-Computadoras con un juego de instrucciones reducido). la unidad cumple con el protocolo IEEE754, usa algoritmos rápidos y aprovecha la arquitectura con vías para lograr una mejora en el rendimiento de 4 a 10 veces. Las vías U y V se conectan cuando acceden a la FPU, posibilitando así un acceso de 64 bits.

El P5 también ofrece un modo de administración de energía (SMM), semejante a los que se usa en los 80486SL . El próximo paso será la modificación del proceso de fabricación para producir el P5 con la tecnología de 3.3 voltios. Esto se obtendrá con una reducción del 30% en el ancho de las líneas, que van, entonces a medir 0,6 micrones.

2.1.7 Microprocesador 80686 (PENTIUM 6)

Se puede afirmar que este "chip" integra las funciones de dos P5, pues posee 4 unidades de números enteros y dos para números reales (punto flotante).

El reloj del P6 deberá ser de 133 MHz y estima unas 200 MIPs, con la posibilidad de un "caché" externo de 512 KB y capacidad para multiprocesamiento superiores a las del P5. Este deberá tener de 4 a 5 millones de transistores y usará líneas de 0.6 micrones (para el P7 se espera alrededor de 25 millones de transistores). Esta gran cantidad de recursos en este "chip" torna clara la preocupación de Intel, pues por primera vez un "chip" suyo va a tener competencia directa. Esta competencia será ofrecida por el POWER PC de la IBM/APPLE/MOTOROLA/ALPHA (DEC).

2.2 Arquitectura y Juego de Instrucciones del 80486

La arquitectura del 80486DX es casi idéntica a la del 80386 mas el coprocesador matemático 80387 y un caché interno de 8 Kbytes. El 80486SX es casi idéntico a un 80386 con un caché de 8 Kbytes. La figura 2.8 muestra la estructura básica interna del microprocesador 80486. Si esto se compara a la

arquitectura del 80386, no se observan diferencias. Las diferencias mas notables entre el 80386 y el 80486, es que casi la mitad de las instrucciones del 80486 se ejecutarán en un período de reloj en vez de los dos períodos que el 80386 requiere para ejecutarles.

Como con el 80386, el 80486 contiene ocho registros de 32 bits para los propósitos generales: EAX, EBX, ECX, EDX, EBP, EDI, ESI y ESP. Estos registros se pueden usar como los registros para la información de 8, 16 o 32 bits o para direccionar una localidad en el sistema de la memoria. Los registros de 16 bits son el mismo conjunto encontrado en el 80286 y son asignados: AX, BX, CX, DX, BP, DI, SI y SP. Los registros de 8 bits son: AH, AL, BH, BL, CH, CL, DH y DL.

Además de los registros de propósito general, el 80486 también contiene los mismos registros de segmento que el 80386 los cuales son: CS, DS, ES, SS, FS y GS. Cada uno tiene 16 bits de ancho, como en todas las versiones anteriores de la familia.

El IP (Apuntador de instrucciones) accesa un programa ubicado dentro del 1 Mbyte de memoria en combinación con CS, o como EIP (Apuntador extendido de instrucciones) para direccionar un programa en cualquier localidad dentro del sistema de memoria de 4 Gbytes. En la operación en modo protegido, los registros de segmento funcionan para mantener selectores como lo hicieron en los microprocesadores 80286 y 80386.

El 80486 también contiene los registros para la tabla de descriptores globales, locales y de interrupciones y una unidad de manejo de memoria como el 80386. Estos registros no están mostrados en la figura 2.8, pero están presentes como en el 80386.

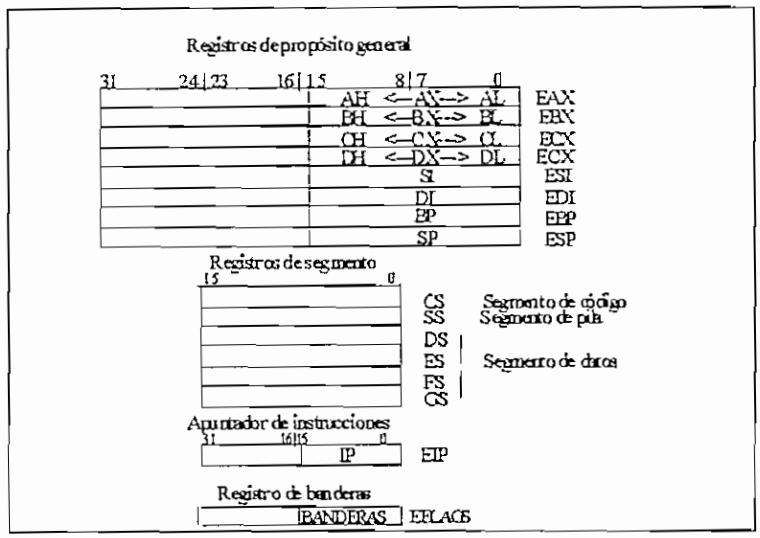


FIGURA 2.8 El modelo de programación interna del 80486

El registro extendido de banderas (EFLAGS). Se muestra en la figura 2.9. Como en los otros miembros de la familia, los bits de la bandera más a la derecha realizan las mismas funciones para compatibilidad. A continuación se da una lista de cada bit de bandera con una descripción de su función.

1. AC (Verificación de Alineación): Nueva para el microprocesador 80486, utilizada para indicar que el microprocesador ha tenido acceso a una palabra en una dirección de paridad impar o una doble palabra almacenada en un límite que no es de doble palabra. Para una ejecución más eficiente de los programas se requiere que la información se almacene en límites de palabra o doble palabra.
2. VM (Modo Virtual) : Se activa este bit mientras que el 80486 se opere en el modo protegido. Este bit siempre será desactivado por una instrucción PUSHF aún si el 80486 está trabajando en el modo virtual.
3. R (Resumen): Utilizado en conjunto con los registros de depuración.

4. NT (Tarea Anidada): Se activa para indicar que el 80486 está realizando una tarea que está anidada dentro de otra tarea.
5. IOPL (Nivel de privilegio de E/S) : Indica el nivel de privilegio máximo actual asignado al sistema de E/S.
6. OF (Sobre Flujo): Indica que el resultado de una operación aritmética con signo ha revasado la capacidad. También se utiliza con la instrucción de multiplicación.
7. DF (Dirección) : Selecciona una operación de auto incremento (DF=0) o autodecremento (DF=1) para las instrucciones de cadenas.
8. IF (Habilitación de interrupciones) : Habilita la terminal INTR si este bit está activo.

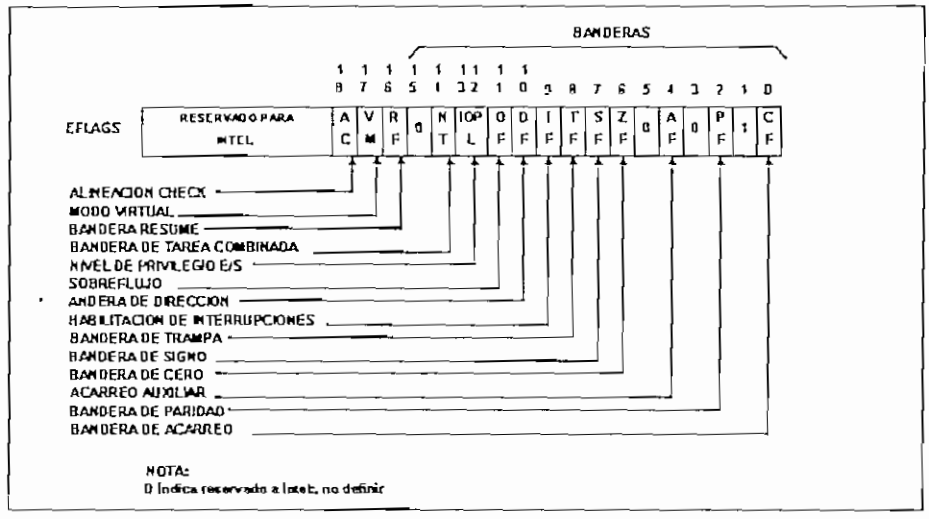


FIGURA 2.9 El registro EFLAGS del 80486

9. TF (Trampa) : Activado para realizar la depuración.
10. SF (Signo) : Indica que el signo del resultado es positivo o negativo.
11. ZF (Cero) : Indica que el resultado de una operación aritmética o lógica es cero (ZF=1) o diferente de cero (ZF=0).

12. AF (Auxiliar) : Utilizado con las instrucciones DAA y DAS para ajustar el resultado de una suma o resta BCD.
13. PF (Paridad) : Indica la paridad del resultado de una operación aritmética o lógica. Si la paridad es impar, PF=0, y si la paridad es par, PF=1.
14. CF (Acarreo) : Muestra si ocurrió un acarreo después de una suma o un préstamo después de una resta.

2.2.1 Sistema de Memoria de 80486.

El sistema de la memoria del 80486 es idéntico al del microprocesador 80386. El 80486 contiene 4G bytes de memoria comenzando en la localidad 00000000H y terminando en la localidad FFFFFFFFH. El cambio principal en el sistema de memoria es interno en el 80486, en la forma de una memoria para caché de 8 Kbytes la cual acelera la ejecución de instrucciones y la adquisición de información. Otra adición es el verificador/generador de paridad incluido en el microprocesador 80486.

Verificador/generador de paridad.- La paridad es frecuentemente usada para determinar si la información se leyó correctamente de una localidad de la memoria. Para facilitar esto, Intel ha incorporado un verificador/generador de paridad interno. La paridad se genera en el 80486 durante cada ciclo de escritura. La paridad será generada como paridad par y un bit de paridad será proporcionado para cada byte de la memoria. Los bits para la verificación de la paridad aparecen en las terminales DP0-DP3, las cuales también son entradas así como salidas de paridad. Estos son normalmente archivados en la memoria durante cada ciclo de escritura y leídos de la memoria durante cada ciclo de lectura.

En una lectura, el microprocesador revisa la paridad y genera un error de revisión de paridad, *si esto ocurre*, en la terminal $\overline{\text{PCHK}}$. Un error de paridad

no causa ningún cambio en el procesamiento a menos que el usuario aplique la señal PCHK a una entrada de interrupción. Las interrupciones son utilizadas frecuentemente para señalar un error de paridad en los sistemas de computadora que utilizan DOS. la figura 2.10 muestra la organización del sistema de la memoria del 80486, que incluye almacenamiento para la paridad. Observe que éste es el mismo del 80386, excepto por el almacenamiento del bit de paridad. Si la paridad no se utiliza, Intel recomienda que las terminales DP0-DP3 sean llevadas a +5.0 V.

Memoria caché. El sistema de la memoria caché almacena los datos utilizados por un programa y también las instrucciones. *El caché* está organizado como un conjunto asociativo de cuatro vías con cada localidad (línea) conteniendo 16 bytes o 4 dobles palabras de información. El caché funciona en el modo de escritura a través del caché. Observe que el caché solo cambia si ocurre una falla. Esto significa que los datos escritos a una localidad de la memoria no están capturados hasta que no se escriban al caché. En algunos casos, mucha de la porción activa de un programa se encuentra completamente dentro de la memoria caché. Esto ocasiona que la ejecución ocurra a una velocidad de 1 ciclo de reloj para muchas de las instrucciones que son utilizadas mas comunmente en un programa. Casi la única manera de que estas instrucciones eficientes sean retrasadas es cuando el microprocesador debe llenar una línea en el caché. Los datos también se almacenan en el caché, pero tienen menos impacto sobre la velocidad de ejecución de un programa, porque la información no se referencia tan repetidamente como lo son muchas porciones de un programa.

El registro de control 0 (CR0) es utilizado para controlar al caché, con dos bits nuevos de control no presentes en el microprocesador 80386. (ver figura 2.11) Los bits CE (deshabilitar caché) y WP (Escritura no a través del Caché) son nuevos para el 80486 y son utilizados para controlar el caché de 8 Kbytes. Si el bit CE es un 1 lógico, todas las operaciones del caché son inhibidas. Esta función

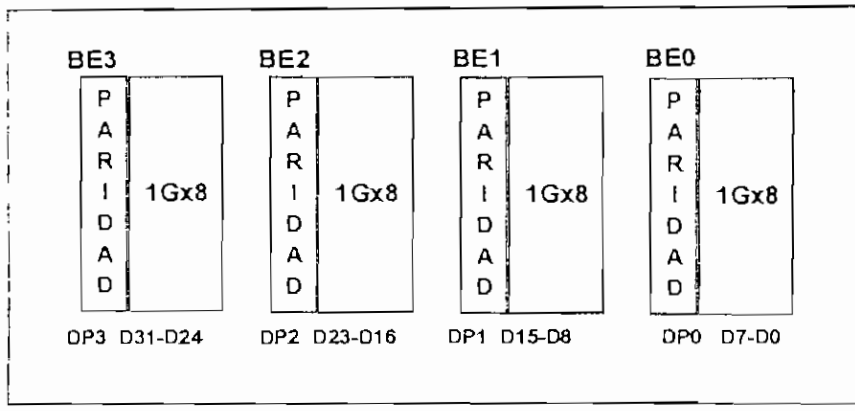


FIGURA 2.10 La organización de la memoria del 80486 mostrando paridad

solo será usada para depuración de programas y normalmente permanece inactivo. El bit WP se usa para inhibir una operación de escritura directa al caché. Como con el CE, el caché a través de escritura es solo inhibido para realizar pruebas. Para el funcionamiento normal de un programa, CE=0 y WP=0.

Debido a que el caché es nuevo en el microprocesador 80486 y se llena usando ciclos de ráfaga no presentes en el 80386, se necesita de algunos detalles para entender los ciclos de llenado del canal. Cuando una línea del canal no está llena, el 80486 deberá adquirir 4 números de 32 bits del sistema de memoria para llenar una línea en el caché. El llenado se logra con un ciclo de ráfaga. El ciclo de ráfaga es una lectura de memoria especial en donde 4 números de 32 bits se recuperan del sistema de la memoria en 5 períodos de reloj. Esto asume que la velocidad de la memoria es suficiente y que no se requiere ningún estado de espera. Si la frecuencia del reloj del 80486 es de 33 Mhz podemos llenar una línea del caché en 167 ns, lo cual es muy eficiente considerando que una operación normal de lectura de la memoria de 32 bits requiere de 2 períodos de reloj.

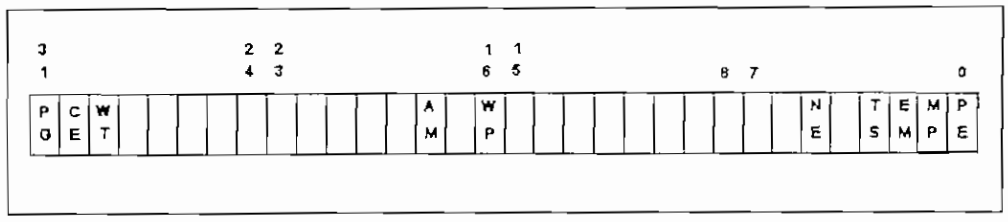


FIGURA 2.11 Registro de control cero (CR0) para el microprocesador 80486

Temporización de la lectura de memoria. La figura 2. 12 muestra la temporización de lectura del 80486 para una operación no ráfaga a la memoria. Observe que 2 períodos de reloj son usados para transferir los datos. El período de reloj T1 proporciona la dirección de la memoria y las señales de control y el período del reloj T2 donde los datos se transfieren entre la memoria y el microprocesador. Observe que $\overline{\text{RDY}}$ se debe convertir en un cero lógico para ocasionar que la información se transfiera y para terminar el ciclo del canal. El tiempo de acceso para un acceso no ráfaga se determinará tomando 2 períodos del reloj menos el tiempo requerido para que la dirección aparezca en el canal de direcciones, menos el tiempo de estabilización para las conecciones del canal de datos. Para la versión de 20 MHz del 80486, dos períodos del reloj requieren de 100 ns menos 28ns para el tiempo de estabilización de la dirección y 3 ns para el tiempo de estabilización de los datos. Esto proporciona un tiempo de acceso no ráfaga de 100 ns - 31 ns o 69 ns. Por su puesto que el tiempo del decodificador y los tiempos de retardo están incluidos, el tiempo de acceso permitido para la memoria es aún menor para una operación sin estados de espera. Además, si una versión de frecuencia más alta del 80486 se usa en un sistema, el tiempo de acceso a la memoria es aún menor.

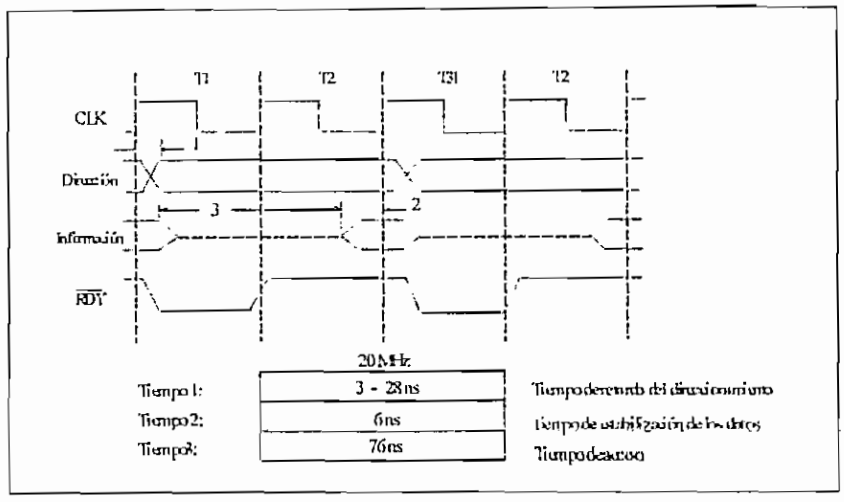


FIGURA 2.12 Temporización de la lectura por no ráfaga para el microprocesador 80486.

La figura 2.13 muestra el diagrama de temporización para llenar un línea caché con 4 números de 32 bits usando una ráfaga. Observe que las direcciones (A31-A4) aparecen durante T1 y permanecen constantes durante el ciclo de ráfaga. Observe también que A2 y A3 cambian durante cada T2, después del primero para direccionar 4 números consecutivos de 32 bits en el sistema de la memoria. Como se menciona , llenar un caché utilizando una ráfaga requiere de solo 5 períodos de reloj (un T1 y 4 T2) para llenar una línea caché con 4 dobles palabras de datos. El tiempo de acceso usando una versión de 20 Mhz de 80486 para la segunda y subsecuentes dobles palabras es 50ns - 28ns - 3ns, o 19ns asumiendo que no hay retardos en el sistema. Para usar las transferencias del modo ráfaga, necesitamos memoria de alta velocidad. Debido a que el mejor tiempo de acceso a la memoria DRAM es de 55ns, estamos obligados a utilizar SRAM para las transferencias mediante ciclos por ráfaga. Aun con estas RAMs estamos tocando los límites de la tecnología con un microprocesador 80486 de 20 Mhz. Observe que $\overline{\text{BRDY}}$ produce transferencia en ráfaga mientras que $\overline{\text{RDY}}$ reconoce una transferencia de memoria normal.

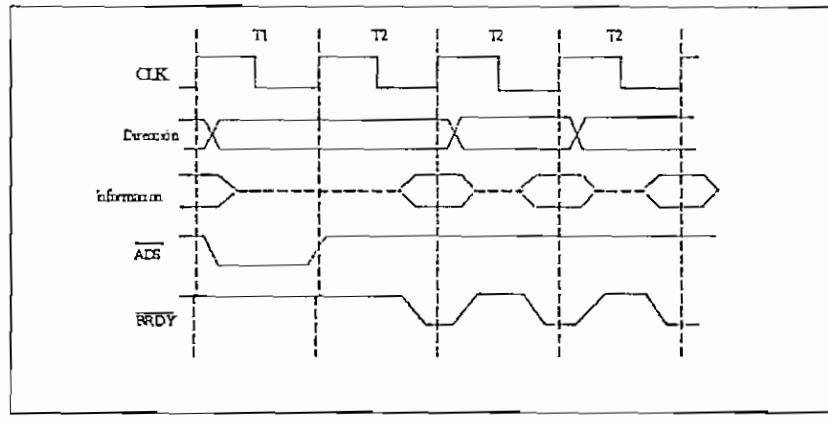


FIGURA 2.13 Un ciclo por ráfaga lee 4 dobles palabras en 5 periodos de reloj en el sistema 80486.

2.2.2 Administrador de Memoria del 80486

El 80486 contiene el mismo sistema de administración de la memoria que el 80386. Esto incluye una unidad de paginación para permitir que cualquier bloque de 4 Kbytes de memoria física sea asignado a cualquier bloque de 4 Kbytes de memoria lineal. Los tipos de descriptores son exactamente los mismos que para el microprocesador 80386. En realidad, la única diferencia entre el sistema administrador de memoria del 80386 y el sistema administrador de memoria del 80486 es la paginación.

El sistema de paginación del sistema 80486 puede deshabilitar el uso de memoria caché para secciones de páginas de memoria transformadas, mientras que el 80386 no puede hacerlo. La figura 2.14 muestra la tabla de entradas al directorio de las tablas de páginas y la entrada a la tabla de páginas.

El PWT controla cómo funciona el caché para una operación de escritura a la memoria caché externa. No controla la escritura en el caché interno. El nivel lógico de este bit se encuentra en la terminal PWT del microprocesador 80486. Por lo tanto, externamente se puede editar la política de escritura a través del caché externo.

El bit PCD controla el caché integrado. Si el PCD es igual a 0, el caché integrado está habilitado para la página actual de la memoria. Observe que los elementos de la tabla de páginas del 80386 colocan un 0 lógico en la posición del bit PCD habilitando el uso caché. Si PCD es igual a 1 se deshabilita el caché integrado.

2.2.3 La Memoria y el Microprocesador

El espacio de direccionamiento de un sistema basado en un microprocesador, se denomina memoria lógica o memoria física. La estructura de la memoria lógica es diferente, en casi todos los casos, que la estructura de la memoria física. La memoria lógica es el sistema de memoria tal como lo ve el programador, mientras que la memoria física es la estructura real en el hardware en el sistema de memoria.

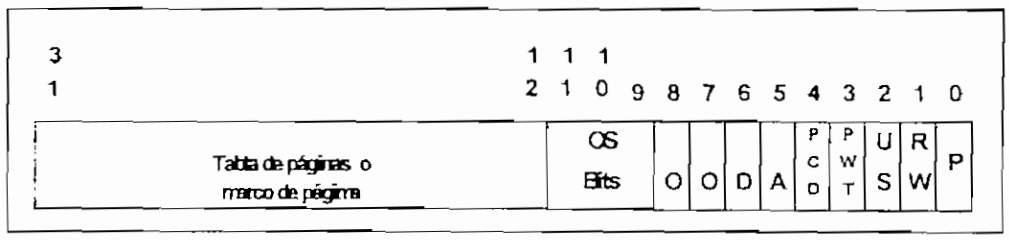


FIGURA 2.14 El directorio de las páginas o elemento de tabla de las páginas para el microprocesador 80486

Memoria Lógica.- El espacio básico de la memoria lógica es el mismo en todos los microprocesadores Intel. La memoria lógica se numera por bytes. En la figura 2.15 se ilustra el mapa de la memoria lógica de todos los miembros de la familia Intel. Se verá que la única diferencia es que algunos miembros contienen más memoria que otros. Además, se debe tener en cuenta que la memoria física puede diferir de la memoria lógica en muchos sistemas.

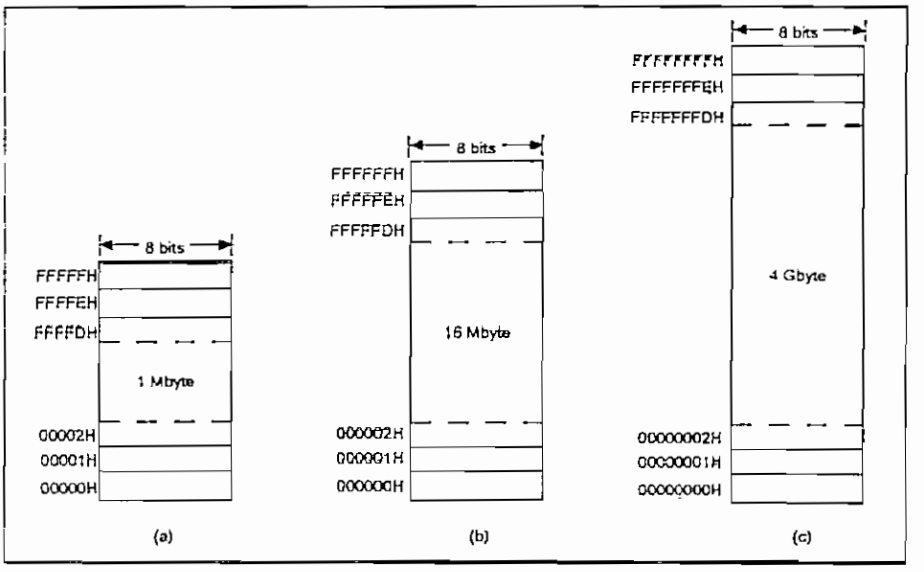


FIGURA 2.15 El mapa de memoria lógica de los microprocesadores: (a) 8086/8088/80186; (b) 80286/80386SX; (c) 80386DX/80486SX y 80486DX

Cuando estos microprocesadores direccionan una palabra de 16 bits en la memoria, se accesan dos bytes consecutivos. Por ejemplo la palabra en la localidad 00122H se almacena en los dos bytes 00122H y 00123H; el byte menos significativo se almacena en la localidad 00122H. Si se accesa a una doble palabra de 32 bits, esta doble palabra la contiene cuatro bytes consecutivos. Por ejemplo, la doble palabra almacenada en la localidad 00120H se almacena en los bytes 00120H, 00121H, 00122H y 00123H; el byte menos significativo se almacena en 00120H y el byte más significativo en 00123H.

Memoria Física.- Las memorias físicas de los miembros de la familia Intel difieren en ancho. La memoria del 8088 es de 8 bits de ancho; las memorias del 8086/80186/80286/80386SX tienen 16 bits de ancho; las memorias del 80386DX y 80486 son de 32 bits de ancho. Para la programación, no hay diferencia en el ancho de la memoria, porque la memoria lógica siempre es de 8 bits de ancho;

pero, como se puede ver en la figura 2.16, hay una diferencia para el diseñador del hardware.

La memoria está organizada en bancos de memoria en todas las versiones del microprocesador, excepto el 8088 que tiene un solo banco de memoria. Un banco de memoria es una sección de 8 bits de ancho. Los microprocesadores de 16 bits tienen dos bancos de memoria para formar una sección de memoria de 16 bits de ancho, a la cual se direcciona por bytes o por palabras. Los microprocesadores de 36 bits tienen 4 bancos de memoria, pero se les direccionan como bytes, palabras o dobles palabras.

La memoria en el computador personal .- Cualquier estudio de la familia Intel requiere entender la estructura de la memoria de la computadora personal. Debido a que la computadora personal original estuvo basada en el microprocesador 8088, se considera que su memoria principal tiene una longitud de 1M byte. Esta memoria principal se llama memoria real. Nunca se toleró una ampliación de la memoria hasta que LIM (Lotus - Intel - Microsoft) crearon un estándar para un sistema de memoria ampliada (EMS).

La memoria ampliada se colocó en un marco de página vacío (64 Kbytes) ubicado entre la memoria de solo lectura (ROM) del BIOS (Sistema básico de entrada/salida) en el sistema. Con este marco de páginas de 64 Kbytes, el estándar LIM permite acceso a un número ilimitado de páginas de memoria ampliada de 64 Kbytes , si bien el acceso a esas páginas es lento. El sistema de memoria ampliada se volvió obsoleto con el advenimiento del microprocesador 80286 y otros más nuevos, aunque todavía está soportado por los antiguos sistemas basados en los 8086 y 8088.

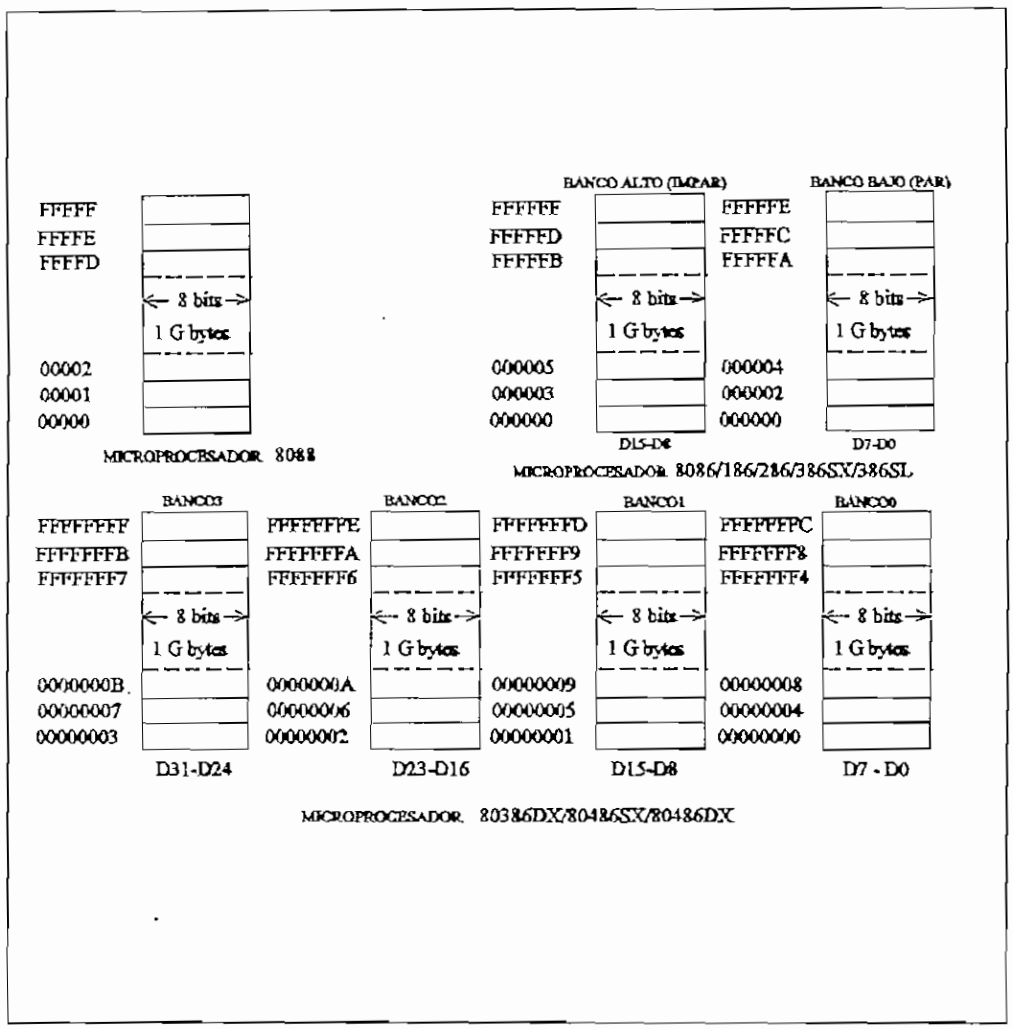


FIGURA 2.16 Los sistemas de memoria física de la familia de microprocesadores 8086-80486.

Con los nuevos 80286, 80386 y 80486 se pueden direccionar memorias por encima del primer byte 1M. Esta memoria adicional, llamada sistema de memoria extendida (XMS) contiene 15 Mbytes adicionales en el sistema de 80286 y 80386SX, y 4095 Mbytes en los sistemas de 80386DX y 80486. El sistema XMS ha sustituido al sistema EMS de las primeras computadoras personales.

En la figura 2.17 se muestra un mapa de memoria de una computadora personal, con la memoria etiquetada por zonas. Los primeros 640 Kbytes del sistema

de memoria en todas las computadoras personales, se llama el área de programa transitorio (TPA). El TPA contienen una memoria RAM (Lectura y escritura) para almacenar las aplicaciones de software, el sistema operativo y diversos programas que controlan los dispositivos de E/S. Después del TPA está almacenada la zona de sistemas que contiene varios BIOS(Sistema básico de entrada/salida) para controlar el sistema, una RAM de video y zonas abiertas que se pueden utilizar para un marco de página de EMS y con

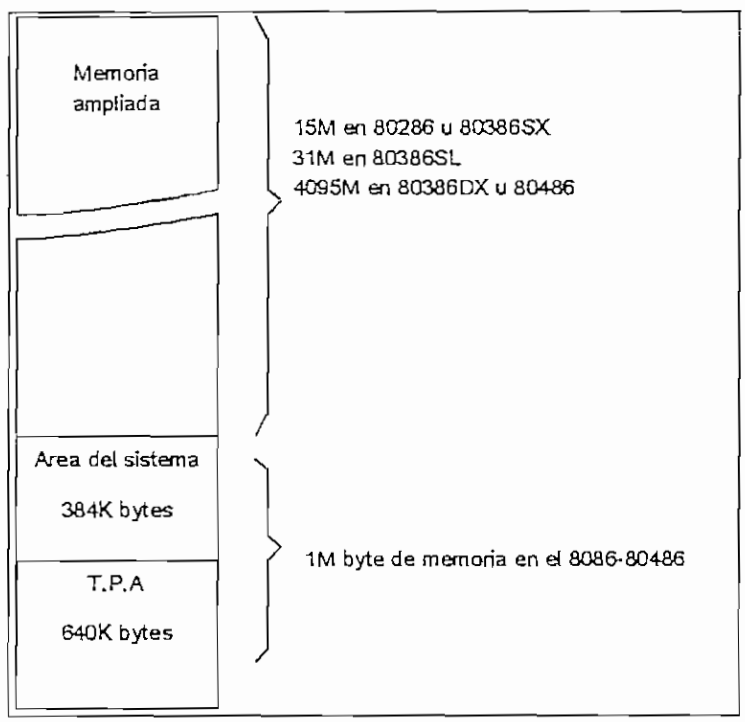


FIGURA 2.17 El mapa de memoria de un sistema de computadora personal

opciones instalables en el sistema de la computadora. Encima de esta zona de memoria de 1M byte, está el sistema de memoria extendida, que tiene sistemas para

discos en cacheo y otros segmentos de datos definidos por el sistema operativo.

2.3 Coprocesador Matemático

La familia Intel de coprocesadores aritméticos incluye los 8087, 80287, 80387SX, 80387DX, 80487SX y el microprocesador 80486DX, que contiene su propio coprocesador aritmético integrado. El conjunto de instrucciones y la programación para estos dispositivos es casi idéntica; la diferencia consiste en que cada coprocesador está diseñado para funcionar con un microprocesador Intel diferente.

La familia del coprocesador, que identificaremos como 80X87, puede multiplicar, dividir, sumar, restar y calcular raíz cuadrada, tangente parcial, arco tangente parcial, y logaritmos. Los tipos de datos incluyen: números enteros con signo de 16, 32 y 64 bits; datos en BCD de 18 dígitos y números con el punto decimal flotante de 32, 64 y 80 bits. Las operaciones realizadas por el 80X87 por lo general se ejecutan aproximadamente 100 veces más rápidas que otras operaciones equivalente escritas por los programas más eficientes.

El 80X87 está diseñado para operar conjuntamente con el microprocesador. Observe que el microprocesador 80486 contiene su propia versión interna del 80387 que es totalemente compatible. En otros miembros de la familia, el coprocesador es un circuito integrado externo que contiene la mayoría de las conexiones del microprocesador. El 80X87 ejecutará 68 instrucciones diferentes con el microprocesador. El microprocesador ejecutará todas las instrucciones normales y el 80X87 ejecutará las instrucciones del coprocesador aritmético. Para ilustrar uno de los coprocesadores, la figura 2.18 muestra el diagrama de base del coprocesador

aritmético 80287, llamado algunas veces un coprocesador numérico.

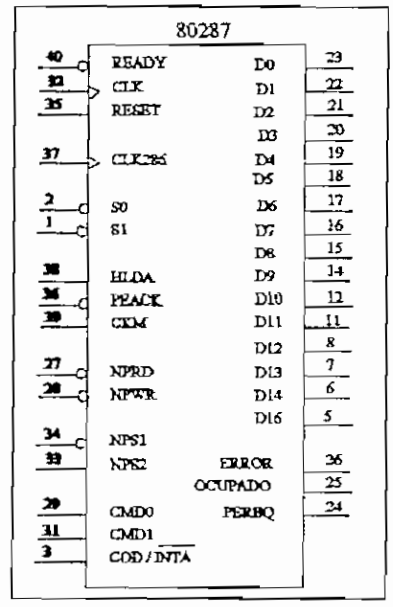


FIGURA 2.18 El coprocesador aritmético 80287

El coprocesador matemático 80287/80387 está diseñado para que funcione en paralelo con el microprocesador 80286 / 80386. El conjunto de instrucciones del 80287/80387 incluye muchas operaciones potentes en punto flotante.

Cuando el 80286 / 80386 encuentra una instrucción en punto flotante, envía el código de operación necesario y direcciones de memoria de operandos al 80287 / 80387. Esto libera al 80286 / 80386 de ejecutar la siguiente instrucción, mientras el 80287/80387 realiza simultáneamente el cálculo numérico.

El 80287/80387 puede hacer peticiones de acceso a memoria a través de un canal de datos dedicado permanente en el 80286/80386. Tales accesos son también comprobados para la violación de reglas de protección y generan excepciones de error apropiadas. Para ciertas operaciones del 80286 / 80386 es necesario forzar la espera

del 80286/80386 hasta que el coprocesador matemático devuelva sus resultados. Esta sincronización con el resultado de vuelta es facilitada utilizando bien la instrucción WAIT o bien la FWAIT del 80286 / 80386.

2.3.1 Pila de Punto Flotante

La pila del 80287 / 80387 consta de 8 elementos de 80 bits (se muestra en la figura 2.19) divididos en campos. Estos campos corresponden al formato de los datos temporalmente reales utilizados en todos los cálculos de la pila del coprocesador.

Los elementos individuales de la pila de punto flotante pueden ser direccionados implícita o explícitamente. Ciertas instrucciones de punto flotante omiten a ciertos elementos de la pila, otras instrucciones permiten obtener un elemento de pila seleccionado por programador.

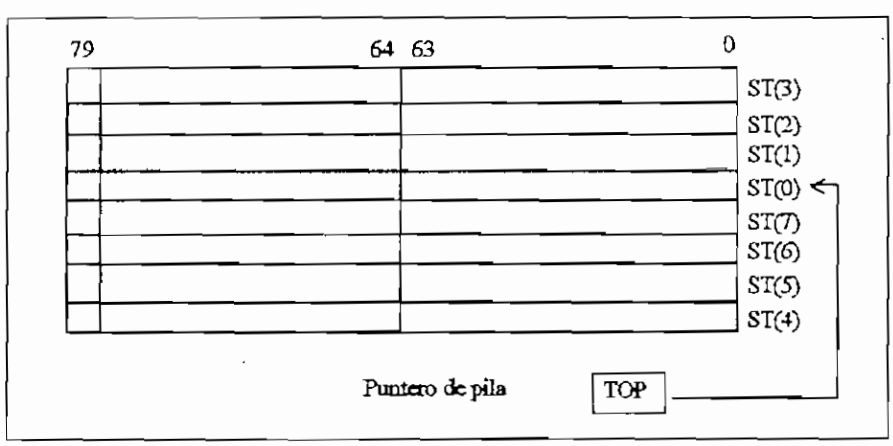


FIGURA 2.19 Pila del 80287/80387

2.3.2 Palabra de status

La palabra de status del 80287/80387 (figura 2-20) refleja la condición global del coprocesador. La palabra de status está dividida en dos campos: Campo bit de indicador de excepción y campo bit de status. La palabra de status puede ser examinada almacenándola en una posición de memoria con una instrucción del 80287/ 80387 y después examinando los bits individuales utilizando el código del 80286/80386.

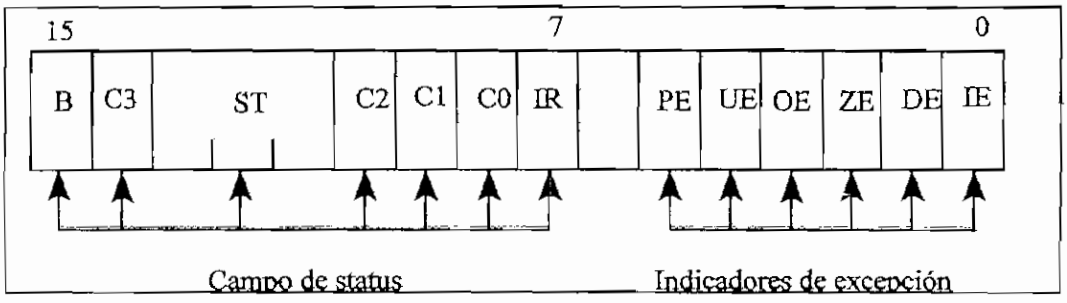


FIGURA 2.20 Palabra de Status del 80287/80387/80487

[B] Este es un campo de un bit, utilizado para indicar si el 80287/80387 está ejecutando actualmente una instrucción o si está desocupado.

[ST] Este campo de 3 bits indica cual de los 8 elementos de la pila está actualmente en la parte superior de la pila.

Valores de ST:
000 Elementos 0 está en parte superior de la pila
001 Elemento 1 está en parte superior de la pila
.
.
.
111 Elemento 7 está en parte superior de la pila

[C3, C2, C1, C0] Estos cuatro campos de un bit se utilizan para obtener

información adicional sobre la parte superior de la pila, estos resultados se usan después para hacer ciertas bifurcaciones condicionales.

[IR] Este campo de petición de interrupción es de un bit que indica que el 80287/80387 quiere interrumpir al 80286 / 80386.

[PE] El indicador de excepción de presición es un indicador de un bit que se inicializa en 1 si el resultado debe ser redondeado para que pueda ser representado en formato de punto flotante.

[UE] Error de subflujo: indica un resultado diferente de cero que es demasiado pequeño para representarlo.

[OE] Error de sobreflujo: indica que un resultado es demasiado grande para ser representado. Si este error se enmascara, el coprocesador genera infinito.

[ZE] Un indicador de un bit, denominado indicador de excepción de Divide por Cero, indica si la división fue por cero y si el dividendo fue un número diferente de cero.

[DE] El indicador de excepción de Operando Desnormalizado (operando no normalizado) es un señalizador de un bit, utilizado para indicar si una instrucción ha intentado operar con un operando desnormalizado .

[IE] El indicador de Excepción de Operación Inválida es de un bit que indica si ha sido realizada o no alguna operación ilegal, como, por ejemplo, una operación en un NAN (no un número), o toman raíz cuadrada de un número negativo.

2.3.3 Palabra de control

La palabra de control del 80287 / 80387 (figura 2-21) contiene las máscaras de excepción y una máscara de habilitación de interrupción y varios bits de control.

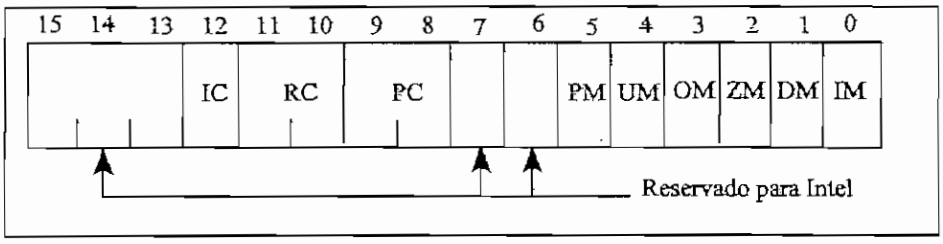


FIGURA 2.21 Palabra de control del 80287/80387/80487

[IC] Control DE infinito es un campo de un bit que especifica cuál de dos tipos de aritmética infinita se está usando. Si IC se reinicializa a 0, entonces se está utilizando la proyectiva; si IC se inicializa a 1, ha sido seleccionadl la afín.

[RC] Control de redondeo es un campo de dos bits que indican cuál de las cuatro direcciones de redondeo ha sido escogida: redondeo no polarizado al más próximo o valor par, redondeo hacia +, redondeo hacia - o redondeo hacia cero.

[PC] Control de precisión es un campo de dos bits que indican cuál de las tres precisiones ha sido escogida: temporalmente real (resultado de 64 bits), real grande (resultado de 53 bits) o real pequeño (resultado de 24 bits).

2.3.4 Máscaras de Excepción

Las máscaras de excepción en la palabra de control se utilizan para indicar qué excepciones deben ser tratadas por las acciones correctivas del 80287/80387 estándar (esto es un ejemplo de una excepción de máscara) y qué excepciones deben hacer que el 80287 / 80387 genere señales de interrupción (una excepción sin máscara). Las máscaras de excepción son:

[PM] Máscara de precisión

[UM] Máscara de underflow

[OM] Máscara de overflow (rebose)

[ZM] Máscara de división para cero

[DM] Máscara de operando desnormalizado

[IM] Máscara de operación inválida

2.3.5 Palabra de Indicación

La palabra de indicación contiene indicadores que describen los contenidos del elemento asociado a la pila. Los valores tabulados se muestran aquí.

TAG(7) TAG(6) TAG(5) TAG(4) TAG(3) TAG(2) TAG(1) TAG(0).

Valores indicadores:

00 = Válido (normal o no normal)

01 = Cero (cierto)

10 = Especial (infinito o no válido)

11 = Vacío

El índice i del TAG(i) no está relacionada con la parte alta. Un programa

normalmente utiliza el campo "alto" de la palabra de estado para determinar qué campo del marcado (i) se refiere a la parte alta lógica de la pila.

2.3.6 Tipos de datos

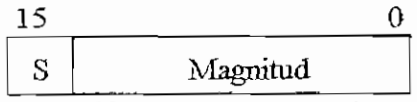
El 80287 / 80387 soporta 7 tipos de datos diferente. Estos valores numéricos son accedidos utilizando todos los modos de direccionamiento estándares soportados por el 80286 / 80386.

Para los siete formatos de datos, el signo del número se almacena siempre en el dígito más significativo, o dígito de más a la izquierda del campo.

2.3.6.1 Enteros binarios

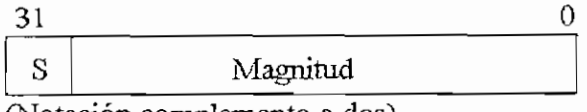
Los tres formatos de datos enteros binarios son idénticos, con la excepción de la longitud del campo, que gobierna el rango del número. El bit más a la izquierda es interpretado como el signo del número. Los número negativos están representados en la notación de complemento a dos. El cero se representa con un signo positivo. El entero palabra del 80287/80387 es idéntico al formato de dato entero con signo de 16 bits del 80286 / 80386.

* Entero palabra (rango: - 32768 <= X <= + 32767)



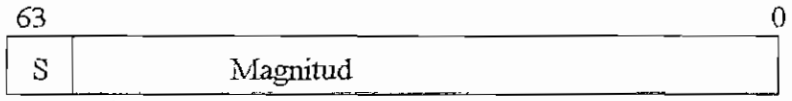
(Notación complemento a dos)

* Entero corto (rango: $- 2 \times 10^9 <= X <= + 2 \times 10^9$)



(Notación complemento a dos)

* Entero largo (rango: - $9x10^{18}$ <= X <= + $9x10^{18}$)

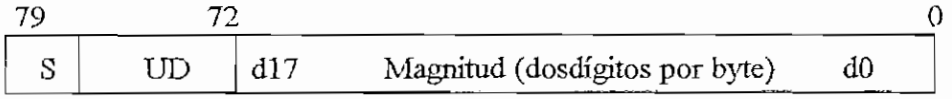


(Notación complemento a dos)

2.3.6.2 Notación decimal empaquetada

La notación decimal empaquetada se utiliza para almacenar enteros decimales, almacenando dos dígitos decimales o paquete, en cada byte. El bit del signo determina si el número es positivo o negativo.. Los dígitos deben estar en el rango de 0H - 9H, inclusive.

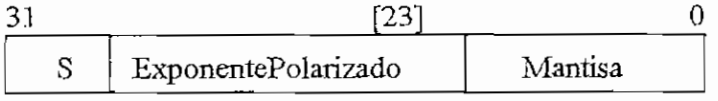
Decimal empaquetado (rango - 99 ... 99 <= X <= + 99 ... 99 (18 dígitos))



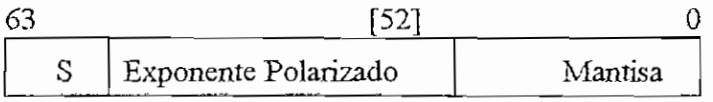
2.3.6.3 Formato de real corto, real largo y real temporal

Tanto los formatos de datos real grande y real pequeño existen solamente en memoria, cuando un número almacenado en uno de estos formatos se carga en la pila de punto flotante, este es automáticamente convertido al formato real temporal.

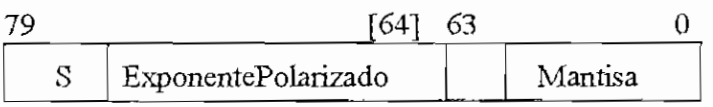
Real corto (rango: 0, 1.2×10^{-38} <= X <= 3.4×10^{38})



Real largo (rango: 0, $2.3 \times 10^{-308} <= X <= 1.7 \times 10^{308}$)



Real Temporal (rango: 0, $3.4 \times 10^{-4932} <= X <= 1.1 \times 10^{4932}$)



Valores de exponentes polarizados (normalizados):

Real corto:	127 (7FH)
Real Largo:	1023 (3FFH)
Real Temporal:	16383 (3FFFH)

2.3.6.4 Valores especiales

El coprocesador matemático 80287/ 80387 incluye varios valores especiales para incrementar la flexibilidad de los cálculos numérico. Estos valores especiales incluyen.

- No normales ("Unnormals")[1]
- Denormales ("Denormals")[2]
- Valores indefinidos
- Valores NAN (Not - A- Number)

[1] Números que se presentan en el formato real temporal, y pueden reconocerse por que el bit 63 es cero
[2] Números que tienen a cero el campo de exponente y la mantisa es distinta de cero.

- Representaciones infinitas + y -

- Cero con signo

Es importante dar a conocer el formato de la memoria cuando los registros del 8087 se almacenan mediante la instrucción FSAVE como se lo muestra en la figura 2.22, también en la figura 2.23 se presenta algo importante como es la estructura interna del 80X87.

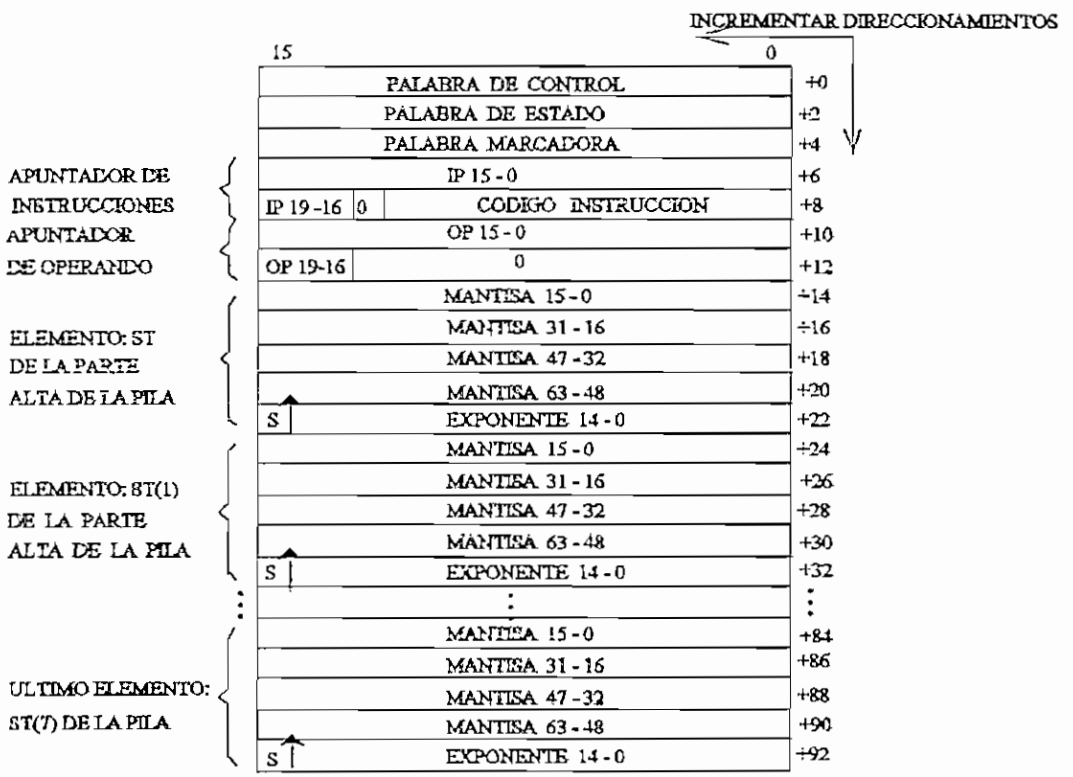


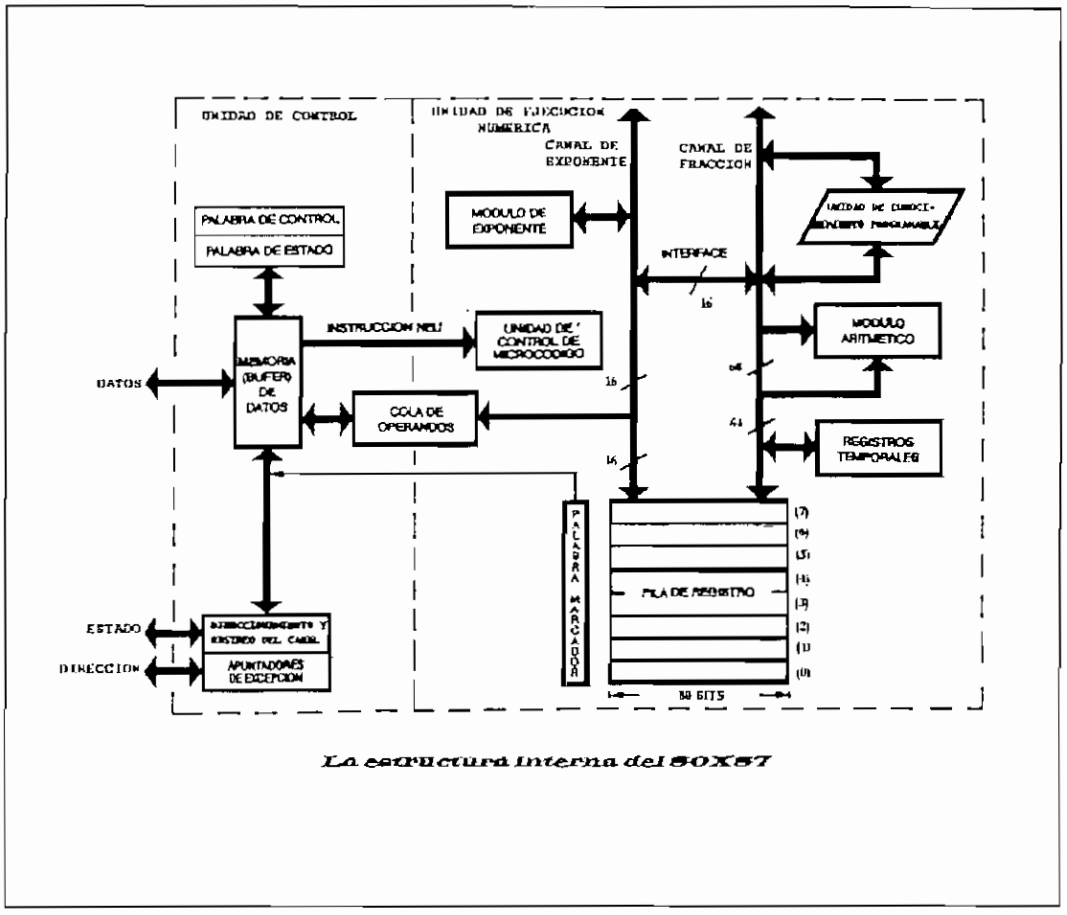
FIGURA 2.22 Formato de la memoria cuando los registros del 8087 se almacenan mediante la instrucción FSAVE

2.3.7 Aritmética de los Números Reales y los Coprocesadores Intel

A la pila se puede introducir números en formato entero e imprimirlos en formato entero, esto depende de la forma en que los números reales son codificados por el coprocesador. Por ejemplo considérese el listado siguiente:

```
.8087
STACK     SEGMENT   PARA STACK
          DB        64 DUP ('MYSTACK')
STACK     ENDS
MYDATA    SEGMENT   PARA 'DATA'
RADIUS    DQ   8.567
AREA DQ   ?
MYDATA    ENDS
```

Si el directivo .8087 no estuviese incluido, el número real 8.567 de RADIUS sería codificado en formato Microsoft -el formato usado por los lenguajes que no soportan el coprocesador- . Con el directivo dado .8087, 8.567 será codificado en el formato de punto flotante del IEEE. Los coprocesadores Intel requieren formatoIEEE. Para desplazar y almacenar números reales utilizando el procesador 80286/80386, los números deben aparecer como enteros. Estos números son codificados automáticamente en el formato especificado, cuando se ensambla el programa. Desgraciadamente, no hay proceso de decodificación para números que son devueltos por el coprocesador , porque no es un proceso comparable a ensamblar cuando se ejecuta un programa . Eso significa que el resultado real tendrá que ser decodificado por el usuario para que tenga sentido.



La estructura interna del 80X87

FIGURA 2.23 La estructura interna del 80X87

2.3.8 Formatos de Números Reales IEEE

Hay una serie de formas en las cuales un número real puede ser codificado y formado utilizando dígitos hexadecimales. Afortunadamente el IEEE pone un estándar de punto flotante, el cual los ingenieros de Intel modelaron cuidadosamente para los coprocesadores 80287/80387. Aunque los coprocesadores mantienen datos en formato real temporal (80 bits), son los formatos del real pequeño (32 bits) y del real grande (64 bits) los que más interesan.

El número real corto es devuelto a memoria como un número codificado de 32 bits. El MSB es utilizado para decodificar el signo, los siguientes 8 bits el exponente y los restantes 23 bits la mantisa. Para el número real largo se devuelve un número codificado de 64 bits. El MSB se utiliza para decodificar el signo. Los siguientes 11 bits el exponente y los restantes 52 bits la mantisa. La ecuación general para la conversión es bastante simple:

$$\text{número real} = (-1)^{\text{signo}} \times (\text{mantisa}) \times 2^{\text{exponente}}$$

Por ejemplo, supongamos que se ejecutó un programa, y que se tomó la siguiente información del segmento de datos:

82 ED FC 79 51 D2 6C 40

Organizando los bits, en el orden adecuado, se obtiene el siguiente número real codificado hexadecimal:

406CD25179FCED82

Al convertir este número a formato binario se obtiene lo siguiente:

0100000001101100110100100101000101111001111111001110110110000010

Al separar el número de acuerdo con signo exponente y significado se obtiene:

0 10000000110 1100110100100101000101111001111111001110110110000010

Usando la ecuación general, se obtiene el siguiente resultado: El MSB=0, así -1 elevado a la potencia 0 es 1 y el signo es positivo. El número 10000000110 convertido al formato decimal es 1030. Para números reales largos, el exponente está polarizado por 1023. El exponente real es entonces 7. La mantisa es un número en base 2, formada por restantes dígitos, con el primer dígito antes del punto decimal asumido a 1:

1.1100110100100101000101111001111111001110110110000010

Convertir este número binario a decimal involucra el trabajar con pesos fraccionarios. El primer lugar a la derecha del punto decimal representa 1/2, el segundo 1/4, el tercero 1/8 y así sucesivamente. En cada posición donde hay un 1 usted añadirá el peso de ese dígito.

1.0
.5
.25
.03125
.015625
.00390625
.00048828125
.00006103515625

1.80133056640625

Esto sólo cuenta para los 8 primeros unos (1).

Ahora volviendo a la fórmula:

Capítulo 3

HERRAMIENTAS DE PROGRAMACION DEL 80486

3.1 Interrupciones

Las interrupciones son llamadas generadas en el exterior del microprocesador por una señal de hardware (Interrupciones Hardware) o derivada en el interior por una instrucción (Interrupciones Software o Excepciones), y permiten utilizar más eficientemente los recursos del sistema. En ambos casos se interrumpe el programa por que se llamará a una *Rutina para servicio de interrupción* o un *Controlador de interrupción.*

Vectores de Interrupción.- Un vector de interrupción es un número de 4 bytes almacenado en los primeros 1024 bytes de la memoria (000000H-0003FFH) cuando el microprocesador trabaja en el modo real. En el modo protegido se tiene en cambio una *Tabla de Descriptores de Interrupción* que consta de descriptores de 8 Bytes por cada interrupción. Se tienen 256 vectores de interrupción posibles, cada uno de estos vectores contiene la dirección inicial de un procedimiento (conocido como rutina o procedimiento de servicio de interrupción) ordenado de la siguiente forma:

2 bytes superiores : Desplazamiento

2 bytes inferiores : Segmento

La Figura 3.1 muestra la tabla de los vectores de interrupción en modo Real, la dirección del vector de interrupción correspondiente a la interrupción n es 4(n-1), puesto que cada vector de interrupción ocupa 4 bytes.

Los cinco primeros vectores de interrupción son idénticos en todos los microprocesadores Intel. Hay otros vectores cuya función se ha ido implementando en las nuevas versiones de microprocesadores. Intel reserva los primeros 32 vectores de interrupción para emplearlos en sus diversos microprocesadores, de éstos solo los vectores 1 a 6, 7, 9, 16 y 17 funcionan en el modo real y protegido, los vectores restantes sólo funcionan en el modo protegido.

Vector	Dirección	Microprocesador	Función	Instrucción que puede ser causante
0	0h - 3h	8086-80486	Error al dividir	DIV, IDIV
1	4h - 7h	8086-80486	Ejecución paso a paso	Cualquier instrucción
2	8h - Bh	8086-80486	NMI (Interrupción de Hardware)	INT 2 o NMI
3	Ch - Fh	8086-80486	Punto de ruptura	INT
4	10h - 13h	8086-80486	Interrupción de sobreflujo	INTO
5	14h - 17h	80286-	Interrupción de BOUND	BOUND
6	18h - 1Bh	80286-	Código inválido	Cualquier instrucción Ilegal
7	1Ch - 1Fh	80286-	Interrupción de emulación del coprocesador	ESC, WAIT
8	20h - 23h	80386-80486	Dohle error	Cualquier instrucción que puede generar una excepción.
9	24h - 27h	80386	Desbordamiento de segmento de coprocesador	ESC
10	28h - 2Bh	80386-	Segmento de estado de tarea inválido	JMP,CALL,IRET,INT
11	2Ch - 2Fh	80386-80486	No hay segmento	Instruccioues para el registro de segmento
12	30h - 33h	80386-	Falla de Pila	Referencias de Pila
13	34h - 37h	80386-	Falla general de la protección	Cualquier referencia de la memoria
14	38h - 3Bh	80386-80486	Falla de página	Cualquier acceso de la memoria o recuperación de código
15	3Ch - 3Fh		Reservado	
16	40h - 43h	80286-	Error de punto decimal flotante	ESC, WAIT
17	44h - 47h	80486SX	Interrupción de verificacióo de alineación	
18-31	48h - 7Fh	8086-80486	Reservadn	
31-255	80h - 3FFh	80386-80486	Interrupciones definidas por el usuario	

*** Todas las interrupciones pueden ser llamadas por la instrucción INT n (n número de interrupción)**

FIGURA. 3.1 Tabla de los Vectores para Interrupción

Cambio de un vector de Interrupción.- Debido a que los vectores de interrupción residen en RAM, se puede modificar su valor para que apunte a una rutina definida por el usuario. Antes de cambiar un vector de interrupción se debe guardar su contenido inicial para restaurarlo antes de terminar el programa. La forma más adecuada de realizar el cambio es mediante la Interrupción 21h del DOS (funciones 25h y 35h), para usar ésta interrupción, CX debe contener el desplazamiento y DS el segmento de la nueva rutina, AH el valor 25h y, AL el número de interrupción a cambiar :

```
PUSH      DS              ;salvar DS
MOV       AL,n            ;AL= número de la interrupción
MOV       AH,35h          ;función: recuperar vector de interrupción
INT       21h                   ;Interrupción 21h
MOV       VIEJOSEG,DS     ;guardar segmento originalen memoria
MOV       VIEJODESP,DX    ;guardar desplazamiento original
                          ;en memoria
MOV     DS,SEG RUTINA           ;mover segmento nuevo
MOV     DX,OFFSET RUTINA        ;mover desplazamiento nuevo
MOV     AL,n                    ;AL= número de la interrupción
MOV     AH,25h                  ;función :poner vector de
                                ; interrupción
INT     21h                     ;Interrupción 21h
POP     DS                      ;restaurar DS
```

Generalmente se definen rutinas nuevas de una interrupción, para procesos que son muy utilizados y cuyo manejo se facilite mediante una interrupción *software*. Supongamos que se requiera constantemente mover a ES el valor de DS y poner en ES un valor de 0000h. La rutina para este proceso sería la siguiente :

```
NUEVARUTINA PROC FAR
            PUSH AX           ;Guardar los registros afectados
            MOV  AX, ES       ;AX = ES
```

```
                MOV  DS,AX      ;DS = AX
                XOR  AX,AX      ;AX = 0
                MOV  ES,AX      ;ES = 0
                POP  AX         ;Restaurar el valor de AX
                IRET            ;Retorno de la interrupción
NUEVARUTINA     ENDP
```

Funcionamiento de una Interrupción.- Cuando el microprocesador concluye de ejecutar la instrucción en curso, para determinar si hay una interrupción pasa a comprobar : (1) alguna instrucción en ejecución, (2) trampa (int 01h), (3) NMI, (4) sobreflujo del segmento del coprocesador, (5) INTR, (6) la instrucción INT. Si está presente una condición de interrupción, el microprocesador actúa de la siguiente forma :

1. Se salva el contenido del registro de banderas en la pila.
2. Se desactivan las banderas de interrupción (IF) y de trampa (TF). Esto deshabilita la terminal INTR (fuente de interrupción de hardware) y también la característica de trampa.
3. Se salva el contenido del registro (CS) de segmento de código hacia la pila.
4. Se salva el contenido del apuntador de instrucción (IP) en la pila.
5. Se recupera el contenido del vector de la interrupción y se coloca en IP y CS, de modo que la siguiente instrucción se ejecute en el procedimiento de servicio de la interrupción direccionado por el vector.

Los valores salvados en la pila se recuperan cuando el microprocesador encuentra la instrucción IRET, que indica retorno de la rutina de servicio a la interrupción, de tal forma que las banderas vuelven a su valor original. El valor de la dirección de retorno (IP y CS) guardados en la pila durante la interrupción, a veces apunta hacia la siguiente instrucción en el programa y, a veces, apunta hacia la instrucción o punto en el programa en que ocurrió la interrupción (interrupciones 0,5,6,7,8,10,11,12 y 13) esto permite que el procedimiento de servicio de interrupción vuelva a intentar probar la instrucción en ciertos casos de error.

Dado que la bandera de acarreo (CF) se usa muy poco, excepto con sumas y restas de palabras múltiples, se la emplea para indicar error al retornar de un procedimiento o rutina de interrupción. Al retornar de la ejecución de la interrupción, si CF = 1, ha ocurrido un error y si CF = 0 no ha ocurrido un error. En casi todos las interrupciones de DOS y de BIOS se utiliza la bandera de acarreo para indicar condiciones de error.

3.2 Interrupciones Hardware o Interrupciones de Periféricos

El microprocesador tiene dos terminales de entrada para interrupciones de periféricos y una salida asociada a la entrada INTR tal como se muestra en la Figura 3.2.

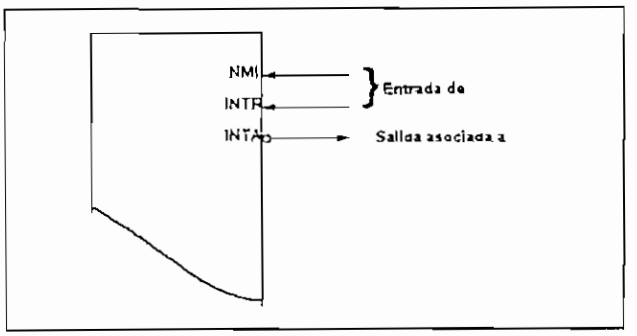


FIGURA. 3.2 Terminales de Interrupción Hardware del 80486

NMI.- Produce una interrupción tipo 2 por que se decodifica en el interior. Es disparada por flanco ya que solicita una interrupción con el flanco positivo (transición de 0 a 1) , debe permanecer en 1 lógico hasta que la reconozca el microprocesador. Y antes de la transición positiva debe estar en 0 lógico durante al menos 2 períodos de reloj. Es usada para errores de paridad y otros problemas graves, como las interrupciones de energía

INTR e $\overline{INTA}$.- La entrada de interrupción (INTR) es sensible al nivel, se la debe mantener en 1 lógico hasta que se la reconozca. Se la debe desactivar en la rutina de servicio a la interrupción para evitar que ocurra otra llamada mientras se está atendiendo a la anterior. Esta entrada se deshabilita en forma automática una vez que la ha aceptado el microprocesador, y se vuelve a habilitar con la instrucción IRET al final del procedimiento del servicio a la interrupción. En el 80386 y 80486 en modo protegido se usa IRETD. El microprocesador responde a la entrada INTR pulsando dos veces la salida $\overline{INTA}$ para recibir el número de vector de la interrupción en la conexión D7-D0 en el canal de datos. El vector de interrupción puede estar entre 0 y 255 (0 y FFh) pero se recomienda usar desde la interrupción

20h en adelante. Una vez que el microprocesador tiene el número de vector de interrupción pasa a ejecutar la rutina de servicio correspondiente.

Expansión de la estructura de Interrupción .- Debido a que todos los dispositivos externos que deseen enviar datos al microprocesador (teclado, impresora, puertos seriales, etc) solicitan ser atendidos mediante interrupciones, se hace necesario implementar un sistema que permita a partir de INTR ampliar el número de interrupciones externas. En la Figura 3.3 se muestra la idea general de expandir las interrupciones. El Circuito Lógico debe realizar las siguientes acciones al producirse una petición de interrupción Iri :

- Debe activar adecuadamente el terminal INTR para que el microprocesador acepte el pedido de interrupción.
- Poner en el bus de datos (D0-D7) el valor del vector de interrupción asociado a la interrupción externa IRi, hasta que el microprocesador envié la señal en $\overline{\text{INTA}}$ aceptando el número de vector de interrupción.
- Se debe implementar un sistema de prioridades para las interrupciones IRi, en los casos en que se produzcan varias de ellas a la vez.

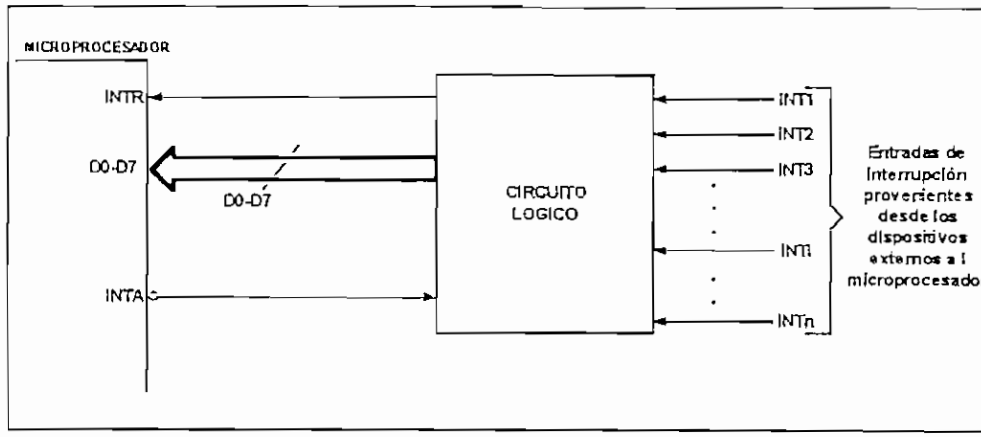


FIGURA 3.3 Expansión de las interrupciones

3.3 Interrupciones Software o Excepciones

En esta parte se estudiarán las interrupciones de acuerdo a la función que realiza su respectiva rutina de servicio asociada, en la Figura 3.4 se presenta una tabla con los tipos de interrupción bajo este concepto.

Número (Decimal)	Número (Hexadec)	Tipo	Dirección Vectores
0 - 31	0 - 1Fh	BIOS	0 - 7Fh
32 - 63	20h - 3Fh	DOS	80h - FFh
64 - 255	40h - 7Fh	Usuario	100h - 1FFh

Fig. 3.4 Tabla de Tipos de interrupciones de acuerdo a su uso

3.3.1 Interrupciones BIOS (Basic Input/Output System)

En la Figura 3.5, se listan todas las interrupciones del BIOS y su uso. De todas las interrupciones BIOS se recomienda usar únicamente la 10H (servicios de vídeo cuyas funciones se presentan en detalle en el apéndice E) ya que las demás funciones se encuentran en las interrupciones DOS y ofrecen mayor facilidad. A continuación se presenta un resumen de las interrupciones BIOS:

Puntos de Entrada al BIOS	Uso
10h	Servicios de Vídeo
11h	Obtener Configuración del Equipo
12h	Obtener el tamaño de la memoria convencional
13h	Servicios Directos de Disco
12h	Servicios del Puerto Serial
15h	Servicios Misceláneos del Sistema
16h	Servicios de Teclado
17h	Servicios del Puerto Paralelo
18h	ROM BASIC
19h	Reiniciar el Sistema
1Ah	Servicios de Reloj

Servicios Directos del Sistema	
Interrupciones Internas del Microprocesador	
00h	División por cero
01h	Ejecución paso a paso
02h	Interrupción no enmascarable (NMI)
03h	Punto de Ruptura (INT 3)
04h	Error de sobreflujo Aritmético
05h	Imprimir Pantalla y BOUND
06h	Error de Instrucción Ilegal
07h	Extensión de Procesador (Coprocesador) no disponible
Rutinas de Interrupciones Hardware del controlador de Interrupciones (8259A)	
08h (IRQ0)	Pulso de Reloj
09h (IRQ1)	I/O del Teclado
0Ah (IRQ2)	Cascada al PIC (8259A) secundario
0Bh (IRQ3)	Puerto secundario de comunicaciones (COM2)/Red
0Ch (IRQ4)	Puerto primario de comunicaciones (COM1)/Ratón
0Dh (IRQ5)	LPT2/Red/Sonido/CD/Sobreflujo de Segmento
0Eh (IRQ6)	I/O del Controlador de Disco
0Fh (IRQ7)	Puerto Paralelo (LPT1)/Sonido/CDROM
Rutinas de Interrupciones Hardware del Controlador secundario (8259A)	
70h (IRQ8)	Alarma de Reloj en Tiempo Real
71h (IRQ9)	EGA/VGA (RQ2 redirigida)
72h (IRQ10)	Reservado/Red
73h (IRQ11)	Reservado/Sonido
74h (IRQ12)	Mouse (COM2)
74h (IRQ13)	NMI (Coprocesador)
76h (IRQ14)	Primer Disco Duro
77h (IRQ15)	CDROM/Segundo Disco Duro, etc
Rutinas de Usuario	
1Bh	Manejador de CTRL+BREAK
1Ch	Servicios del Temporizador de Usuario
Parámetros del BIOS	
1Dh	Apunta a la Tabla de Parámetros de Vídeo
1Eh	Apunta a la tabla de Parámetros del Controlador de disco
1Fh	Apunta a la Tabla de Patrones de caracteres gráficos

FIGURA 3.5 Tabla de las Interrupciones BIOS y su Uso.

3.3.2 Interrupciones DOS

En la tabla 3.6 se presenta como están distribuidas las interrupciones para el sistema operativo MS-DOS, la mayoría de ellas todavía no tienen asignada una aplicación y se hallan reservadas para ampliaciones futuras.

Vector	Uso
20h	Terminar Programa
21h	Petición de Función del DOS
22h	Manejador de terminación de programa
23h	Manejador de Ctrl-C
24h	Manejado de error crítico
25h	Leer sector de Disco (Ignora la estructura lógica)
26h	Escribe sector de disco (Ignora la estructura lógica)
27h	Terminar programa y dejarlo Residente
28h	DOS en espera
29h - 2Eh	No se usa (Reservada)
2Fh	Interrupción Multiplex
30h - 3Fh	No se usa (Reservada)

Fig 3.6 Tabla de distribución de las interrupciones del DOS

La interrupción 21H es la mayormente empleada, permite hacer uso de las diferentes rutinas del DOS como : acceso a teclado, pantalla, memoria, manejo de archivos, etc. Con cada versión de DOS se han incrementado funciones, algunas tienen el mismo fin que otras ya existentes, pero su uso es más simple. En el Apéndice F se presenta un resumen de las funciones de la interrupción 21H.

3.4 El Programa Ensamblador (MASM) y Enlazador (LINK)

El objetivo en este literal es aprender a crear, enzamblar y enlazar un programa en lenguaje ensamblador.

Un ensamblador es un programa de computadora escrito para leer un archivo texto estilizado sintácticamente, denominado código de máquina. El código fuente se escribe en abreviaturas inglesas denominadas mnemotécnicos.

El ensamblador toma estos nemotécnicos y los traduce a ceros (0) y unos (1) binarios que son el lenguaje nativo del microprocesador. Esta versión traducida se denomina *codigo de máquina.*

Cada mnemotécnico es una abreviatura de una orden de una instrucción en codigo máquina real. Típicamente, hay una correspondencia uno a uno entre una instrucción en codigo máquina real y su equivalentem nemotécnico inglés.

El macro ensamblador de microsoft es capaz de ensamblar programas para que corran en los microprocesadores 8086-80486 y para sistemas que utilicen el coprocesador matemático. Este ensamblador (MASM versión 6.11) requiere la versión de DOS 3.3 y posteriores implementaciones y un mínimo de 840 Kbytes de memoria de disco, incluidos todos sus archivos adicionales.

Crear un programa en lenguaje ensamblador es un proceso que consta básicamente de tres pasos (ver figura 3.7). En el paso 1, se utiliza un editor de texto para crear el código fuente. En paso 2, el emsamblador se utiliza para convertir del código fuente a código objeto, el cual es similar al código de máquina. El paso 3 es para enlazar, donde se cambia el código objeto a un archivo .EXE, listo para ejecución.

Como es muy improbable escribir código libre de errores, puede ser incluido un paso adicional de depuración para completar el ciclo de desarrollo de un programa en lenguaje ensamblador.

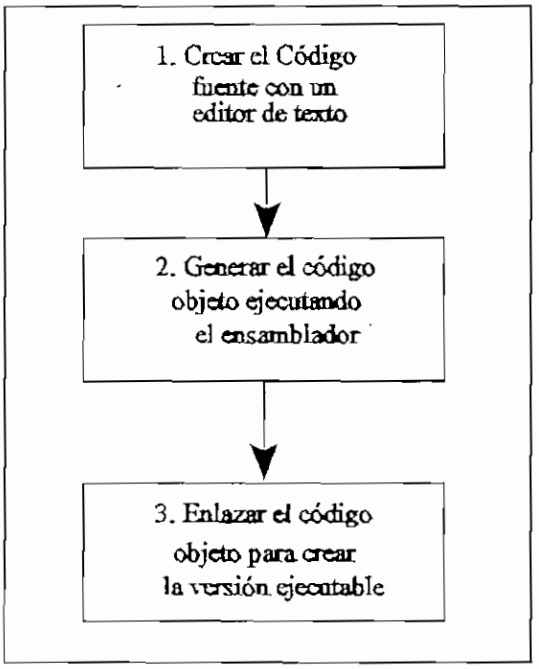


Figura 3.7 Proceso típico de ensamblamiento

Paso 1: *Creación del código fuente.*- Para crear el código fuente, puede utilizarse cualquier editor de texto que genere archivos texto en el formato ACCII , el archivo creado deberá ser de extensión .ASM

Paso 2: *Generación del código objeto.*- En este punto, el código objeto será generado cuando el ensamblador se ejecute.

Una vez creado en el paso 1 el archivo con extensión .ASM, al que lo denominaremos en adelante como SAMPLE.ASM y si el ensamblador se encuentra en la unidad C, deberá teclear:

```
C:\>MASM SAMPLE          ;para el macro ensamblador IBM
C:\>MASM SAMPLE          ; para el ensamblador MICROSOFT.
```

C:\> TASMB SAMPLE ; para el TURBO EDITASM

Estos tres macroensambladores son los más conocidos en el mercado por lo que se los ha enunciado.

Nótese que no fué necesario incluir la extensión del archivo .ASM. Esto es porque los ensambladores suponen esa extensión.

Se obtiene un ensamblamiento con éxito cuando se crea el archivo objeto SAMPLE.OBJ. Aún cuando el archivo tenga todas las instrucciones del programa en código de máquina, este no está en un formato que pueda ser cargado en memoria por el sistema operativo. Es el archivo .OBJ el que será utilizado en el tercero y paso final del proceso de ensamblamiento.

Paso 3: *Enlace.*- En este punto, es necesario utilizar el programa enlazador (LINK) para convertir el archivo objeto en un archivo ejecutable.

El programa LINK es proporcionado con cada ensamblador o con cada DOS. Si el enlazador LINK se encuentra en la unidad C, teclear:

C:\>LINK SAMPLE ; para el macroensamblador IBM

C:\>LINK SAMPLE ; para el ensamblador MICROSOF.

C:\>LINK SAMPLE ; para el TURBO EDITASM

Algo muy importante en este punto es mencionar que si se dispone de varios módulos constituyendo un solo programa, a estos se los deberá enlazar, si así fuera el caso, listándolos separados por el operando "+" . Así por ejemplo, si el programa principal se denomina SAMPLE y sus módulos adicionales son MODULO1 y MODULO2, entonces se debe teclear

C:\>LINK SAMPLE + MODULO1 + MODULO2

Ahora suponiendo que se ha tenido éxito con LINK, entonces se encontrará

el archivo SAMPLE.EXE creado y almacenado en la unidad C. Para ejecutar el programa; suponiendo todavía que la unidad C es la implicita, teclear:

C:\>SAMPLE

3.5 Presntación del 80486.

3.5.1 Modo Real

Los microprocesadores 80286 - 80486 funcionan en el modo real o en el protegido. Los 8086, 8088 y 80186 sólo funcionan en el modo real.

El funcionamiento en el modo real permite que el microprocesador sólo direccione al primer 1 Mbyte de espacio en la memoria, aunque sea un microprocesador 80486. En los sistemas de funcionamiento MSDOS o PCDOS se supone que el microprocesador funciona en el modo real en todo momento. El funcionamiento en el modo real permite que el software de aplicación escrito para el 8086 u 8088 que sólo contienen 1Mbyte de memoria, funcionen en los microprocesadores 80286, 80386 y 80486. En todos los casos cada uno de los microprocesadores empieza a funcionar en modo real en forma implícita (default), siempre que se aplica la alimentación de corriente o si se reestablece el microprocesador.

Segmentos y desplazamientos.- Una dirección de segmento y una dirección de desplazamiento, generan una dirección en la memoria en el modo real. Todas las direcciones en la memoria en modo real, consisten en un segmento y un desplazamiento. El segmento que este ubicado en uno de los registros de segmento, define la dirección inicial de cualquier segmento de memoria de 64K bytes. la dirección de desplazamiento selecciona una localidad dentro del segmento de memoria

de 64K bytes. En la figura 3.8 se ilustra cómo el esquema de direccionamiento de segmento más desplazamiento para seleccionar una localidad en la memoria. En esta ilustración se muestra un segmento de memoria que empieza en la localidad 10000H y termina en la 1FFFFH, de 64 K bytes de longitud. También se muestra la forma en que un desplazamiento de F000H selecciona la localidad 1F000H en el sistema de la memoria. Se verá que la dirección de desplazamiento es la distancia desde el inicio del segmento.

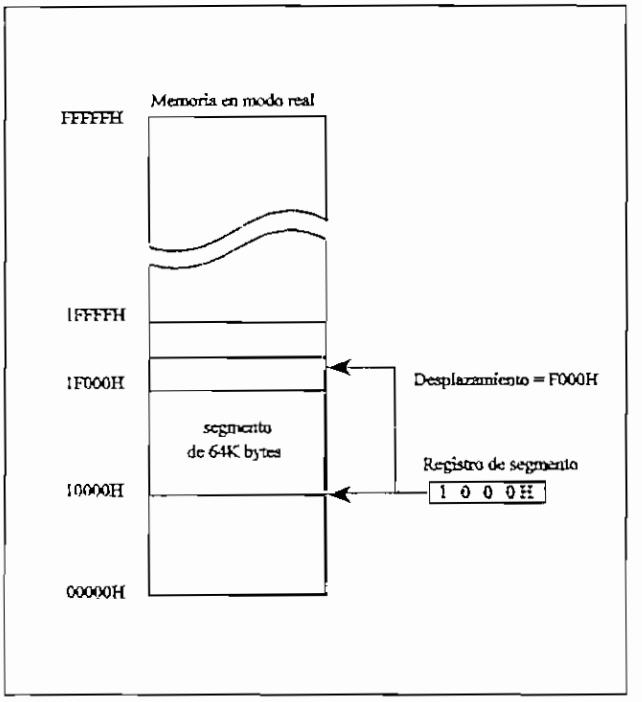


FIGURA 3.8 Modo de direccionamiento en modo real con el empleo de una dirección en el segmento más un desplazamiento

El registro de segmento en la figura 3.8 contiene 1000H, con lo que apunta a una dirección inicial 10000H. En el modo real, a cada registro de segmento se le agrega 0H en su extremo derecho, para formar una dirección de memoria de 20 bits

que le permite colocar el inicio del segmento dentro del primer Mbyte de memoria.

Debido a que un segmento de memoria en modo real tiene 64K bytes de longitud, una vez que se conoce la dirección inicial, para encontrar la dirección final se agrega una FFFFH a la dirección inicial. Por ejemplo, si un registro de segmento contiene 3000H, la primera dirección en el segmento es 30000H y la última dirección es 30000H + FFFFH. En la tabla 3-1 se presentan algunos ejemplos de contenidos de registros de segmento y las direcciones inicial y final de los segmentos de memoria, seleccionadas por cada dirección de segmento.

Registro de segmento	Dirección inicial	Dirección final
2000H	20000H	2FFFFH
2100H	21000H	30FFFH
AB00H	AB000H	BAFFFH
1234H	12340H	2233FH

TABLA 3-1 Ejemplo de direcciones de segmentos

La dirección del desplazamiento se suma a la del segmento para ubicar una dirección en el segmento. Por ejemplo si la dirección del segmento es 1000H y la dirección del desplazamiento es 2000H, el microprocesador direcciona la localidad de memoria 12000H. La dirección del segmento y del desplazamiento, a veces, se escribe 1000:2000 para una dirección de segmento de 1000H y un desplazamiento de 2000H.

Se debe tener en cuenta que en el modo real, sólo los 16 bits en la extrema derecha del registro extendido direccionan a una localidad dentro del segmento de memoria. Nunca se deberá poner un número mayor que FFFFH en un registro de desplazamiento si el microprocesador funciona en el modo real. Si en el modo real se direcciona a una memoria mayor que 100000H (o 10FFEFH si está instalado el sistema HIMEM) ocasionará que el microprocesador interrumpa el programa e indique un error.

Los 8086-80286 permiten tener cuatro segmentos de memoria; el 80386 y el 80486 permiten seis segmentos de memoria. En la figura 3.9 se ilustra un sistema que contiene cuatro segmentos de memoria. Se debe tener en cuanta que los segmentos de memoria pueden tocarse e incluso traslaparse si no se requieren 64K bytes de memoria para un segmento. Pensar que los segmentos son como ventanas que se pueden mover en cualquier superficie de la memoria para accesar a datos o código.

Suponga que un programa de aplicación requiere 1000H bytes de memoria para su código, 190H bytes de memoria para sus datos y 200H bytes de memoria para su pila. Esta aplicación no requiere un segmento adicional. Cuando el DOS coloca este programa en la memoria, se carga el TPA(área de programa transitorio) en la primera zona disponible encima de los manejadores y otros programas de la TPA. En la figura 3.9 se muestra la forma en que esta aplicación se almacena en el sistema de memoria. Los segmentos muestran una superposición o traslape debido a que la cantidad de datos que hay en ellos no requiere 64K bytes de memoria. La vista lateral de los segmentos muestra con claridad el traslape y la forma en que los segmentos se pueden desplazar a cualquier parte de la memoria. Por fortuna para todos, el DOS calcula y asigna las direcciones en el segmento.

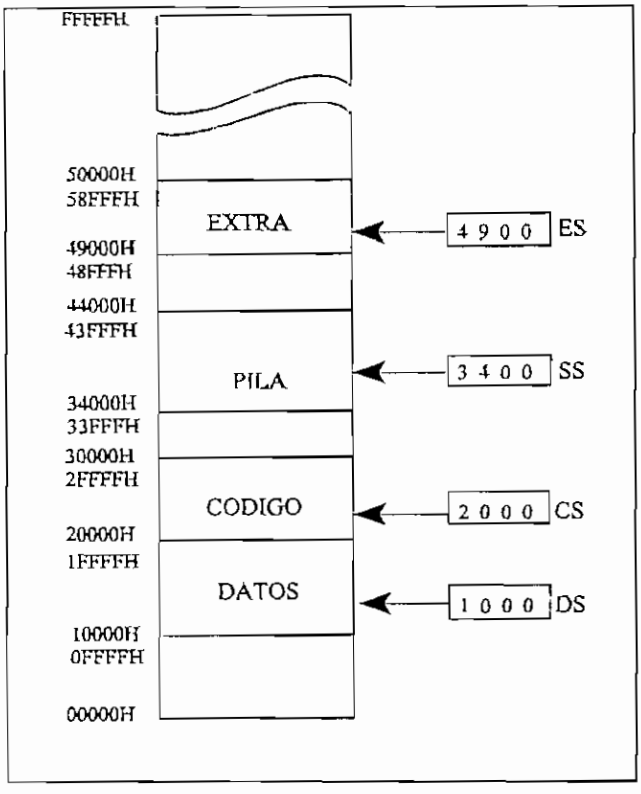


FIGURA 3.9 Un ejemplo de sistema de memoria y se ilustran cuatro segmentos de memoria

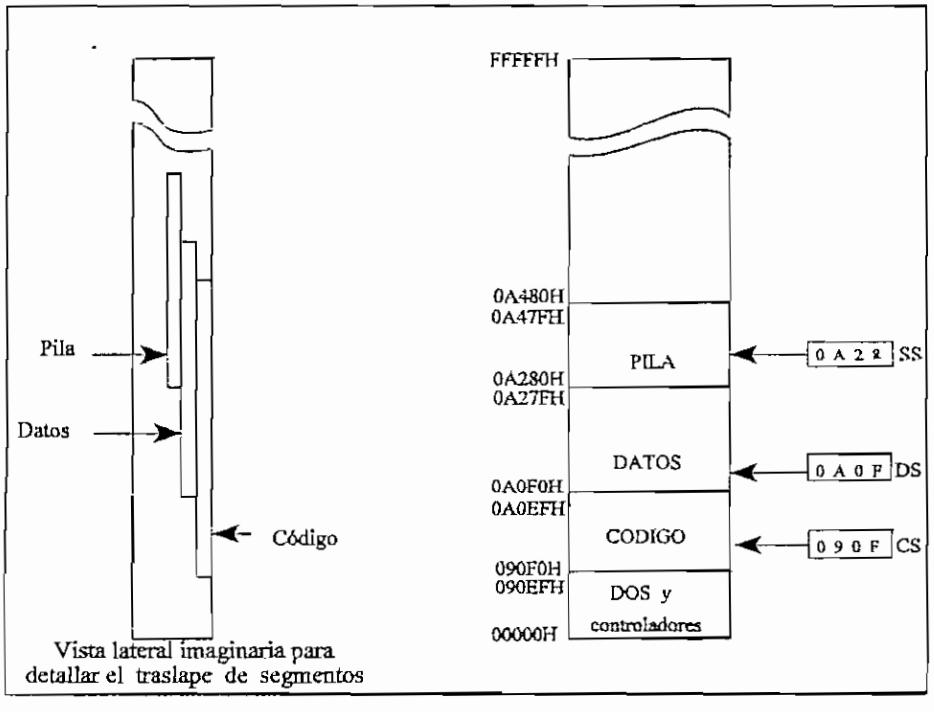


FIGURA 3.10 Mapa de memoria de una aplicación,en que se ilustran segmentos y segmentos traslapados

En el modo real, utilizado principalmente para el arranque, el 80386 se comporta como una versión de 32 bits del 8086. En modo protegido del que se hablará en el siguiente literal, el modo usual de operación, el 80386 actúa como una versión de 32 bits del procesador 80286. La elección entre ambos modos depende del bit 0 del registro de control 0 (el bit "protection enable" (PE), protección permitida). La selección es a nivel del sistema. El cambio de modo real a modo protegido es una acción drástica que el sistema realiza sólo una vez.

El modo real tiene las siguientes características.

- La segmentación se realiza exactamente como en el 8086.
- El espacio de direcciones es de 1Mbyte más 64 Kbytes. Los 64 Kbytes extras provienen del acarreo en el cálculo de la segmentación. El 8086 simplemente ignora el acarreo, con lo que su espacio de direccionamiento es exactamente de 1 Mbyte.
- Las direcciones efectivas(dentro de un segmento) están limitadas a 64 Kbytes.
- Los operandos y las direcciones son por defecto de 16 bits.

Por tanto, el modo real no es exactamente como un 8086, ya que las capacidades de 32 bits del 80386 y las nuevas instrucciones orientadas a aplicaciones se encuentran todavía disponibles. Se podría necesitar un lenguaje ensamblador o compilador especiales con un modo 80386 para acceder a las mismas.

3.5.2 Modo Protegido

El direccionamiento de la memoria en modo protegido (sólo 80286, 80386 y 80486) permite acceso a los datos y programas ubicados en la memoria ampliada

(arriba del primer Mbyte de memoria). El direccionamiento de esta sección extendida del sistema de memoria (la memoria encima del primer Mbyte de memoria se denomina *memoria extendida* o XMS), requiere un cambio en el sistema de direccionamiento de segmento y desplazamiento, utilizado con el direccionamiento de la memoria en modo real. Cuando se direcciona a los datos y programas en la memoria ampliada, todavía se utilizan la dirección de desplazamiento para accesar la información ubicada dentro del segmento. La dirección del segmento, que se describió en relación con el direccionamiento de la memoria en modo real, ya nó está presente en el modo protegido. En lugar de la dirección del segmento, el registro del segmento contiene un *selector* que selecciona un *descriptor*; éste describe la ubicación, longitud y derechos de acceso al segmento de memoria. Debido a que el registro de segmento y la dirección de desplazamiento todavía accesan a la memoria, las instrucciones en el modo protegido se ven iguales que en el modo real. En la práctica, la mayor parte de los programas escritos para funcionar en el modo real, funcionarán sin ningún cambio en el modo protegido. La diferencia entre los modos está en la forma en que el registro de segmento accesa al segmento de memoria.

Para poder cambiar la operación del 80386 del modo real al modo protegido, se deben seguir varios pasos. La operación en modo real se accesa después de una reinicialización dura o cambiando el bit PE a un 0 lógico en CR0. El modo protegido se accesa colocando un 1 lógico dentro del bit de PE del CR0, pero antes que se realice esto se tiene que inicializar algunas otras acciones (existen textos al respecto que explican paso a paso y muy detalladamente como accesar al modo protegido).

3.5.2.1Modo Virtual 8086 (V86)

En modo protegido, el 80386 tiene un submodo denominado *modo virtual 8086*. Se selecciona poniendo a 1 la bandera "Virtual Mode (VM)", correspondiente al bit 17 del registro de banderas extendido. En el modo V86, como en el modo real, el 80386 se comporta como un 8086. La diferencia entre ambos modos es que el cambio a modo V86 puede realizarse a nivel de tarea. Esto es, cualquier tarea que inicialice el registro extendido de banderas puede escoger entre el modo protegido y el modo V86. Así, un sistema puede ejecutar, bien "simultáneamente" (bajo multitarea), o secuencialmente, tanto programas 8086 en modo V86, como programas 80386 generalizados en modo protegido. El cambio a modo V86 no tiene efectos drásticos sobre la segmentación de memoria o la generación de direcciones.

Este modo V86 está diseñado para que múltiples programas de aplicación para el modo real 8086 se puedan ejecutar simultáneamente. La figura 3.11 muestra dos aplicaciones del 8086 mapeadas en el 80386 utilizando el modo virtual. Cuando el sistema operativo permite que se ejecuten aplicaciones múltiples, generalmente se realizan a través de la técnica llamada *división del tiempo*. El sistema operativo designa una cantidad determinada de tiempo para cada tarea. De esta manera todas las tareas reciben un parte del tiempo de ejecución del 80386, resultando en un sistema que parece ejecutar más de una tarea a la vez.

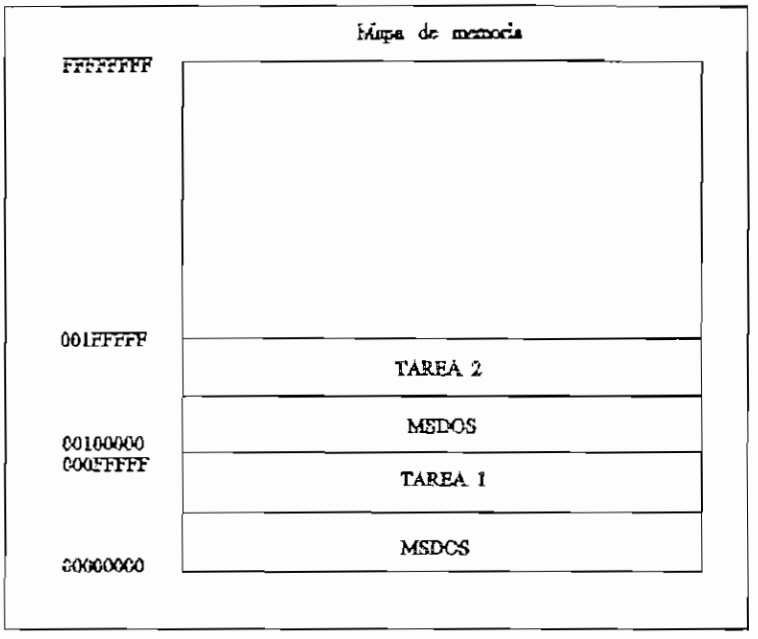


FIGURA 3.11 Dos tareas residentes a un 80386 operadas en el modo virtual 8086

La diferencia principal entre la operación del modo protegido 80386 y el modo virtual 8086 es la forma en que se interpretan los registros de segmento del microprocesador. En el modo virtual 8086, los registros de segmento se utilizan como en el modo real. O sea como una dirección de segmento y una dirección de desplazamiento capaz de accesar un espacio de memoria de 1 Mbytes desde la localidad 00000H - FFFFFH. El acceso a muchos sistemas en modo virtual 8086 se hace posible por la unidad de paginación. A través de la paginación, el programa aún accesa la memoria abajo del límite de 1M byte, sin embargo el microprocesador puede direccionar un espacio en la memoria física en cualquier localidad en el campo de 4G bytes del sistema de memoria.

3.6 Programación en Lenguaje Ensamblador. Programas Ejemplo

El lenguaje ensamblador es un lenguaje potente que da al programador control absoluto sobre la computadora a diferencia de los lenguajes compilados de alto nivel como Pascal, Fortran y otros que dejan al programados a merced del compilador. El lenguaje ensamblador permite el control íntimo de los periféricos, gestión de memoria, velocidad, eficiencia de código, seguridad de datos, además de proporcionar al programador acceso directo a registros, memoria, y a las únicas instrucciones orientadas al bit; permitiendo así escribir rutinas propias o aprovechar la potencia de las rutinas del BIOS suministradas por los fabricantes de computadoras; el dominio de la programación en lenguaje ensamblador requiere experiencia y atención al detalle.

Para la programación en lenguaje ensamblador es importante entre otra cosas, conocer las técnicas de direcionamiento de datos que es la base para la programación en este lenguaje. Existen 8 modos importantes de direccionamiento, en la figura 3.12 se muestran todas las variantes de los modos de direccionamiento de datos con el empleo de la instrucción MOV. Esta ilustración ayuda a mostrar cómo se formula cada modo de direccionamiento de datos con la instrucción MOV y sirve también de referencia. Se verá que son los mismos modos de direccionamiento de datos de todos los microprocesadores 8086-80486, excepto en el modo de direccionamiento con índice escalado. El direccionamiento con índice escalonado sólo se encuentra en los microprocesadores 80386 y 80486.

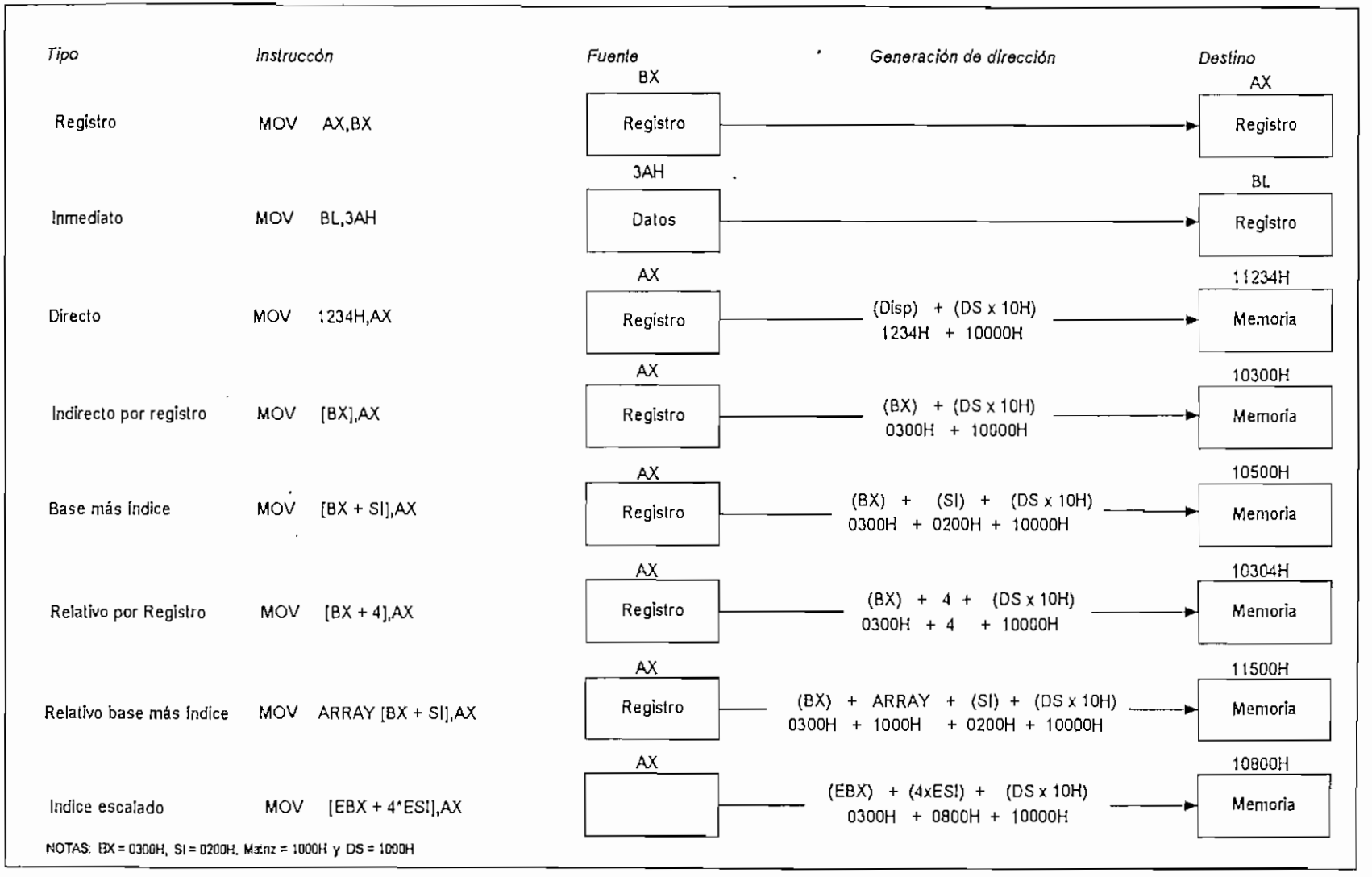


FIGURA 3.12 Modos de direccionamiento de datos

Un programa en lenguaje ensamblador es una serie de sentencias ejecutables que le dicen al ensamblador que operaciones tiene que realizar. Esta serie de sentencias, a menudo, se denomina código fuente. Como en cualquier otro lenguaje, el código fuente del lenguaje ensamblador tiene una sintaxis predefinida.

Cada sentencia del lenguaje está compuesta de cuatro campos:

Campo Etiqueta Campo Nombre Campo Operando Campo Comentario

Sin embargo, ciertas instrucciones del ensamblador no utilizan todos los campos. El campo comentario existe para expresar propósitos o documentación de programación interna y es opcional.

Campo Etiqueta.- Asigna un nombre simbólico a la dirección de comienzo de memoria real de una instrucción del ensamblador. Esto permite al ensamblador que referencia a una instrucción por etiqueta y elimine la necesidad de seguir la pista de direcciones de las instrucciones. Esto es especialmente útil al generar código reubicable. Aunque a cualquier instrucción se le puede dar un nombre, este campo está habitualmente reservado para aquellas instrucciones que serán referenciadas en las definiciones de datos, constantes, segmentos, lazos, bifurcaciones y llamadas a subrutinas.

No se puede utilizar un nombre que coincida con una palabra reservada o directivo del ensamblador. Se puede utilizar caracteres : A-Z , 0-9 , - , $, . , ? , @ , %, en un máximo de 31 caracteres por etiqueta.

Campo Nombre.- Este campo contiene un nemotécnico para una instrucción real del microprocesador. El nemotécnico es una "ayuda de memoria" de dos a siete

caracteres. En lugar de ser una instrucción binaria o hexadecimal para una instrucción en código de máquina, el nemotécnico es una abreviatura en inglés. El nemotécnico de operación hace el código más fácil de leer y comprender y es solamente una tabla de conversión interna del valor binario de código de máquina real. Una operación, o nemotécnico, puede representar una instrucción de máquina, macro instrucción o pseudo-operación.

Campo de Operando.- El campo de operandos contiene la posición o posiciones donde están los datos que van a ser manipulados por la instrucción de la operación. Si la instrucción requiere uno o dos operandos, los operandos están separados de la instrucción por al menos una instrucción en blanco. Si hay dos operandos, estos están separados por una coma. Sin embargo, hay operaciones que no requieren operandos.

Cuando una instrucción requiere dos operandos, el primer operando se denomina operando destino y el segundo se denomina operando fuente.

Campo Comentario.- Es el último de los cuatro campos y puede ser uno de los más útiles. El campo comentario se utiliza para documentar internamente el código fuente del ensamblador. Los comentarios son ignorados por el ensamblador y son útiles sólo al listar el código fuente. Si un comentario se incluye con una instrucción, entonces debe estar separado por al menos un espacio en blanco y comenzar por un punto y coma. Un comentario debe ser utilizado para describir aquellas líneas de código fuente que no son comprensibles inmediatamente.

A continuación se lista un programa en el que constan los cuatro campos antes enunciados.

```
Microsoft (R) Macro Assembler Version 6.11                03/30/97 16:32:05
EJEMPLO.asm                                               Page 1 - 1


                        ;  ***********COMENTARIOS**************
                        ;  PROGRAMA PARA ILUSTRAR UNA SIMPLE
                        ;  ADICION HEXADECIMAL
                        ;
                              .model small
                              .486
                              .stack  64h           ;área para datos de la pila

                        ;  ***********AREA PARA DATOS***********

0000                          .data
0000 05                 num1  db    5h            ;número para la adición
0001 0F                 num2  db    0fh           ;número para la adición

                        ;  ***********AREA PARA CODIGO***********
0000                          .code

                        ;┌──────────────────────────────────────────────┐
                        ;  CAMPO    CAMPO   CAMPO          CAMPO
                        ;    1        2       3              4
                        ;└──────────────────────────────────────────────┘

0000  B8 ---- R         Inicio:   mov   ax,@data    ;pasar a AX valor del
                                                    ;del segmento de datos
0003  8E D8                       mov   ds,ax       ;pasar a DS valor del
                                                    ;segmento de datos

0005  A0 0000 R                   mov   al,num1     ;mueve al registro AL
                                                    ;el valor 5h
0008  02 06 0001 R                add   al,num2     ;suma al registro
                                                    ;AL el valor 0fh
000C  04 10                       add   al,10h      ;suma a AL el valor
                                                    ; inmediato 10h

                                  .exit             ;retorno al DOS
                                  end               ;fin del programa
```

La primera columna contiene la posición de memoria de comienzo real de donde se almacena la operación nemotécnica traducida. Estas son direcciones de 16 bits representadas en formato hexadecimal. La segunda y tercera columnas contienen

la versión del código de máquina real de la instrucción. Estas columnas varian en su número de entradas de acuerdo con el tipo de instrucción y número de operandos requeridos.

Las restantes columnas siguen el formato de programación para un código fuente en lenguaje ensamblador que contiene un campo de etiqueta (campo 1), nombre (campo 2), operando (campo 3) y comentario (campo 4).

En los apéndices se encuentran las instrucciones para el microprocesador y coprocesador matemático como también ejemplos más detallados que ilustran la programación en lenguaje ensamblador.

Capítulo 4

DESARROLLO DEL SOFTWARE

4.1 Definición del problema, necesidad de un programa depurador para computadoras 80486

El tema planteado responde a una necesidad de nuestra Facultad, de contar con un depurador de programas para los microprocesadores de la serie 80486 que son de uso muy extendido por constituirse en el cerebro de las computadoras de la serie IBM PC y compatibles, siendo estas las de mayor uso en nuestro país y para las cuales se desarrolla la gran mayoría de nuevos paquetes de software en el mercado, además será para esta serie de computadoras para las que se desarrollará también software en lenguaje Ensamblador en nuestra facultad. El paquete a elaborarse favorecerá especialmente en el desarrollo de programas escritos en este lenguaje hasta su ejecución correcta constituyendose así en una herramienta útil de depuración e investigación de las características de los microcomputadores.

Básicamente la elaboración de esta tesis se basa en una estructura de módulos autónomos que realizan las diferentes funciones necesarias para el depurador, esto facilita de manera el realizar una programación más independiente permitiendo hacer modificaciones y mejoras en cada módulo sin afectar el funcionamiento de los demás.

La presentación de datos en pantalla se realiza en base a ventanas previamente elaboradas y almacenadas en un archivo independiente, a las ventanas más importantes se las deja residentes en un buffer de memoria temporal, desde donde las invoca el programa principal, mientras que a las menos usadas se las llama directamente desde el archivo de ventanas, esto con el fin de lograr mayor rapidez. El uso de ventanas facilita la modificación de información sobre la pantalla y la sobreposición temporal de otra información. El archivo de ventanas es elaborado mediante el programa "VENT.EXE" que permite la elaboración de áreas de pantalla o ventanas en modo texto, la información de estas puede ser fija o variable de acuerdo a la forma como fueron elaboradas según necesidades del depurador. Las ventanas son almacenadas en el archivo de ventanas bajo un formato predeterminado que debe ser tomado en cuenta al momento de invocarlas desde el programa depurador.

PRO486.EXE está elaborado para depurar programas en modo real (1 Mbyte de memoria) y no en modo protegido, este fué enfocado para fines didácticos y para la depuración de programas escritos en lenguaje de bajo nivel por lo que no se enfocará el formato del modo protegido.

4.2 Desarrollo del programa depurador

4.2.1 Estructura del programa

El depurador está formado por varios archivos, uno principal con el cual se inicia una sesión de depuración denominado " PRO486.EXE " y varios archivos auxiliares utilizados por el archivo principal ya sea directamente durante la

ejecución del depurador o indirectamente cuando el usuario elige las opciones de menú o funciones que requieren de tales archivos. Algunos de los archivos auxiliares son indispensables como Child486.exe y Wincolor.vnt o Winbn.vnt, mientras que otros son opcionales como Pro.mcr, Pro486.hlp, Command.com, Edit.com, Masm.exe y Link.exe. La ausencia de los archivos indispensables no permitirá iniciar una sesión de depuración, mientras que la ausencia de los archivos auxiliares no indispensables únicamente limitará el uso de alguna de las facilidades del depurador. En los siguientes párrafos se describe la función de cada uno de éstos archivos auxiliares.

CHILD486.EXE.- El depurador trabaja en dos niveles, el nivel Padre que corresponde al depurador en sí y el nivel Hijo que teóricamente corresponde al programa que se está depurando. El programa principal inicialmente libera memoria del sistema mediante la función 4A de la interrupción 21h, ya que si un programa se está ejecutando, este toma el control de toda la memoria del sistema (640kbytes) y si se trata de ejecutar otro proceso (proceso hijo) habrá un error de falta de memoria. Una vez liberada la memoria se ejecuta Child486.exe como proceso hijo, que básicamente cumple dos funciones :

- Realiza una llamada a la interrupción 63h con AX = 8080h que ha sido previamente redireccionada a una rutina que entrega el control al proceso padre, esto significa que se está en el nivel hijo pero se sigue ejecutando el proceso padre (depurador), este artificio permite al depurador poder cambiar de programa a depurar sin tener que salir nuevamente al sistema

DOS, simplemente regresa al nivel padre y se repite el proceso de bajar al nivel hijo. A continuación de la llamada a la interrupción el programa contiene el código para terminar el programa (función 4C de la interrupción 21h) en caso de que por alguna razón la interrupción 63h no se haya podido redireccionar adecuadamente.

- Al ejecutar un proceso Hijo, el sistema DOS busca el primer segmento (64Kbytes) de memoria libre para cargar desde ahí al proceso hijo , manteniendo reservado el espacio de memoria del proceso padre. Una vez ejecutado Child486.exe como proceso hijo, el depurador toma los segmentos de este como espacio para cargar al programa que se va ha depurar. Se lo carga como Overlay (Sobrepuesto), esto es listo para ejecutarse, manteniendo todos los formatos de Código, Datos y Pila para posteriormente ser ejecutado.

WINCOLOR.VNT, *WINBN.VNT*.- Son los archivos que contienen las ventanas para monitores a Color y Blanco/Negro respectivamente. Existen las ventanas principales que son usadas con más frecuencia y las ventanas secundarias que se usan esporádicamente. Al inicio, el programa principal (PRO486.EXE) copia todas las ventanas principales desde el archivo de ventanas a un buffer de memoria temporal para ganar velocidad en la presentación de las ventanas, mientras que al ser requeridas las ventanas secundarias se leen directamente desde el archivo de ventanas, hacia una determinado buffer de memoria en donde permanecen como residentes hasta que el espacio se requiera para otra ventana.

Una vez que una ventana secundaria está como residente si la requiere una segunda vez consecutiva es leída directamente del buffer. Se aceptan máximo 40 ventanas principales.

El archivo de ventanas es creado con un formato predeterminado por el programa VENT.EXE. Consta de un índice que ocupa 6 grupos (cada grupo tiene 256 bytes), distribuidos en 70 bloques de 24 bytes cada uno, el primer bloque ver Figura 4.1, tiene información general del archivo. A partir del segundo bloque se asigna cada uno de estos a una ventana y mantienen la información del formato de la ventana, ver Figura 4.2. Luego viene la información que contendrá cada una de estas ventanas.

Desplazamiento (bytes)	Parámetro	Valor actual (Valores posibles)
0	Ventanas válidas	0 (0 a 61)
1	Grupos válidos	0 (0 a 244)
2	Ultima ventana válida	2 (2 a 63)
3	Ultimo grupo válido	5 (5 a 249)
4	Primera ventana libre	3 (3 a 63)
5	Primer grupo libre	6 (6 a 249)
6	Ventana actual	3 (3 a 63)
7	Grupo actual	6 (6 a 249)
8	Número de ventanas principales	0
9	Número de Grupos principales	0 (0 a 40)
10-11	Dirección de ventanas residentes	
12	Tamaño de área de ventanas residentes	
13-14	Manejador del archivo de ventanas	
15	próximo grupo del área de ventanas residentes a ser usado	
16-24	Sin uso	

FIGURA 4.1 Bloque de Parámetros Generales en el archivo de ventanas

Desplazamiento (bytes)	Parámetro	Valor actual (Valores posibles)
0 - 7	Nombre de la ventana	
8 (bit cero)	Categoría de la ventana	0 (0 /1)
8 (bit uno)	Residencia	0 (0 /1)
9	Número de grupos de la ventana	0 (0 /4)
10 - 11	Número de variables en la ventana	0 (0 a 512)
12 - 13	Posición de la esquina superior izquierda	0 (0 a 79 y 0 a 22)
14 - 15	Posición de la esquina superior derecha	0 (0 a 79 y 0 a 22)
16 - 19	Números de Grupos	0 (6 a 69)
20 - 23	Números de grupo en área de residentes	0

FIGURA 4.2 Uno de los bloques de Parámetros particulares

Algunas ventanas incluyen información fija, mientras que otras están formadas por segmentos de información fija y variable en donde se puede insertar los datos deseados durante la presentación de la ventana, esta información puede presentarse en formato ASCII y en formato hexadecimal de acuerdo a como se las invoque. Por ejemplo si se quiere presentar el byte 38h, si se especifica como ASCII, en la ventana se presenta el carácter 8, mientras que si se especifica como hexadecimal se presentará en la ventana los caracteres 3 y 8.

En el programa principal se deben implementar procesos para que presenten adecuadamente las ventanas en pantalla respetando los formatos de cada una de ellas, para esto se han desarrollado macros (contenidos en el archivo MPRO .ASM) que realizan las diferentes tareas de manejo de ventanas, como: cargar ventanas

principales en memoria, presentar las ventanas en pantalla, presentar la información fija y variable de las ventanas, etc.

PRO486.MCR.- Destinado a almacenar la información de los Macros de Usuario y de no existir dentro del directorio o subdirectorio en el cual se esté ejecutando el depurador, es creado automáticamente con una longitud de 3138 bytes, distribuidos como se ilustra en la figura 4.3.

Desplazamiento	Uso
0 a 64	Espacio para el nombre de 8 macros (7 caracteres por nombre, el primer byte de cada nombre indica el número de comandos del macro)
65	Byte en blanco
66 a 3136	espacio para los datos de los macros (128 bytes por macro)
3137 a 3138	Siempre contienen los valores 46h y 4Fh e indican fin del archivo

FIGURA 4.3 formato del archivo Pro486.mcr

Es importante diferenciar entre los Macros de Usuario y los macros de programación, los primeros son un conjunto de bytes asociados a los códigos de teclado, grabados bajo un nombre. El depurador cada vez que corre o prueba un macro de usuario lo que hace es leer los bytes del macro como si fueran ingresados desde teclado. Por ejemplo si el macro de usuario contiene los códigos de las teclas flecha a la derecha (4Dh), Enter (0Dh), F6 (40h), al ejecutar el macro, el depurador automáticamente mueve la posición de la opción del menú a la derecha, luego realiza un Enter esto es, ingresar a las subopciones de la opción actual si las

tiene o ejecuta la rutina correspondiente a la opción actual si no tiene subopciones, luego ejecuta la función F6 que es editar una archivo .ASM en este momento finaliza la ejecución del macro y vuelve a la ejecución normal recibiendo códigos del teclado.

El programa principal carga los macros desde el archivo Pro.mcr al buffer de macros del depurador, desde donde serán leídos al ser usados. Los macros nuevos que se creen se graban inicialmente en el buffer de macros y una vez que se sale del depurador, este se encarga de grabar automáticamente los macros en el archivo.

Los macros de programación por el contrario son básicamente instrucciones escritas por el usuario e identificadas mediante un nombre, al ser invocado dicho nombre dentro de un determinado programa, las instrucciones de este macro serán insertadas (al momento de ensamblar) y ejecutadas (al momento de depurar o correr el programa) como si se tratará simplemente de otra instrucción del lenguaje de programación que se esté utilizando y que realiza tareas que a bien desee el programador, facilitando así la programación.

PRO486.HLP.- Este es un archivo de texto, contiene información acerca de las características y uso del depurador, Interrupciones BIOS y DOS con sus respectivas funciones, Instrucciones del Microprocesador (hasta el 80486) y Coprocesador Matemático, información general de la Memoria del Microprocesador, Códigos de teclado, Atributos, etc. Se accede a este archivo

mediante la tecla <F2> estando en el menú principal, es necesario además que esté presente el editor, si no existe el archivo Pro486.hlp se iniciará el editor con un archivo vacío. Si se presiona <F2> desde otros niveles de menú se presenta una ventana con una ayuda breve acerca de las opciones del menú actual.

COMMAND.COM.- Es el archivo del sistema operativo, y se requiere para utilizar la opción SHELL del depurador que permite salir temporalmente al sistema para realizar cualquier trabajo de manejo de archivos, y retornar nuevamente al depurador digitando EXIT. Este archivo debe estar ubicado en el directorio o subdirectorio en el que se encuentre Pro486.exe o en la RAIZ del sistema.

EDIT.COM.- Este es un editor de archivos de texto que se usa para algunas opciones del depurador : Función Ayuda (<F2>) y para la función editar el archivo fuente (<F6>). Debe estar ubicado en el directorio desde el que se está ejecutando el depurador.

MASM.EXE y LINK.EXE.- Estos archivos son utilizados en la opción eXe del menú principal que permite Ensamblar y Enlazar el archivo previamente editado con la opción <F6>. Estos archivos deben estar ubicados en el directorio en el que se este ejecutando el depurador, además en este mismo directorio deben estar presentes los siguientes archivos : DOSXNT.EXE, ML.EXE, ML.ERR, necesarios para Masm.exe.

Todos los archivos deben estar ubicados en el mismo directorio de PRO486.EXE. Para poder usar la facilidad del Ratón es necesario que este instalado cualquier controlador de Ratón.

El depurador debe ser usado bajo ambiente MS-DOS, si se ejecuta en ambiente Windows especialmente Windows 95 se presentarán algunas dificultades debido a que realiza un control total del sistema y no permitirá que el depurador realice ciertos cambios en el sistema. En el caso de disponer de Windows95 se debe reiniciar el computador en modo MS-DOS, esto es salir completamente del ambiente gráfico de Windows.

4.2.2 Programación por módulos

Un *Módulo* es uno o varios procedimiento agrupados para realizar una función determinada como parte de un programa principal. La idea de los módulos es mantener una cierta independencia al momento de realizar modificaciones o mejoras en los módulos sin que afecte a los demás. Al hablar de varios módulos que realizan trabajos específicos cada uno, se hace necesario un módulo principal que inicialice el programa, coordine el trabajo de los demás módulos, que controle el flujo del programa y finalice correctamente. El programa deberá inicializar siempre en este módulo y cada vez que se entregue el control a otro módulo, este finalmente lo devuelve al principal.

PRO486.EXE está constituido por seis módulos los que permiten realizar las diferentes tareas del depurador, paralelamente a la mayoría de estos módulos se han creado archivos independientes que contienen macros con las tareas que se

usan más frecuentemente en los módulos, esto ahorra tiempo al momento de escribir un programa.

De acuerdo a lo expuesto anteriormente INICIAL.ASM se constituye en el módulo principal encargado de la coordinación de las tareas, mientras que PRO486.ASM, DEPURA.ASM, MEMORIA.ASM, ENSAMBLA.ASM, DESENS.ASM ejecutan los diferentes trabajos del depurador de acuerdo a como los invoque el módulo principal. A continuación describimos la función de cada módulo :

MODULO INICIAL.ASM .- Este es el módulo principal con el que se inicia el depurador y controla el flujo del programa, su archivo de macros asociado es INIMAC.ASM.

Inicial.asm está encargado de la administración del menú y funciones del depurador. Determina el nivel actual del menú y dentro de este la opción a resaltar, además de mostrar el mensaje explicativo de lo que realiza la opción actual. Una vez que se selecciona una opción del menú o una función, se encarga de seleccionar la rutina correspondiente para realizar el proceso, y retoma nuevamente el control cuando finaliza el mismo.

Para lograr que este módulo sea con el que se inicie el flujo del programa, es decir que el puntero del programa (IP) apunte a este módulo, se debe incluir la directiva END al final de este módulo de la siguiente forma: END EXPRESION_OPCIONAL, donde EXPRESION_OPCIONAL es una etiqueta ubicada en la posición que se desea se inicie el programa, en el caso de PRO486.EXE es INICIO ubicada en el módulo INICIAL.ASM. Si depuramos

PRO486.EXE con otro depurador de programas o con el mismo PRO486.EXE (mediante la línea de comando PRO486 PRO486.EXE) se podrá observar que el puntero de instrucciones (IP) apunta a la primera instrucción que se encuentra después de la etiqueta INICIO de este módulo. Es importante indicar que cuando se tengan varios módulos como PRO486.EXE, solo uno de ellos puede tener la EXPRESION_OPCIONAL, si no se da un punto de entrada, el enlazador (LINK) identificará uno de los primeros módulos (.OBJ) a ser enlazados.

El diagrama de flujo del módulo INICIAL.ASM se encuentra en la figura 4.4, de esta forma se tiene una visión más clara de lo que realiza este módulo.

A partir de la etiqueta INICIO se borra la pantalla, posteriormente hace un llamado al procedimiento RINIC (ubicado en el módulo PRO486.ASM), que se encarga de establecer si existen las condiciones apropiadas para el funcionamiento del depurador como es el tipo de Monitor, la presencia de los archivos auxiliares necesarios, cargar el objetivo, presentara las ventanas del depurador, reduce a nivel hijo, e indica si no se cumplen las condiciones y si se puede continuar con la ejecución del programa, entre otras.

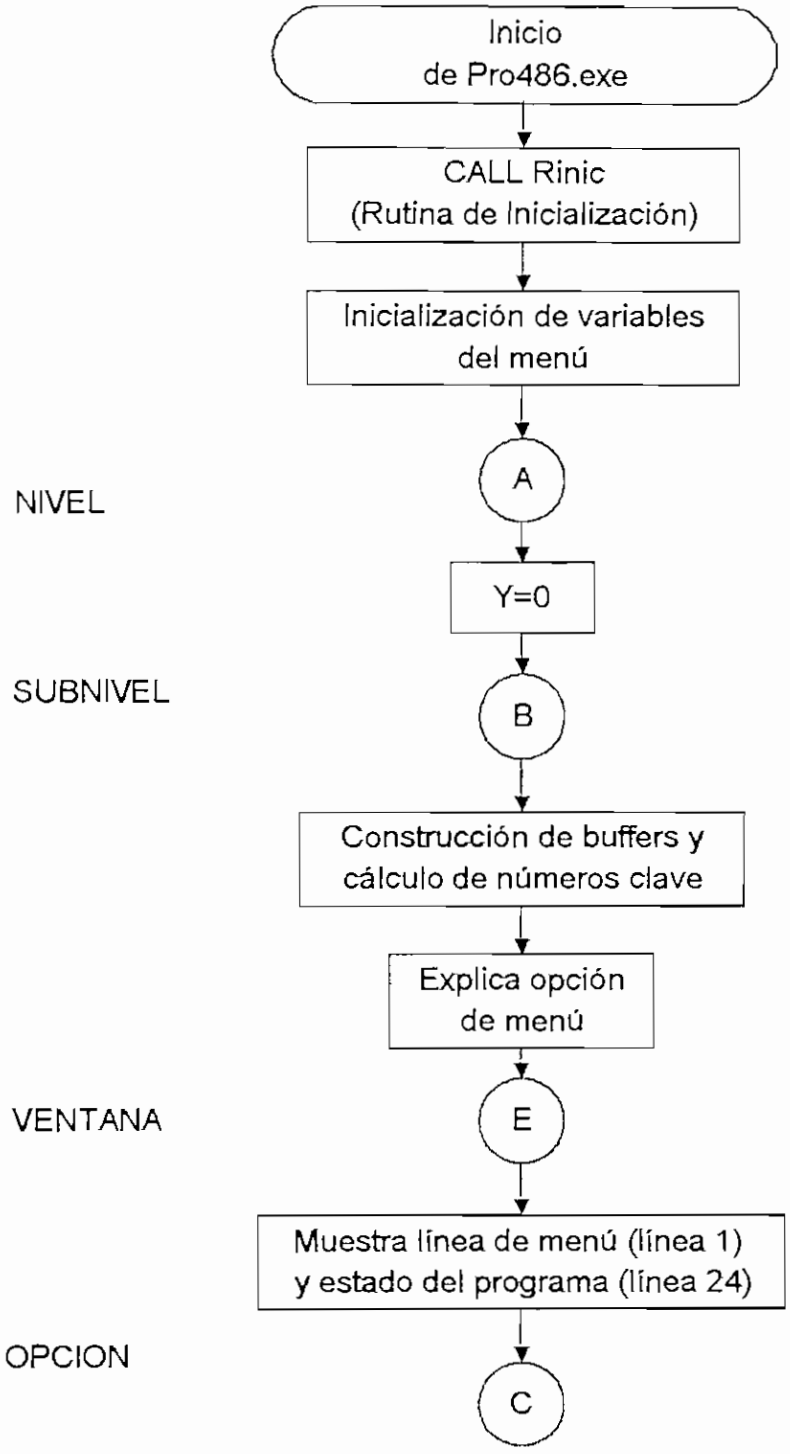


FIGURA 4.4 Diagrama de Bloques del archivo Inicial.asm, primera parte

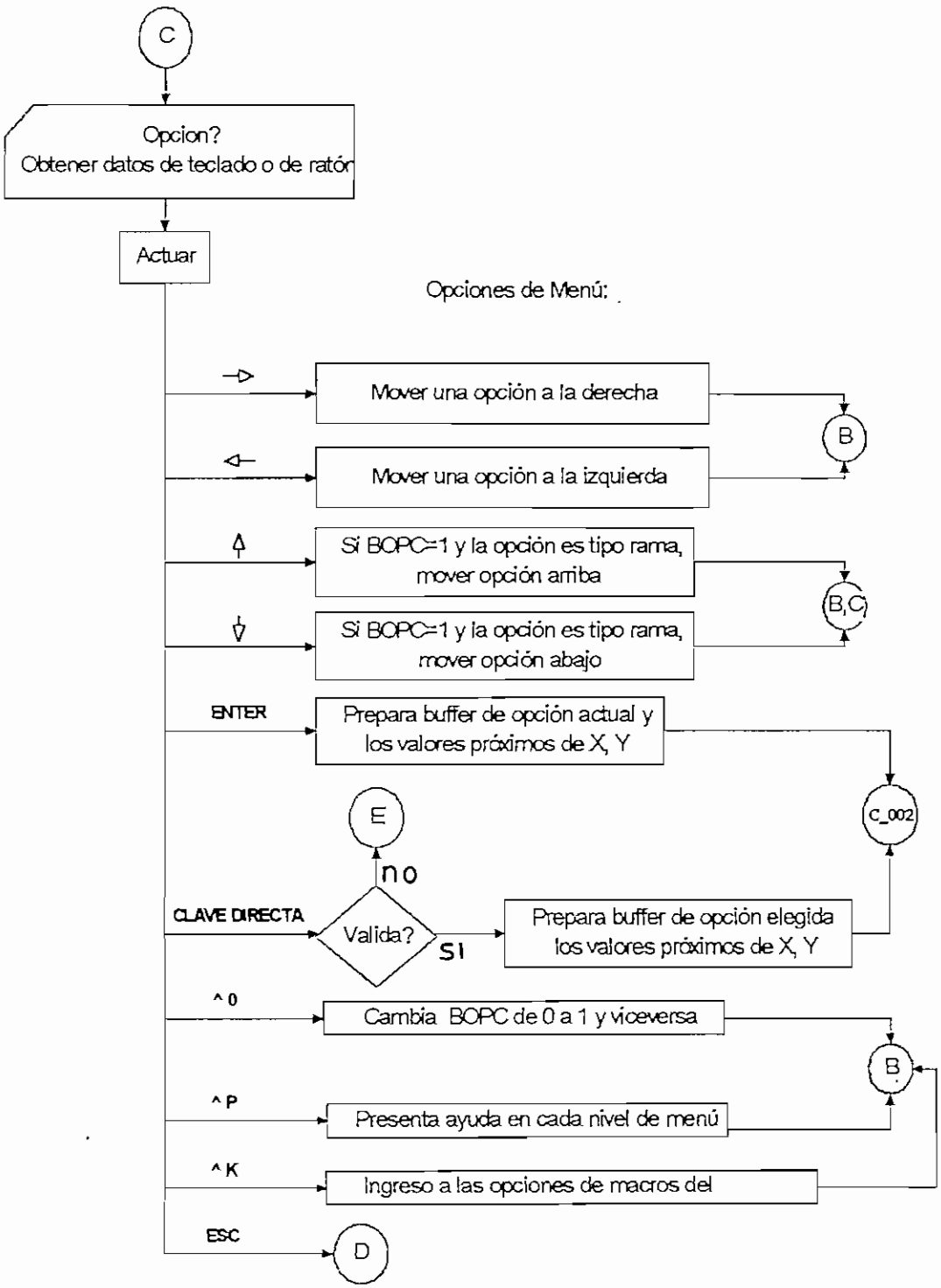


FIGURA 4.4 Diagrama de Bloques del archivo Inicial.asm, segunda parte

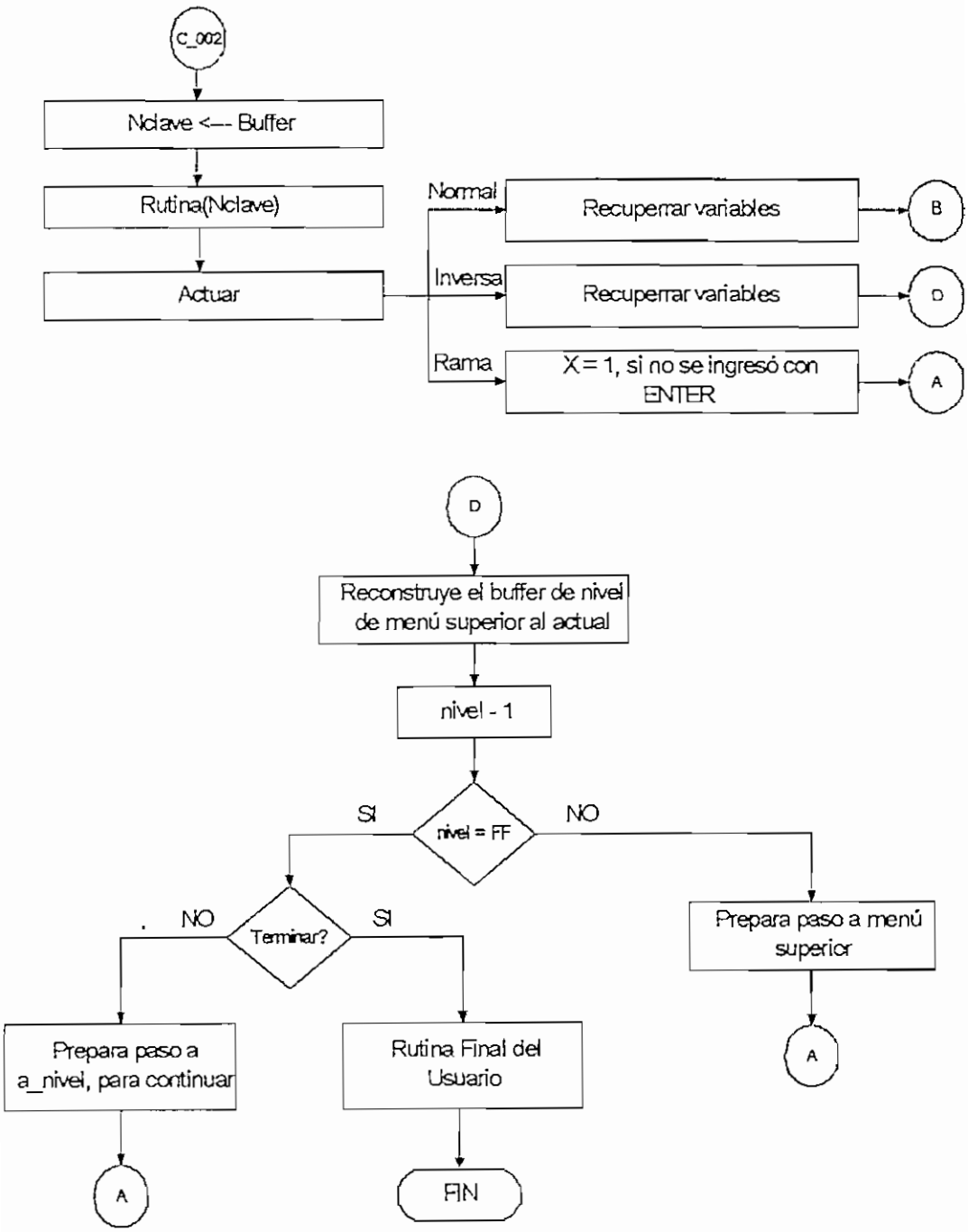


FIGURA 4.4 Diagrama de Bloques del archivo Inicial.asm, tercera parte

Posteriormente se inicializan las variables del menú para indicar su nivel, opción actual, etc. En este punto es necesario la construcción de buffers que contienen información de cada opción de menú como el mensaje explicativo de la opción, la línea de menú que debe mostrarse y la rutina que debe de ejecutarse una vez seleccionada la opción, para luego mostrar en la línea 24 una breve explicación como ayuda de la respectiva opción resaltada, la línea de menú y el estado del programa. En este momento se tiene lista la pantalla con toda la información, ingresando entonces a un lazo de espera hasta que se presione una tecla o se accione el Ratón para elegir cualquiera de las opciones o facilidades del depurador por parte del usuario (este trabajo está contenido en el macro Opcion? ubicado en el archivo Inimac.asm).

Una vez escogida la opción, el macro Actuar determina y salta a ejecutar la rutina adecuada de tratamiento de la misma, para posteriormente devolver el control al nivel o subnivel resepectivo.

MODULO PRO486.ASM.- Este módulo es uno de los más importantes luego de Inicial.asm, contiene todas las rutinas asociadas a cada una de las opciones de menú o de teclas de funciones del depurador, las rutinas encargadas de mostrar las ventanas en pantalla con su respectiva información y también una rutina importante denominada Rinic que determina las características apropiadas del computador para que trabaje el depurador.

Como indicamos anteriormente, el depurador se inicializa mediante la rutina Rinic la que primeramente determina el tipo de monitor (función 0Fh de la interrupción 21h del DOS), y si este es adecuado para el funcionamiento del

depurador. El depurador acepta los siguientes modos de pantalla para vídeo (ver Figura 4.5):

Modo	Tipo	Columnas	Filas	Colores
02h	texto	80	25	B/N
03h	texto	80	25	16 colores
07h	texto	80	25	4 colores

FIGURA 4.5 Modos de vídeo soportados por el depurador

Luego de esto se inicializan ciertas variables necesarias especialmente para el manejo de ventanas como es la detección de la dirección de vídeo para escribir directamente en pantalla. Una vez verificado el tipo de monitor se trata de cargar el archivo de ventanas (Wincolor.vnt o winbn.vnt) por medio del macro Idvnt, primeramente detecta si existe el archivo para luego determinar si realmente contiene la información de las ventanas, en caso de no cumplirse las condiciones anteriores se presenta un mensaje de error y se termina el programa, caso contrario se pasa a determinar la existencia del archivo de macros Pro486.mcr, si este no existe se crea un archivo de macros vacío. En el caso de existir y ser erróneo o haya dificultad al crearlo, se sale del programa con el mensaje de error respectivo.

Posteriormente se libera memoria por medio de la rutina Acortamimem esto es indispensable para ejecutar un subproceso "hijo" , luego se redefine el vector de la interrupción 63h para que apunte a una rutina definida por el usuario, para luego

descender al nivel de proceso hijo mediante la ejecución de Child486.exe, de no existir este archivo se termina el programa con un mensaje de error.

El paso siguiente es cargar el archivo que se va a depurar (mediante la rutina Loadexe ubicada en Depura.asm) aquí se tienen varias opciones:

- El parámetro que se pasó (nombre de archivo con la extensión .EXE o .COM) fue válido y se pudo cargar el archivo correspondiente, en este caso se sigue con la ejecución normal del depurador.
- No se pasó ningún parámetro al entrar al depurador, en este caso se da un mensaje y se continua con la ejecución pudiendo el usuario cargar desde el interior con la opción Archivo del menú.
- El archivo especificado no existe o tiene errores, en cuyo caso se termina con el depurador con el mensaje de error correspondiente.

Aquí se termina con el proceso de inicialización del depurador y finalmente se devuelve el control a Inicial.asm .

Las rutinas de manejo de ventanas se encargan de presentar las diversas ventanas del depurador con la información respectiva de cada una de ellas. Pdebug es la rutina que presenta y actualiza la información de las ventanas más importantes del depurador como son registros, memoria, banderas, código, pila , opcode y la ventana opcional con la información del Coprocesador Matemático.

Las rutinas de opciones del menú en su totalidad preparan las condiciones necesarias como obtener datos necesarios (generalmente desde teclado), inicializar

variables; para posteriormente llamar a otras rutinas (ubicadas en otros módulos más especializados con determinados procesos) que son las que realmente realizan el trabajo. Mientras que las rutinas de funciones no requieren rutinas de otros módulos y realizan el trabajo en su totalidad en este módulo en forma independiente.

Otra rutina importante dentro de este módulo es Rfinal la que es llamada antes de salir del depurador y se encarga de guardar información de los macros al archivo de macros, borra la pantalla, restaurar interrupciones y devolver el control al proceso padre.

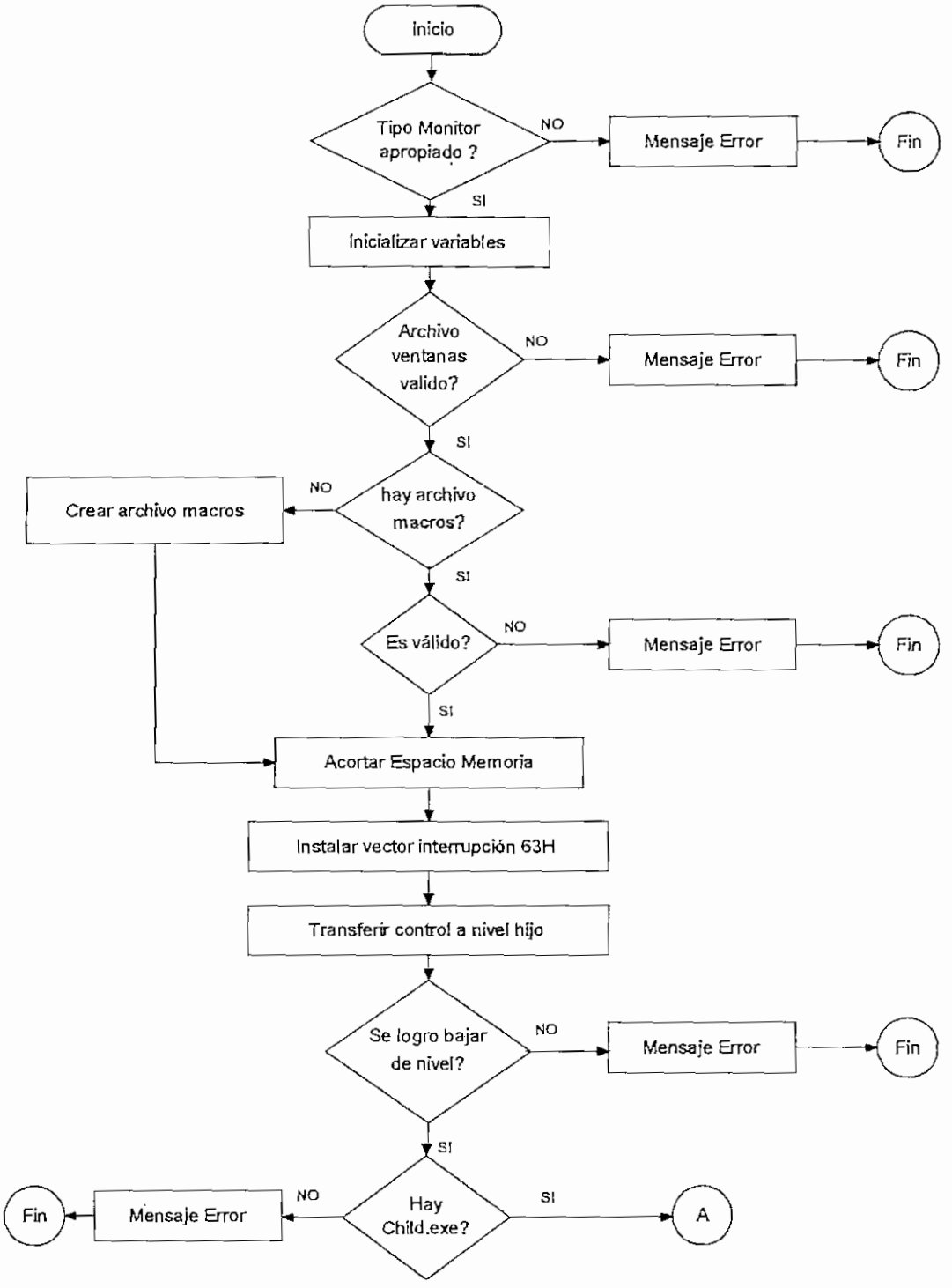


FIGURA 4.5 Rutina principal del módulo PRO486.ASM primera parte

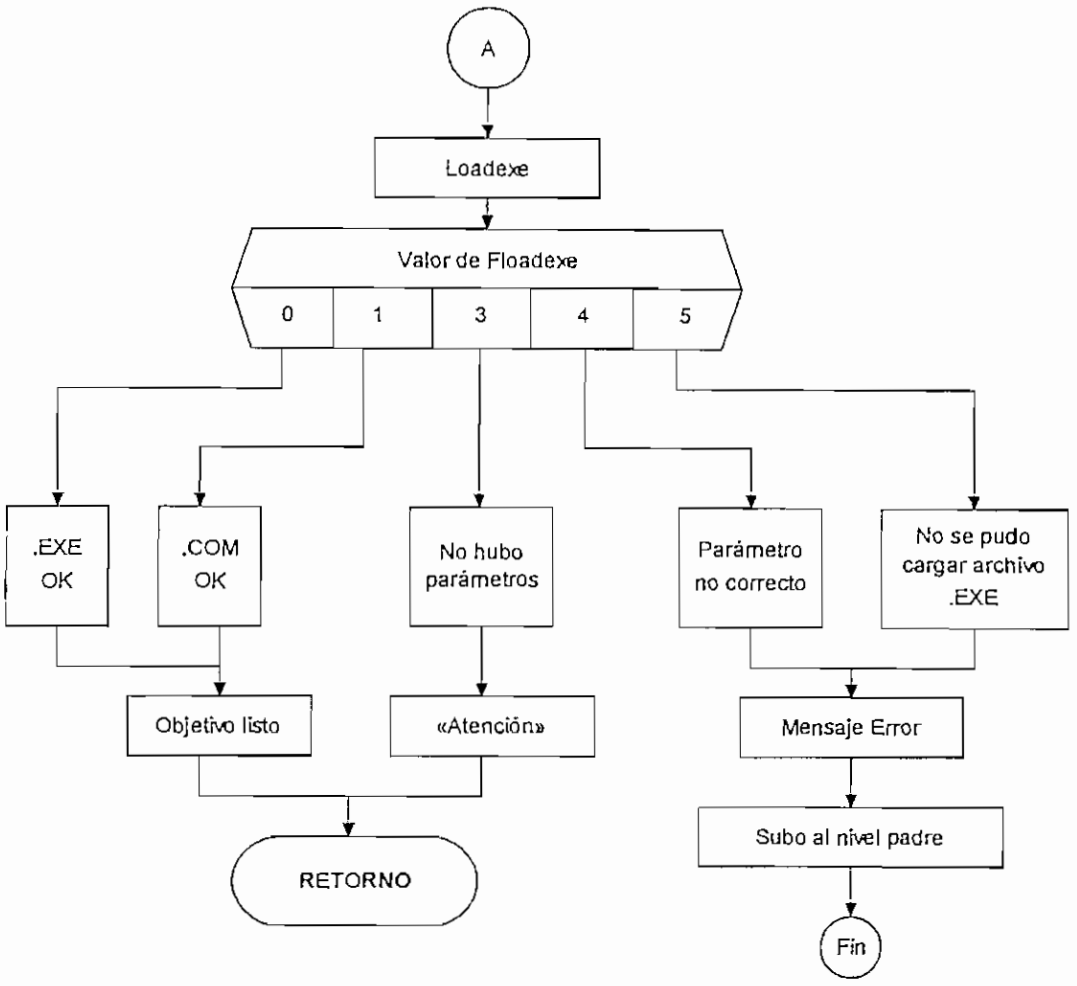


FIGURA 4.5 Rutina principal del módulo PRO486.ASM segunda parte

Los archivos de macros MPRO.ASM, PROVNT.ASM están relacionados con el módulo Pro486.asm, el primero contiene los macros para recepción de datos, manejo de pantalla (borrar, presentar datos, etc.), también contiene los macros de instrucciones para el manejo del archivo de Macros de Usuario (conjunto de comandos) como abrir y leer, cerrar, crear archivo de macros entre otras. Provnt.asm contiene los macros para manejo de ventanas como presentar ventanas,

actualizar ventanas, acceso directo a memoria de vídeo, intercambiar entre pantalla de usuario y del depurador, etc.

MODULO DEPURA.ASM.- Este módulo contiene un conjunto de procedimientos de especial importancia como son ejecución paso a paso, ejecutar el programa hasta encontrar un punto de parada activo, correr hasta una cierta condición o dirección , ingreso o no a subrutinas, entre otras.

La rutina *StoreRegs* pone en memoria los registros y banderas del programa que se está depurando cada vez que se ha ejecutado una instrucción o porción de instrucciones del mismo y se retorna el control al depurador. La siguiente vez que se va ha ejecutar el código del programa que se está depurando se restablecen los valores de los registros y banderas mediante la rutina *LoadRegs.*

La rutina *Exuniq* ejecuta una sola instrucción del código del programa de usuario, ingresando a las llamadas a subrutinas (Call), lazos (Loop), intrucciones de movimientos repetitivos (Reps). Es usada por la opción PASO del menú, esto si la opción de ingreso a subrutinas está habilitada (SiSubr), mediante la función <F8>. Se guarda una porción del código ubicado antes de la instrucción que se desea ejecutar, luego se coloca el código necesario para habilitar la interrupción 01h (poner 1 en la bandera de trampa T) de modo que se active cada vez que se ejecute una instrucción. Previamente se redefine el vector de interrupción de Trampa de forma que apunte a la dirección de retorno luego de ejecutar la instrucción, esto es devolver el control al depurador, en esta parte se guardan los valores de registros del programa de usuario, restaura el código alterado, se retorna al menú .

Un paso importante en la depuración paso a paso es detectar el comportamiento del computador ante la interrupción 1 que se produce cuando la bandera de trampa (T) es activada (puesta a 1), lo que se debería esperar es que inmediatamente después de que T sea puesto a 1 se ejecute una línea de programa y luego se active la interrupción, lo que no ocurre en ciertas computadoras en las que la interrupción se activa después de ejecutarse dos instrucciones, en cuyo caso se deberá insertar una instrucción NOP antes de la instrucción a ser ejecutada para que se ejecute únicamente una instrucción.

El procedimiento *Exhasta* ejecuta desde la dirección actual hasta la instrucción anterior a la dirección de parada requerida, esta rutina es usada por la opción CORRE del menú estando habilitada la opción Dirección mediante la función <F7>. En este caso se inserta la interrupción 03h (Breakpoint-Punto de ruptura) después de la instrucción hasta la que se quiere ejecutar. De igual forma que en la interrupción trampa, el vector de la interrupción de Ruptura se ha redefinido para que apunte a la rutina del depurador que retoma el control después de ejecutar el código del programa que se está depurando. Es importante mencionar que aunque las dos interrupciones 01h y la 03h, son redefinidas por el usuario, su funcionamiento es diferente. La interrupción 01h una vez que ha sido habilitada es activada automáticamente por el sistema al ejecutarse una sola instrucción, en cambio que la interrupción 03h no requiere ser habilitada pero para que se ejecute debe formar parte del código del programa en el punto requerido, es decir el punto hasta el cual se desea que se ejecute el programa.

El procedimiento *Expaso* ejecuta una sola instrucción del programa, pero a diferencia de Exuniq, ésta no ingresa a subrutinas, lazos ni repeticiones, estas son

ejecutadas en un solo paso. Es usado por la opción PASO del menú cuando se observa NoSubr en la línea 23 del menú de funciones (<F8>). Esta rutina analiza que tipo de instrucción se va a ejecutar, en el caso de las instrucciones de llamada a subrutinas, lazos y eventos repetitivos hace un llamado a la subrutina Exhasta y en cualquier otro caso hace un llamado a Exuniq.

El Procedimiento *Excond* ejecuta el programa desde la dirección actual hasta que se cumpla con una condición especificada por el usuario, en caso de no encontrar dicha condición el programa termina normalmente. Esta rutina es usada por la opción CORRE siempre y cuando esté habilitada la opción Condic en la línea de funciones que se activa mediante la función <F7>. Las condiciones de parada pueden ser igualdad o desigualdad de un valor determinado por el usuario con el contenido de un Registro o activación/desactivación de una Bandera.

Excond hace un llamado a Expaso que ejecuta una sola instrucción, luego de la ejecución de cada instrucción se comprueba si se cumple con la condición requerida, de ser así se termina con la ejecución, con el puntero apuntando a la instrucción siguiente en la que se cumplió la condición. La condición puede ser detectada dentro de subrutinas, lazos, decrementos o incrementos de registros en general.

La rutina *AddBrkpt* añade un punto de parada (opción CREA del menú) a una tabla que contiene un número, nombre y dirección de los puntos de parada, la tabla puede contener hasta 255 puntos de parada. Si se trata de crear un punto de parada con un nombre o dirección que ya existe, se borra al anterior y se crea al nuevo. Con rutina *DelBrkpt* se elimina un punto (opción BORRA del menú) de

parada de la tabla, mientras que con *TogBrkpt* se activa o desactiva un punto de parada existente (opción Act/Des del menú).

La rutina *Extotal* corre desde la posición actual del programa que se está depurando hasta encontrar un punto de parada activo, en caso de no existir puntos de parada ejecuta el programa hasta su terminación. Esta rutina es usada por la opción EJECUTA del menú. Su función se basa en insertar una interrupción Ruptura (Int 03h) en cada dirección que indica la tabla de puntos de parada, la interrupción está previamente redefinida para que apunte a la rutina de retorno del depurador. Para el caso que no hayan puntos de parada y el programa se ejecute hasta el fin, antes de entregar el control al proceso hijo se modifica el PSP (desplazamiento 0Ah contiene el valor de IP y desplazamiento 0Ch contiene el valor de CS a donde se salta una vez que termina el proceso hijo) del mismo mediante la rutina *SetTerm* para que al terminar el programa, se entregue el control a la misma rutina del depurador a la que se retorna mediante la interrupción de Ruptura.

Las rutinas *AcortaMimem*, *Insthandler63 y ReduceLevel* preparan las condiciones y ejecutan el proceso hijo, esto se hace al ingresar por primera vez al programa o al realizar un cambio de archivo a depurar.

La rutina *LoadExe* carga el archivo que se ingresó como parámetro al depurador, también se carga el programa cuando se hace un cambio desde el interior del depurador.

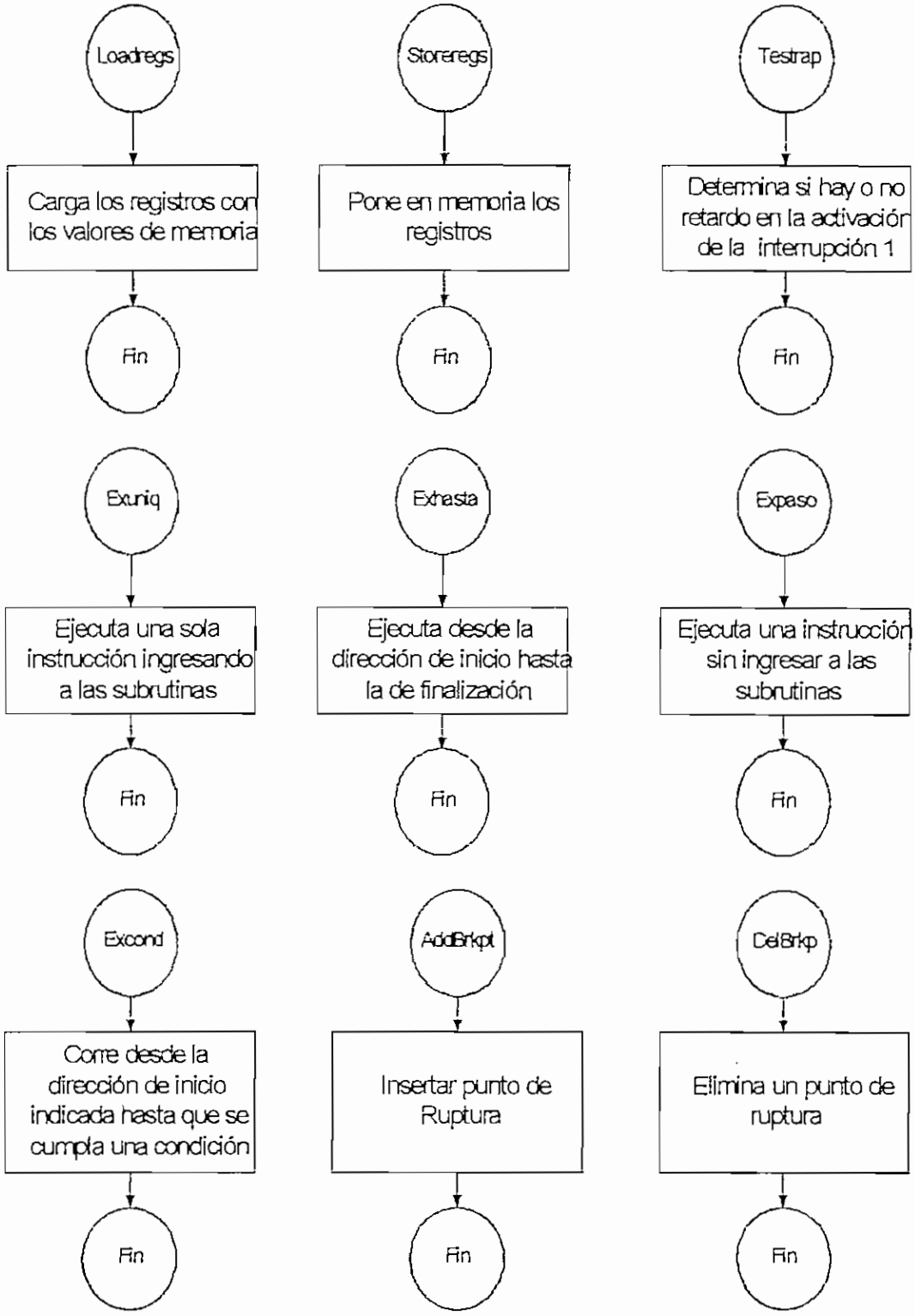


FIGURA 4.7 Diagrama de Bloques del módulo DEPURA.ASM primera parte

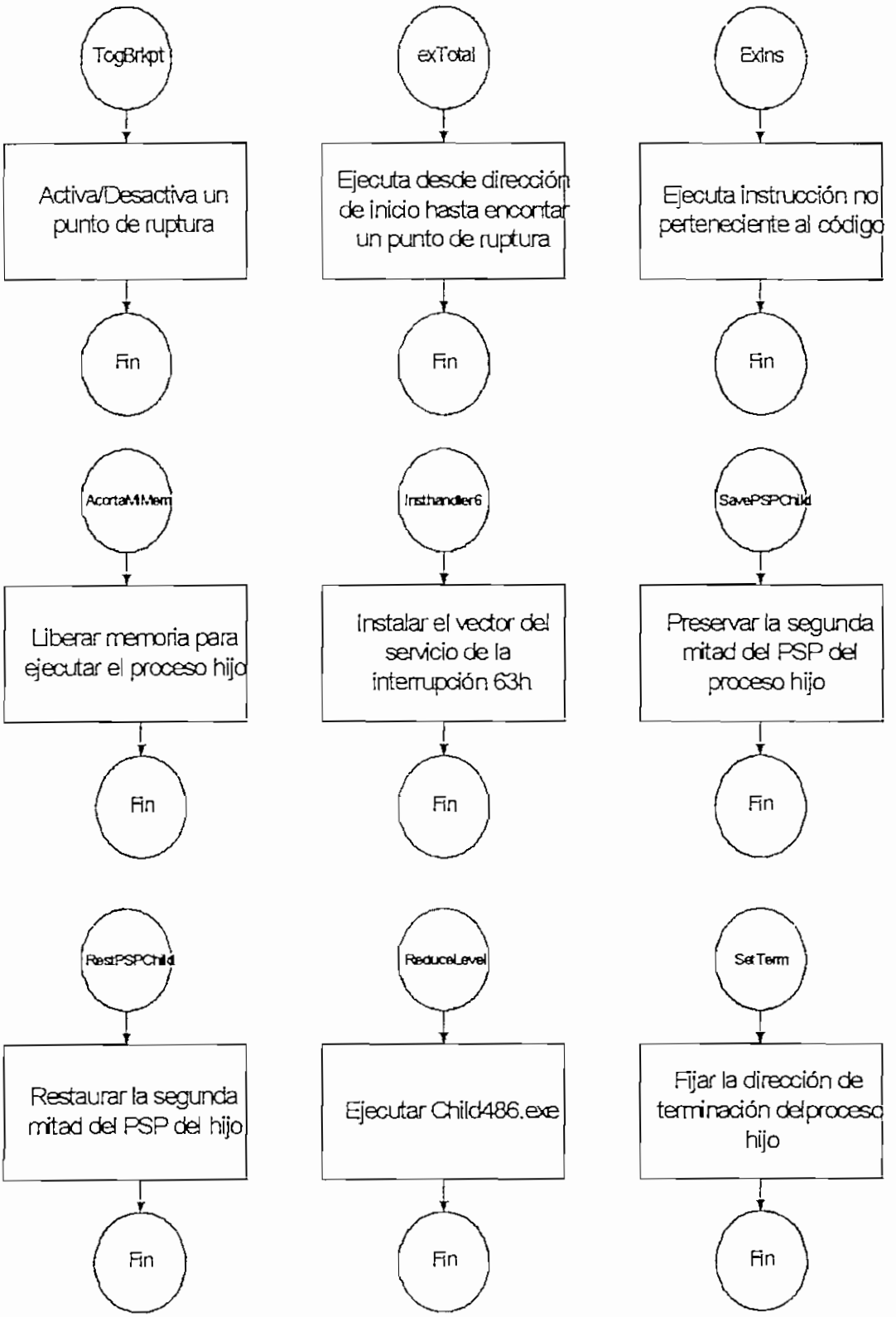


FIGURA 4.7 Diagrama de Bloques del módulo DEPURA.ASM
segunda parte

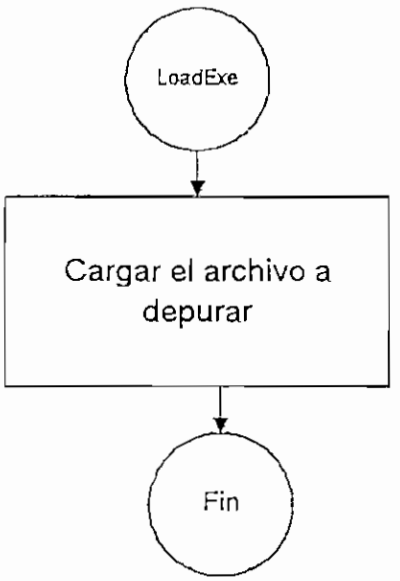


FIGURA 4.7 Diagrama de Bloques del módulo DEPURA.ASM tercera parte

MODULO MEMORIA.- Este módulo está conformado por las rutinas asociadas con las opciones de memoria del menú. A continuación describimos la función de cada uno de estos procedimientos:

Procedimiento Llenar.- Asociado a la opción LLENA del menú, llena un bloque de memoria (especificado en bytes) a partir de una dirección determinada, con el valor ingresado por el usuario. La única función que realiza es mover (mediante la instrucción STOSB) el byte del usuario que está en el registro AL a las localidades de memoria especificadas.

Procedimiento Buscar.- Asociado a la opción BUSCA del menú, busca un string ASCII o Hexadecimal dado por el usuario en todo el segmento (64Kbytes)

que esté actualmente en la ventana de memoria, se realiza mediante la instrucción REPE CMPSB.

Procedimiento Copiar.- Asociado a la opción COPIA del menú, copia un bloque de memoria de un lugar a otro, se debe especificar la dirección inicial de destino del bloque y el número de bytes a copiarse. La dirección de origen es por defecto la del primer byte de la ventana de memoria. Se realiza este trabajo con la instrucción REP MOVSB.

MODULO DESENS.ASM.- Contiene todos los procedimientos para desensamblar instrucciones, esto es a partir del código de máquina obtener el mnemotécnico de la instrucción con sus respectivos operandos que se presentarán en la ventana de código del depurador. El módulo desensambla una instrucción a la vez y requiere como parámetros de entrada que DS:SI apunten al primer byte desde el que se desea desensamblar y ES:DI apunten al área de memoria donde se pondrá la instrucción desensamblada. Como parámetros de salida entrega la longitud de la instrucción en bytes y el tipo de datos con los que trabaja (tipo byte, word, doble word, cuádruple word, etc).

El procedimiento principal de éste módulo es DESENS y es el que básicamente realiza el proceso de desensamblado. Está constituido por varios casos cada uno de los cuales desensambla grupos de instrucciones de similares características en lo relativo al tipo de operandos. Como se indica en el apéndice B, las instrucciones tienen un código de máquina (máximo 14 bytes por instrucción)

con un formato definido y en base al mismo se procede a obtener cada una de las partes de la instrucción como el mnemotécnico (nombre) de la instrucción que además determina si tiene o no parámetros adicionales, tipo de direccionamiento de los operandos (inmediato, directo, por registro, relativo por registros, etc.), tipos de operandos y el orden de los mismos.

Inicialmente se determina si existen bytes de prefijo que indican redefinición de segmento, tamaño de dirección (indica direcciones de 32 bits) y tamaño de operando (indica operando de 32 bits). Por ejemplo la instrucción MOV [SI],AX toma por defecto al segmento DS si se especifica otro segmento diferente a DS, se pondrá antes del OPCODE de la instrucción el prefijo del segmento respectivo como se indica en la figura 4.8:

CODIGO DE REDEFINICION DE SEGMENTO	INSTRUCCION (Ejemplo)	CODIGO DE LA INSTRUCCION (hexadecimal)
DS = 3EH	MOV [SI],AX	89 04 *
ES = 26H	MOV ES:[SI],AX	26 89 04
FS = 64H	MOV FS:[SI],AX	64 89 04
GS = 65H	MOV GS:[SI],AX	65 89 04
SS = 36H	MOV SS:[SI],AX	36 89 04
CS = 2EH	MOV CS:[SI],AX	2E 89 04

Nota: * (como DS es el segmento por defecto no se pone el prefijo de redefinición de segmento)

FIGURA 4.8 Prefijos de redefinición de segmento

El prefijo de tamaño de dirección (67h) se usa para el modo protegido donde las direcciones pueden ser de 32 bits. En modo real se toma por defecto

datos de 8 y 16 bits, si se especifica datos de 32 bits se antepone al código de la instrucción el prefijo de tamaño de operando (66h).

Una vez determinados los bytes de prefijo, se apunta al primer byte de código de operación y se determina su posición en una tabla, en base a esta posición se salta al caso de desensamblado correspondiente a la instrucción.

En el subprocedimiento se lee el nombre (mediante la rutina READARRAY) de la instrucción desde un arreglo de mnemotécnicos asociado a cada caso, luego se procede con la obtención del tipo de operandos que se deben ubicar luego del nombre de la instrucción (de acuerdo al byte de oo-rrr-mmm, ver tablas B-2 a B-7 del apéndice B), para esto existen varios procedimientos auxiliares que se invocan de acuerdo al tipo de operandos los que además entregan en CX la longitud de la instrucción en bytes y en AL un código indicando el tipo de operando (byte, word, doble word, cuadruple word, no definido y temporal, este último solo para el caso de instrucciones del coprocesador matemático) .

Finalmente una vez que se han determinado los componentes de la instrucción se inserta el nombre de segmento respectivo en caso de que haya existido el prefijo de redefinición de segmento.

Otro procedimiento importante es DESENSDET que prepara el formato de la línea de instrucción desensamblada para que sea presentada en la ventana de código del depurador. En la ventana del depurador cada línea de instrucción está compuesta de los siguientes campos:

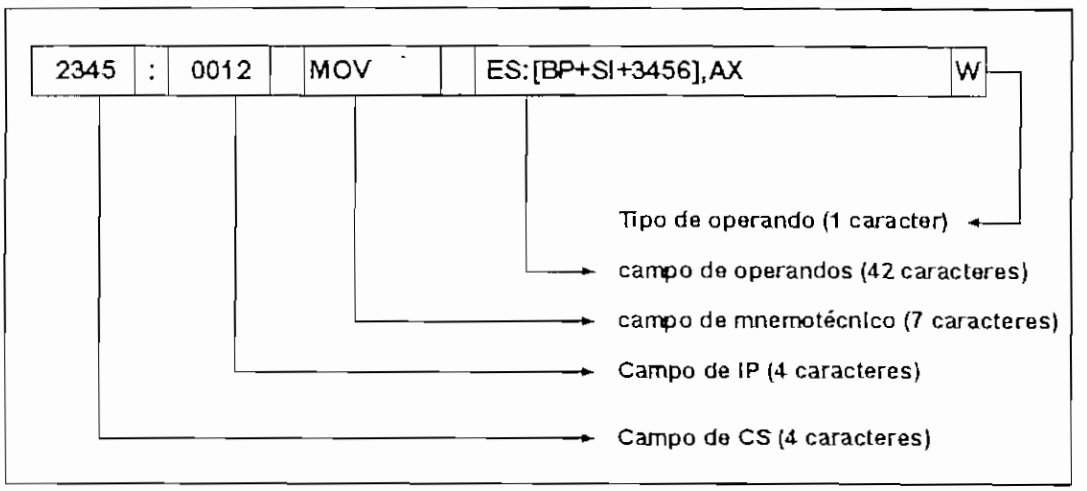


FIGURA 4.9 Línea de instrucción de la ventana de código del depurador

El campo CS es insertado antes de llamar a Desensdet (en la rutina MCOD de Pro486.asm), el campo IP lo pone Desensdet y el resto de campos son obtenidos en Desens.

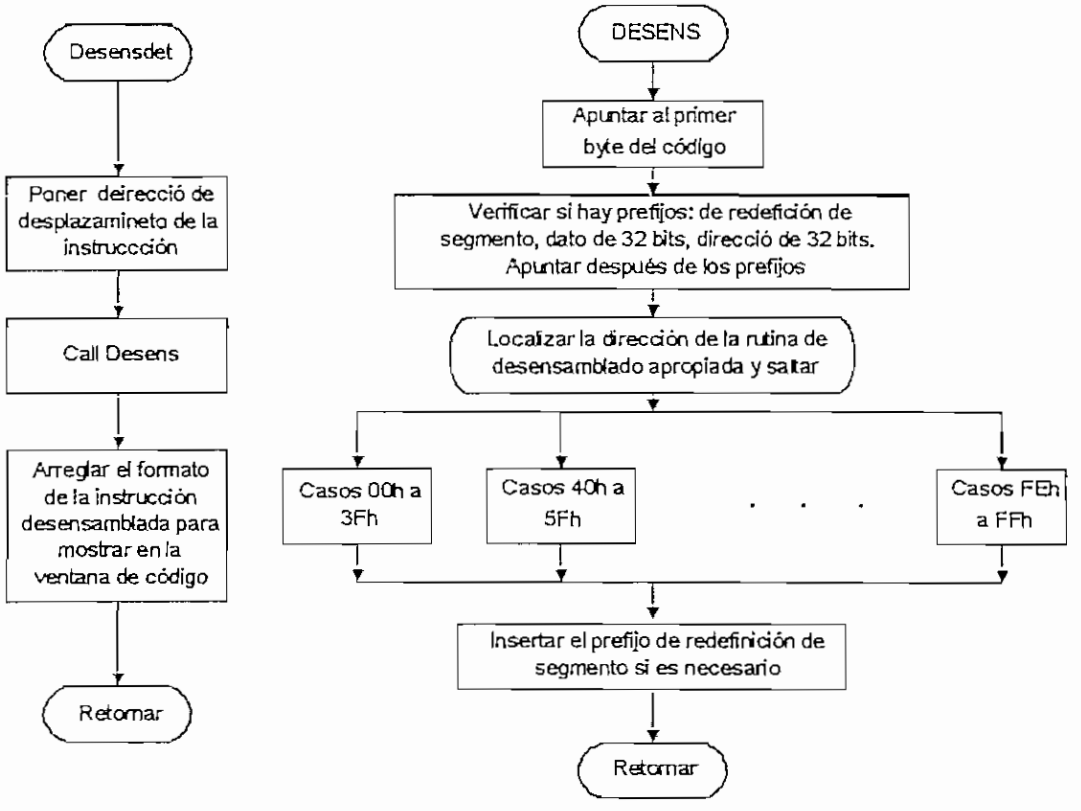


FIGURA 4.10 Diagrama de bloques del módulo DESENS.ASM:

(a) Procedimiento Desensdet . (b) procedimiento Desens

MODULO ENSAMBLA.ASM.- Este módulo convierte un mnemotécnico válido de una instrucción a su código de máquina respectivo. Para cumplir con tal propósito el submódulo ENSAMBLA invoca a dos pasos de compilación denominados Enspaso1 y Enspaso2, el primero genera un "Precódigo" que representa una primera aproximación al código de máquina verdadero, para con el segundo paso obtener el código de máquina verdadero a partir del precódigo obtenido en el primer paso.

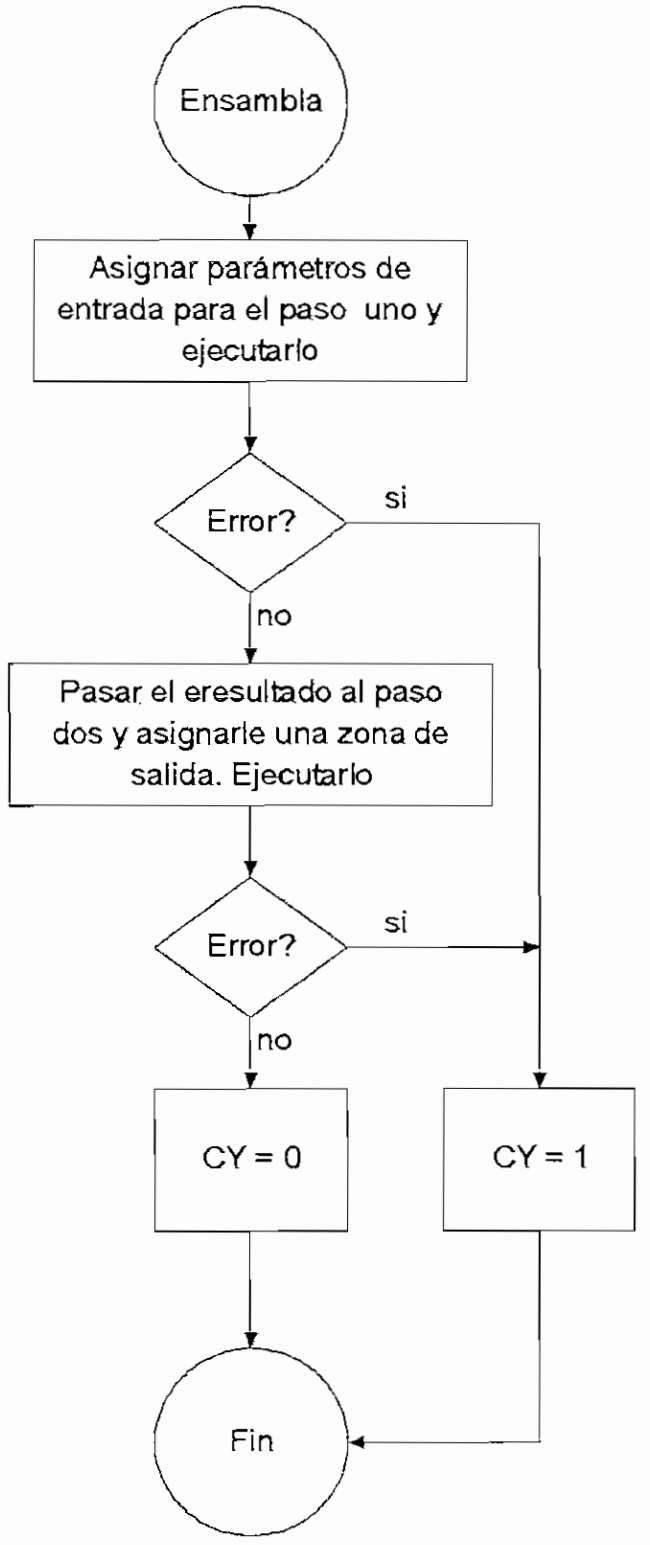


FIGURA 4.11 Diagrama de bloques del procedimiento principal del módulo Ensambla.asm

4.2.3 Flujo del programa, conectividad de los módulos

En esta parte se pretende dar una idea general de la forma como están interconectados los diferentes módulos y archivos que conforman el depurador. En la figura 4.12 se muestra un diagrama que indica como se conectan los módulos entre sí, también se incluyen los archivos adicionales del ensamblador.

El depurador inicia y termina en el módulo INICIAL.ASM, es el lazo principal que controla el menú y el flujo del programa. Este módulo hace uso de los macros del archivo INIMAC..ASM para el manejo del teclado y de pantalla.

INICIAL.ASM se conecta en forma directa solo con el módulo PRO486.ASM, al que entrega el control para que se ejecute la rutina respectiva una vez que se ha elegido una opción en el menú. La mayoría de las rutinas de Pro486.asm se encargan de solicitar datos adicionales necesarios al usuario y preparar las condiciones para posteriormente hacer un llamado a rutinas de otros módulos que realmente realizan la función específica.

PRO486.ASM hace uso de los macros del archivo PROVNT.ASM para controlar al archivo de ventanas WINCOLOR.VNT y WINBN.VNT para monitor a color y blanco/negro respectivamente, también usa los macros del archivo MPRO.ASM para el control de teclado, de la pantalla y para todas las funciones relativas al archivo de macros PRO486.MCR como abrir, cerrar, leer, cargar los macros en memoria, etc.

Tanto PRO486.ASM como INICIAL.ASM hacen uso de los macros del archivo MOUSE.ASM, relacionados a cada una de las funciones de la interrupción asociada al ratón.

Otros archivos que son directamente controlados por PRO486.ASM y que no se indican en el diagrama de conectividad son los programas EDIT.COM, MASM.EXE, LINK.EXE y PRO486.HLP. Se encarga de solicitar al usuario los parámetros que requieren estos archivos y luego los ejecuta.

El proceso de reducir de nivel se realiza en el módulo DEPURA.ASM, que usa el archivo CHILD486.EXE para lograr dicho propósito. Este módulo es el que controla directamente al programa de usuario al momento de cargarlo en memoria (mediante el procedimiento Loadexe). En el diagrama de conectividad se observa que una "copia" del contenido del archivo que se va a depurar se sobrepone en el espacio de memoria de Child486.exe y son estos datos los que se utilizan para depurar al programa.

Para procesos de manejo de memoria de la computadora PRO486.ASM entrega el control al módulo MEMORIA.ASM, este módulo no controla a ningún archivo adicional. Por el acceso directo a cualquier localidad de la memoria RAM de la computadora que realiza este módulo es importante que el usuario este seguro al momento de realizar variaciones en las localidades en memoria por que se puede alterar información de programas residentes y del sistema que pueden inhabilitar a la computadora obligando a que se la reincialice.

DESENS.ASM es otro de los módulos importantes, es el que desensambla el código del programa que se está depurando para ser presentado en la ventana de código del depurador. Este módulo no tiene conexión directa con el archivo que se está depurando debido a que este módulo toma el código a desensamblar desde la información del archivo cargada en memoria.

El último módulo que se conecta con PRO486.ASM es ENSAMBLA.ASM que ejecuta la instrucción directamente ingresada por el usuario sin necesidad de que esta conste en el código del programa, solamente se da la posibilidad de ensamblar y ejecutar las instrucciones más comunmente usadas.

Cada uno de los módulos a excepción de INICIAL.ASM devuelven el control al módulo inmediato anterior que le entregó el control, siempre que se ha terminado totalmente un proceso el control final debe tomarlo el controlador de flujo INICIAL.ASM.

Todos los módulos conforman un solo archivo denominado PRO486.EXE, para obtener este archivo se deben ensamblar todos lo módulos independientemente para crear los archivos objeto (archivos .OBJ) con el uso de cualquier ensamblador que disponga, en esta tesis se ha usado el ensamblador MASM 6.11 de Microsoft. Una vez que se disponga del los archivos objeto, estos deben ser enlazados en conjunto como se indica a continuación :

```
LINK PRO486.OBJ+INICIAL.OBJ+DEPURA.OBJ+ENSAMBLA.OBJ+DESENS.OBJ+MEMORIA.OBJ
```

la extensión OBJ no es indispensable en el listado anterior, el orden tampoco es de importancia, el nombre por defecto para el archivo ejecutable se toma del primer archivo .OBJ, pero el enlazador realiza una petición de aceptación o de cambio del mismo.

El orden en que estos archivos son organizados en el archivo ejecutable está dado por la forma como han sido utilizados durante el desarrollo del programa.

Una rutina debe ser declarada como PUBLIC para que pueda ser utilizada en otros módulos, y en el módulo donde va ha ser usada deberá indicarse que es externa mediante la directiva EXTRN. El llamado a una rutina de otro módulo se hace mediante la instrucción CALL donde el retorno será automático luego de una instrucción RET, también se puede llamar mediante un salto con JMP en tal caso el programador debe controlar el retorno adecuado al módulo origen.

Los archivo de macros no requieren ser ensamblados independientemente, pero deben ser incluidos mediante la directiva INCLUDE Nombre_archivo en el módulo que los va ha usar. Al momento de ensamblar el módulo, los macros son automáticamente incorporados en el archivo objeto del módulo.

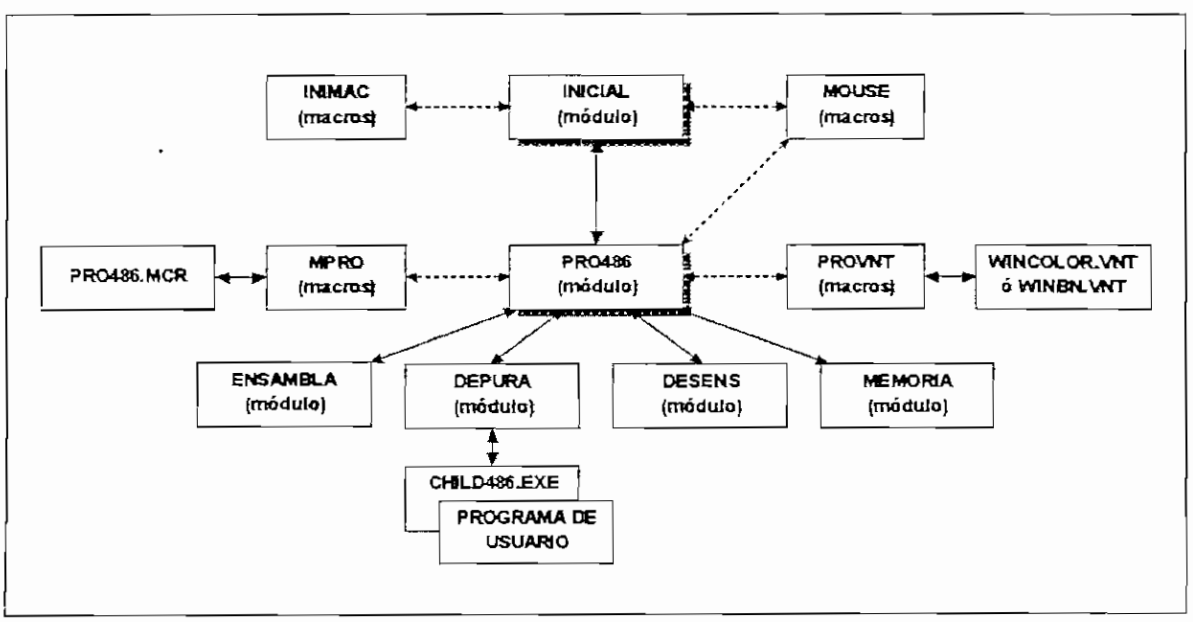


Fig. 4.12 Diagrama de conectividad de los módulos de PRO486.EXE

4.3 Uso del Ratón en lenguaje ensamblador

El ratón es otro de los dispositivos periféricos de entrada de datos a la computadora, que facilita el trabajo en ambientes en los que se tienen diferentes opciones sobre la pantalla (por ejemplo ambiente gráfico de Windows).

En el caso del depurador se ha considerado el uso del ratón para hacer más rápido el acceso a las ventanas del depurador y en cierta forma para dar mayor facilidad cuando se accesa a las opciones de menú y funciones.

El manejo del ratón es similar al de otros dispositivos de la computadora, al igual que el teclado o la pantalla de vídeo. El ratón tiene asociado una interrupción software a través de la cual el usuario maneja el flujo de información desde y hacia el ratón. La interrupción asociada con el ratón es la 33h (51 en decimal), antes de usarla es necesario que se disponga de un ratón conectado a uno de los puertos seriales de la computadora y de un controlador de ratón instalado (software), caso contrario pueden producirse errores debido a que el vector de interrupción no estará debidamente inicializado.

Tal como las interrupciones de teclado, vídeo y otros dispositivos, la interrupción del ratón tiene varias funciones (de acuerdo al valor en AX) que determinan el tipo de trabajo que debe realizar la interrupción.

Por medio de la interrupción del ratón el usuario obtiene información de la posición del puntero del ratón, si se presionó un botón, si ya fue liberado, cuántas veces se pulsó desde la última vez que se acceso a la información, etc. Con esa información se debe determinar que acción hay que tomar, similar a cuando se diseña un menú con selección de opciones mediante el teclado, el programador lo único que espera obtener es el código de una tecla asociada a una opción del menú,

se analiza ese código y una vez determinado a que opción del menú corresponde se ejecuta la rutina correspondiente. Con el ratón en cambio se debe determinar en que posición está el puntero del ratón , si se presionó o liberó uno de los botones del ratón, o si simplemente se lo posicionó en tal sitio, en base a eso se debe determinar si se tiene alguna aplicación asociada a una de estas acciones.

Existen varias funciones para la interrupción del ratón, dependiendo de la marca del ratón se tienen en menor o mayor número. En general existe un número de funciones consideradas básicas y que se incluyen en la gran mayoría de controladores de ratón y son las que se mencionará en el presente estudio. Igualmente lo referente a los botones no está completamente definido, algunas versiones tienen habilitados los tres botones mientras que otras sólo disponen de los botones de los extremos, por lo que previo al uso del ratón se debe investigar las características y facilidades de cada ratón en base a la documentación respectiva o mediante pruebas. A continuación se describen las funciones del ratón para el caso de modo texto, para el modo gráfico son similares, excepto la que determina la forma del cursor :

- AX = 00h .- Llama al controlador del ratón, reseteandolo a las condiciones iniciales y retorna el estado del ratón en AX. Un valor de cero en AX indica que no existe ratón y un valor de -1 (FFFFh) indica que si existe ratón. En BX se entrega el número de botones que dispone el ratón. En el caso de no existir el ratón se aconseja no realizar llamadas a la interrupción 33h.
- AX = 01h.- Hace visible al puntero del ratón que por si solo se desplaza por la pantalla sin necesidad de control del usuario. No requiere otros parámetros de

entrada ni entrega parámetros de salida. Al resetear el ratón el puntero queda inicialmente oculto.

- AX = 02h.- Oculta el puntero del ratón, se usa en conjunto con la función anterior en los casos en los que se guardan temporalmente pantallas de vídeo para ser restablecidas posteriormente, antes de guardar la pantalla se debe ocultar el puntero y se lo hace visible cuando se haya puesto la siguiente pantalla, con esto se evitan efectos indeseables en las pantallas del programa. Se debe tener en cuenta que cada llamada a esta o a la función anterior es acumulativa, esto quiere decir que si se llama cinco veces consecutivas a la función que muestra el puntero, se debe llamar seis veces a la función ocultar el puntero para que éste se oculte. El ocultamiento del ratón "no" afecta las operaciones de seguimiento de su posición y de operación de los botones.
- AX = 03h.- Esta función reporta la posición del puntero del ratón (en el registro CX la coordenada horizontal y en DX la coordenada vertical) y el estado de sus botones (en el registro BX), no requiere parámetros de entrada. Las coordenadas de la posición son reportadas en pixeles, en modo texto siempre se alinean a las coordenadas de inicio de la máscara del carácter sobre el que está el puntero. Los posibles valores para BX son :

  00h no se ha presionado ningún botón

  01h Se ha presionado el botón izquierdo (LSB de BX es 1)

  02h Se ha presionado el botón derecho (segundo bit de BX es 1)

  04h Se ha presionado el botón del centro (tercer bit de BX es 1)

La correspondencia de los bits con los botones del ratón no siempre es como está indicado, aunque la mayoría de veces si se cumple.

- AX = 04h.- Mueve el cursor del ratón a una posición específica de la pantalla. Las coordenadas usadas son absolutas, en pixeles. Las coordenadas se pasan en CX y DX para el eje x y el eje y respectivamente. En modo texto el cursor siempre se alineará a la malla del carácter más cercano a las coordenadas indicadas. No entrega parámetros de salida.
- AX = 5.- Retorna información acerca del botón requerido. En BX se entrega el número del botón del que se requiere la información (0 = izquierdo, 1 = derecho, 2 = centro). Luego de la ejecución de la interrupción se obtiene en AX el estado del botón (0 = liberado, 1 = presionado), en BX el número de veces que se ha presionado el botón desde la última llamada a esta función, y en CX y DX las coordenadas horizontal y vertical en pixeles del puntero del ratón al momento de presionar el botón. Si no se ha presionado el botón todos los registros retornan en cero. Para usar el ratón con doble click es conveniente insertar un retardo de 300 a 600 milisegundos entre cada llamada a esta función para dar tiempo a que se pulse dos veces el botón, este conteo se entrega en BX.
- AX = 06h.- Retorna información de liberación del botón requerido. En BX se entrega el número del botón que se requiere información. Después de ejecutar la interrupción se obtienen los mismos parámetros que en la función anterior con la única diferencia que las coordenadas de la posición del puntero están dadas al momento en que se liberó el botón del ratón.
- AX = 07h.- Determina los límites horizontales mínimo y máximo dentro de los cuales se puede mover el puntero del ratón. En CX se ingresa el valor mínimo y

en DX el valor máximo, ambos en pixeles. No se tiene ningún parámetro de salida.

- AX = 08h.- Similar a la función anterior pero para el sentido vertical. Si los parámetros no son adecuados (CX > DX), no tiene efecto la función.
- AX = 0Ah.- Determina la forma del puntero del ratón. En modo texto existen los punteros tipo Hardware y tipo Software (BX = 0 indica puntero software y BX = 1 indica puntero tipo hardware). El puntero tipo hardware es similar al cursor de la pantalla de vídeo, en CX se indica la línea de inicio del puntero y en DX la línea final dentro de los límites de la malla de un carácter. En este caso el puntero es rectangular, para monitor monocromático el rango va de 0 a 7 y para CGA de 0 a 14 líneas. Para el puntero "software" los bits de la máscara de pantalla (valor en CX) hacen un AND lógico con el carácter existente en la pantalla, éste resultado hace un XOR lógico con la máscara del puntero (valor en DX) y el resultado es el que se presenta en pantalla. Cada bit de la palabra de máscara de pantalla y de cursor representa lo siguiente:

Bit	Descripción
0..7	Código de carácter ASCCI extendido para la forma del cursor
8..10	Color de presentación (Ver atributos para color y B/N)
11	Intensidad de brillo (1 = alta, 0 = baja)
12..14	Color de fondo(ver atributos)
15	Parpadeo (1), sin parpadeo (0)

Valores recomendados para la máscara de pantalla son: 7F00h, FFFFh o en general los 8 bits menos significativos deben ser todos Uno o todos Cero. Para la máscara del puntero los 8 bits superiores deben ser cero, con los 8 menos significativos de acuerdo a la forma del puntero deseada. Para obtener un puntero transparente, esto es que se vea el carácter que se halla debajo del puntero, se recomienda la máscara de pantalla 77FFh y la máscara de puntero 7700h. No existen parámetros de salida.

- AH = 0Bh.- Reporta el movimiento en pasos del puntero desde la última llamada a esta función, los valores de salida indican el número de pasos que se ha movido el puntero en el sentido vertical (valor en DX) y horizontal (valor en CX) desde la última llamada a esta función. Para los ratones más antiguos cada paso representa 1/100 de pulgada (100 mikey/pulgada), los más actuales tienen 1/200 pulgada por paso (200 mikey/pulgada), hasta 320 mikey/pulgada para ratones de alta resolución. Estos datos se usan únicamente en ciertas aplicaciones especiales.
- AX = 0Dh, AX = 0Eh.- Activar/Desactivar la emulación del funcionamiento de lápiz óptico, no se requieren de parámetros de entrada ni se entregan parámetros de salida.
- AX = 0Fh.- Indica tanto en el sentido vertical como horizontal cuántos pasos/pulgada requiere moverse físicamente el ratón para que el puntero se desplace 8 pixeles en la pantalla . En CX se indica el valor para la dirección horizontal y en DX para la dirección vertical. Para un ratón de 200 mikey/pulgada se recomienda usar un valor de 8 Mikey/8pixel para CX, 16

Mikey/8pixel para DX, con esto el puntero cubre la pantalla de 600x200 pixeles al rodar en un espacio de 3,2 x 2,0 pulgadas. No existen parámetros de salida.

- AX = 10h.- Selecciona un cuadro en la pantalla en la que el ratón se oculta automáticamente, tiene efecto acumulativo al igual que la función 02h. Se desactiva con la función 01h. Los límites en pixeles izquierdo, superior, derecho e inferior del cuadro se pasan en CX, DX, SI, y DI respectivamente.
- AX = 13h.- Determina una velocidad umbral del puntero a partir de la cual se aplica una aceleración a su movimiento. En algunos ratones la aceleración aplicada es constante y aumenta la velocidad al doble, mientras que en otros se aumenta la velocidad de acuerdo a una curva de aceleración dada por el fabricante. En DX se pasa el valor de la velocidad umbral en Mikey/segundo, los valores permitidos van de 0 a 7FFFh pero un valor promedio es 300 mikey/segundo. El efecto de aceleración puede ser inhabilitado poniendo un valor umbral de 7FFFh o rehabilitado poniendo un valor de cero.

Para usar el ratón en el depurador se ha creado un archivo de macros (MOUSE.ASM, que se lista en el apéndice), en el que los parámetros se pasan o se reciben en variables tipo estructura, esto facilita el manejo de las funciones. También en estos macros se han incluido un control para evitar la ejecución de las funciones en caso de no existir el ratón, de igual forma se controla para evitar ejecuciones sucesivas de la función 01h (ocultar ratón) al igual que la función 02h (mostrar puntero). La función 00h (resetear) se ha implementado de forma que si el ratón está presente se ejecuta automáticamente la función para mostrar el ratón.

Para controlar lo que debe hacer el programa de acuerdo a las acciones del ratón se ha implementado la rutina PANTALLA ubicada en el módulo

INICIAL.ASM. El controlador de flujo se mantiene en un lazo que primero verifica si existe un dato en teclado, de no ser así llama a la rutina Pantalla para ver si se ha elegido alguna opción por medio del ratón.

La pantalla se la ha dividido en líneas o grupos de líneas a las que se asocia un arreglo de valores que indican los límites de los campos de las mismas, por ejemplo a la línea uno correspondiente al menú se le asocia el siguiente arreglo :

```
linea0  db      4,12,20,28,36,44,52,60,68,76,0
```

donde cada número indica el límite de una opción del menú, la primera opción (..) va de la columna 0 a la columna 4, las demás opciones ocupan 8 espacios cada una, el valor 0 al final indica terminación de opciones. Para el caso de líneas en las que existen ventanas se asignan arreglos con los límites de las ventanas.

En la rutina Pantalla lo primero que se hace es dividir la coordenada vertical dada en pixeles para la altura de un carácter (que depende del modo de vídeo y debe determinar el programador) para hallar la línea en la que se encuentra el puntero del ratón. Para la mayor parte de las regiones de la pantalla la única acción que interesa es que se haya pulsado el botón izquierdo, únicamente en la línea de menú interesa que el puntero este posicionado allí o que se pulso el botón izquierdo, en cambio en otras áreas no tiene efecto ninguna acción del ratón. De acuerdo a la línea en la que este ubicado el puntero se verifica si la acción realizada por el ratón es útil, de no ser así se termina la rutina con el valor FFh en BX que indica que no se ha seleccionado ninguna opción. En caso que la acción realizada sea válida se apunta con SI al arreglo correspondiente a la línea y se llama a la

rutina SCANLINEA que devuelve en BX el valor del campo de la línea sobre la que está el puntero y de acuerdo a este valor se envía el código de tecla respectivo en AX, y en BX se pone 00h para indicar opción válida. Por ejemplo si se hace click en la primera opción del menú, primeramente la rutina pantalla determina que el puntero se halla en la línea uno, luego verifica que se ha pulsado el botón izquierdo y que esta acción es válida para esta línea, entonces llama a Scanlínea que devolverá el valor 01 correspondiente a la posición de la primera opción, en base a este valor se pone en AX el valor 1C0Dh que es el código de teclado correspondiente a ENTER y en BX el valor 00h, y se continua con la ejecución del controlador de flujo como si el dato en AX hubiese sido obtenido desde teclado.

Existen controles adicionales, uno de ellos es por ejemplo cuando se está editando en las ventanas del depurador no se debe enviar en ningún caso el código de ENTER.

En la figura 4.13 se presentan los diagramas de flujo de las rutinas para el control del ratón.

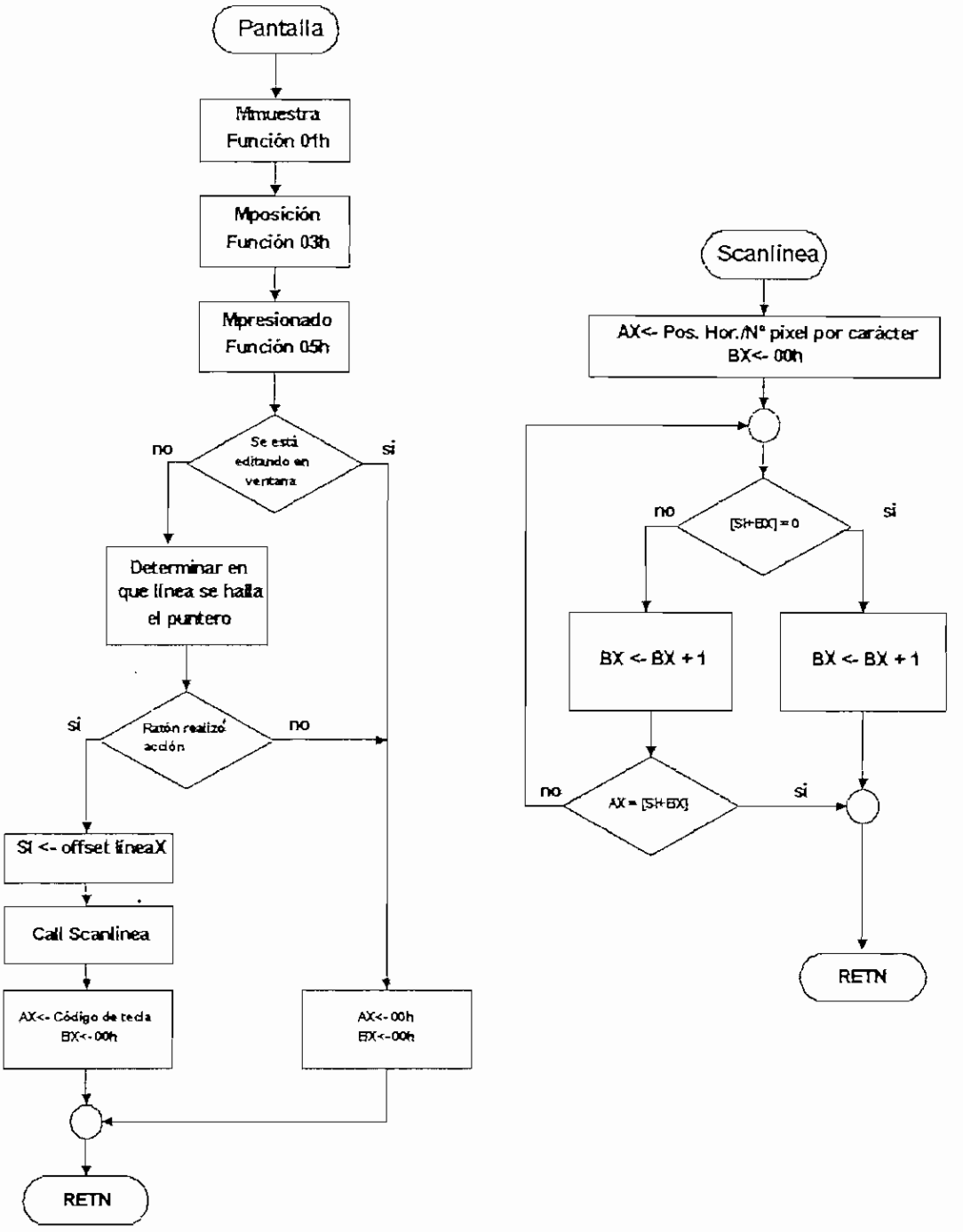


FIGURA 4.13 Diagrama de Flujo de las rutinas de control del ratón

4.4 Ejemplos de depuración de Programas

PRURAT.EXE.- Este programa ilustra el uso del ratón en lenguaje Ensamblador, hace uso del archivo de macros que incluye las funciones del manejo del ratón.

El método de borrar pantalla se basa en escribir directamente en las 2000 localidades en memoria de la página de vídeo 0 (dirección de inicio B800H para color), el atributo normal (07h) y el carácter espacio en blanco (20h). La rutina para borrar la pantalla se encuentra en el módulo PASBORRA.ASM que debe ser ensamblado independientemente y se debe enlazar junto con el archivo objeto de Prurat.asm.

El programa también presenta información del modo de vídeo que se obtiene mediante la función 0fh de la interrupción 10h del BIOS, que devuelve en BH la página activa , en AH el modo de vídeo y en AL el número de columnas. Estos datos deben ser transformados a base decimal y luego a formato ASCII para presentarlos en pantalla.

Otro método fácil para cambiar los atributos de vídeo, que también se usa en este programa, es mediante la función 07h de la interrupción 21h del DOS.

Mediante la rutina CUADRO se dibuja un cuadro de dos caracteres por lado, que en adelante se lo llamará "cuadro de salida", por que en esta área deberá hacer click el usuario para abandonar el programa.

Para presentar la información de la posición del puntero del ratón sobre la pantalla, se hace uso de la función 03h (macro Mposicion) de la interrupción para el ratón (33h), la información entregada debe ser transformada a decimal y luego a ASCII para presentar en pantalla, esto se realiza repetitivamente en un lazo.

En el lazo se comprueba si se ha presionado el botón izquierdo del ratón mediante la función 05h (macro Mpresionado), de ser así se cambia de color el área del cuadro de salida, luego se verifica si se ha liberado el botón mediante la función 06h (macro Mliberado), en cuyo caso se pasa a comprobar si se lo hizo sobre el área del cuadro de salida, para abandonar el programa.

También se sale del lazo del programa cuando el usuario presiona cualquier tecla, para comprobar esto, se hace uso de la función 01h de la interrupción 16h que lee el buffer del teclado si es que existen datos, caso contrario continua con la ejecución del programa.

Antes de salir del programa, se restablece el cursor que se lo ocultó para evitar el efecto indeseable que produce cuando se escribe repetitivamente las coordenadas del cursor en el lazo del programa, se oculta el puntero del ratón para que no quede visible al salir del programa, se borra la pantalla y finalmente se termina el programa con la función 4Ch de la interrupción 21h del DOS.

El listado del archivo PRURAT.ASM se encuentra en el apendice G.

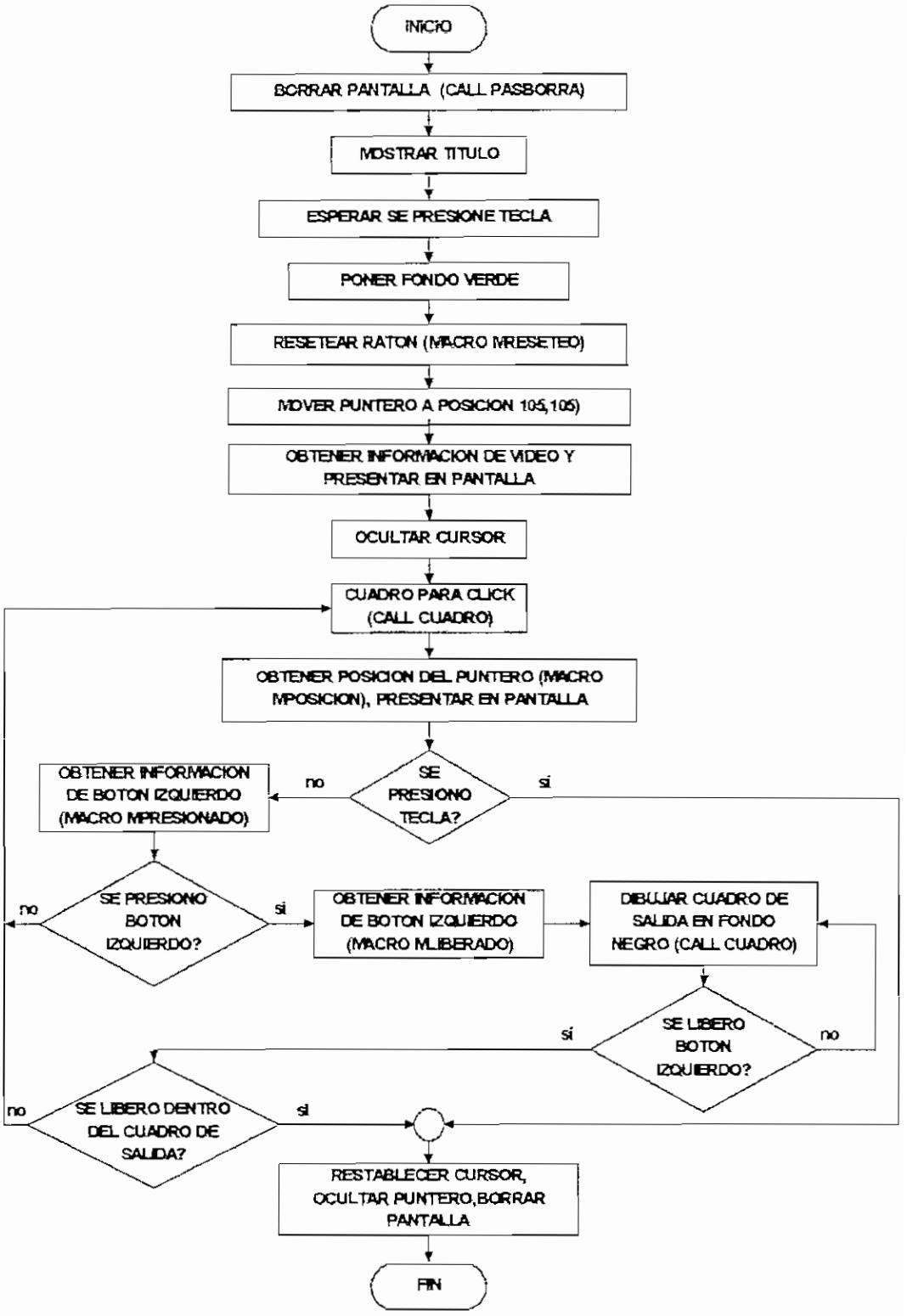


Fig. 4.14 Diagrama de bloques del ejemplo PRURAT.ASM

FUNCION.EXE.- Este ejemplo se lo elaboró para ilustrar el uso de las instrucciones del coprocesador matemático, utilización de macros y procedimientos en lenguaje ensamblador. Este programa obtiene las funciones seno, coseno y tangente de un ángulo (.000° - 90.0°) en grados ingresado desde teclado por el usuario.

Inicialmente se borra la pantalla y se presenta el menú principal que contiene las opciones para el cálculo de seno, coseno, tangente y salir del programa. Si no se seleccionó la opción salir, se pasa a recibir el ángulo mediante el macro Readnumero al cual se le pasa como parámetros la dirección donde se grabarán los datos y el número de datos a leerse, analiza si se ingresó un dato válido.

Una vez que se tiene el ángulo almacenado en la variable Num1, se lo convierte a base hexadecimal sin considerar el punto decimal, esto es, se tiene un número entero para posteriormente llamar al macro Arith. En este macro se carga el ángulo a la pila del coprocesador (ST(0)), este ángulo es convertido a radianes y dividido para una constante adecuada para restablecer el punto decimal que anteriormente fué ignorado.

La conversión a radianes es necesaria para obtener las funciones seno, coseno y tangente del ángulo, se multiplica el resultado por una constante (1000000) para convertirlo a un valor entero, permitiendo así sacar de la pila el resultado en un valor entero, se usa este artificio por que el coprocesador entrega datos en formato temporal, cuyo proceso de decodificación a un valor decimal es complicado. Se multiplica por 1000000 porque la instruccion que se utiliza

únicamente saca la parte entera del valor de la pila, como las funciones seno y coseno generalmente tiene parte entera y decimal, se recorre el punto decimal 6 lugares a la derecha con el propósito de obtener una mejor presición.

Finalmente el resultado obtenido en hexadecimal es convertido a un valor decimal para presentar tal resultado en pantalla. Debido a que en la pila al resultado final se lo multiplicó por 1000000 entones es necesario insertar el punto decimal 6 posiciones a la izquierda del resultado obtenido.

Se utilizó la instrucción FSINCOS que calcula la función seno (resultado se almacena en ST(0)) y coseno (resultado se almacena en ST(1)) a la vez.

Una vez presentado el resultado, se retorna al menú principal para continuar con el programa.

El listado de este archivo FUNCION.ASM se encuentra en el apendice G.

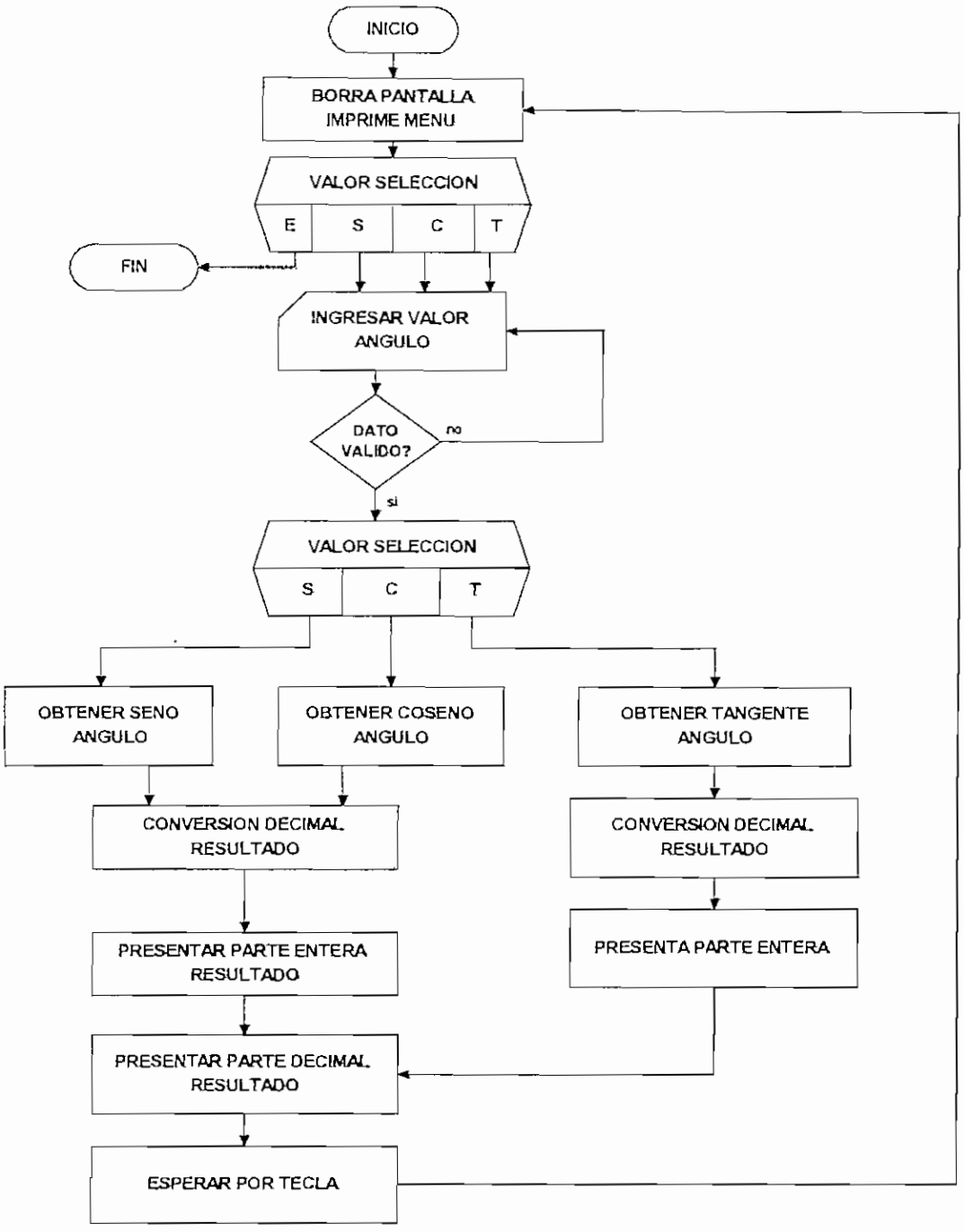


FIGURA 4.16 Diagrama de bloques, ejemplo FUNCION.ASM

Capítulo 5

CONCLUSIONES

Pro486.exe es un depurador de programas para sistema MS-DOS, por lo que su ambiente de trabajo será el modo real (1 Mbyte de memoria), destinado a depurar programas desarrollados en lenguaje Ensamblador que no incluya el cambio a modo protegido esto debido a que va enfocado el paquete para fines didácticos y aprendizaje en la depuración de programas como también a la investigación de las características de los microprocesadores no más allá del modo real que es el modo en el que por defecto trabajará el usuario y sería innecesario para dichos propósitos el abarcar toda la capacidad en memoria de la que se dispone en un 80486 logrando mas bien el implementar muchas facilidades en el modo real en forma más sencilla y rápida sin necesidad de realizar tareas engorrosas en detrimento de la rapidez y sencillez con la que se realiza si únicamente se lo limita al modo real.

Las ventajas del microprocesador 80486 son la incorporación del coprocesador matemático en el mismo chip, que es compatible con versiones anteriores de coprocesadores, 8Kbytes de memoria caché interna, mayor velocidad en la ejecución de las instrucciones, mayor número de instrucciones para mayor facilidad en la programación, un microprocesador completo de 32 bits que simplificó la operación con datos en un formato extendido. Tiene total compatibilidad hacia arriba con anteriores versiones de microprocesadores que permite que el software realizado para dichos microprocesadores también funcione

en un 80486, mientras que programas realizados utilizando instrucciones exclusivas del 80486 no correrán en versiones anteriores.

La programación en lenguaje ensamblador permite realizar programas más rápidos y de menor extensión comparados a los programas ejecutables que se obtienen con los lenguajes de alto nivel. Por otro lado la programación en lenguaje ensamblador requiere que el usuario tenga un conocimiento básico del funcionamiento del microprocesador, la forma como está organizada la memoria y bases del funcionamiento de los dispositivos de la computadora. Actualmente la programación en lenguaje ensamblador sirve de complemento a los lenguajes de alto nivel, para realizar especialmente rutinas de manejo directo de los dispositivos de la computadora, para mejorar la velocidad de ejecución de los programas.

El trabajo con el ratón en lenguaje ensamblador es similar que en los lenguajes de alto nivel, la diferencia radica en la mayor facilidad que prestan éstos últimos para realizar comparaciones y ejecución de rutinas de acuerdo al valor de una variable. El acceso a la información del ratón es en base a una interrupción, igual a como se manejan los demás dispositivos de la computadora (teclado, vídeo, etc.). El ratón puede ser usado en modo texto y en modo gráfico, siendo las funciones similares, en modo texto la posición del puntero siempre se alinea a las coordenadas de inicio (en pixeles) del carácter más cercano, cosa que no ocurre en modo gráfico.

En el caso que se requiera redefinir vectores de interrupción para que ejecuten rutinas diseñadas por el usuario, es recomendable usar las interrupciones de usuario (30h en adelante), y verificar que la interrupción seleccionada no esté siendo usada por programas residentes en memoria, como por ejemplo la

interrupción 33h que es usada por el controlador del ratón. Siempre se debe guardar la dirección a la que estaba apuntando el vector que va a ser alterado, de forma que antes de terminar con la ejecución del programa, se restablezcan los vectores afectados.

El coprocesador está conformado por una pila de ocho registros en los que se almacenan los datos para las diferentes operaciones, dicha pila puede ser accesada de la forma clásica, esto es, el último en entrar es el primero en salir, o también se pueden accesar directamente para colocar o sacar un dato de un determinado registro. Existen otros registros adicionales que indican el estado del coprocesador. Los registros del coprocesador solo pueden tomar o poner datos directamente desde memoria, por esta razón si el procesador quiere comunicarse con el coprocesador matemático debe grabar la información en memoria y posteriormente se pasará a los registros del coprocesador, por otro lado si quiere obtener información del coprocesador, primero se almacenan los registros del coprocesador requeridos en memoria y de aquí a los registros del procesador.

El coprocesador matemático realiza las operaciones aritméticas a mayor velocidad que el procesador esto en cuanto a trabajo con operandos enteros. Los valores almacenados en la pila están en formato real temporal IEEE (80 bits), existen instrucciones que permiten sacar directamente valores enteros binarios o enteros en formato decimal empaquetado. Para el caso de valores reales el ensamblador se encarga de codificar a formato real temporal los valores reales definidos en el segmento de datos, los resultados entregados por el coprocesador pueden ser sacados en los formatos real corto, real largo y real temporal, siendo el programador el encargado de convertirlos al formato decimal.

Para presentar los datos de la pila del coprocesador matemático en formato decimal, se utilizó el estándar de punto flotante de la IEEE actuando así para la descodificación de los datos de la pila sobre el formato real temporal (80 bits) que como se mencionó anteriormente es el formato en el que se encuentran los datos en el interior de la pila del coprocesador, razón por la que los valores en formato decimal que se presentan de la pila, si bien son bastante exactos, no es directa su lectura y es necesario realizar operaciones adicionales por el usuario para obtener el resultado final en formato real, esto también debido a que desgraciadamente , no hay proceso de descodificación directo por parte del procesador para números que son devueltos por el coprocesador , por que no es un proceso comparable a ensamblar cuando se ejecuta un programa. Existen librerías que permiten la descodificación de los datos de la pila a un formato real, pero su utilización es compleja por las varias condiciones que debe cumplirse antes de invocar a tales librerías además de que su uso no es muy común por lo que es difícil conseguirlas a menos que se compre el paquete que las incluye, un ejemplo de estas librerías es IBMUTIL.LIB que es suministrado por IBM en su paquete ensamblador.

BIBLIOGRAFIA

1. Barry Brey; "Los Microprocesadores Intel 8086/8088, 80186, 80286, 80386, 80486" Prentice Hall 1994.

2. William Murray y Chris Pappas; "Programación en Lenguaje Ensamblador 80386/80286" OSBORNE/McGraw-Hill 1990.

3. Lance Leventhal; "Guia de Programación 80386" Macrobit Editores 1987.

4. Microsoft; "Assembly -Language Development System MASM Version 6.11" Microsoft Corporation 1993.

5. Ezzell, Ben; "Programación de gráficos en turbo C++" Addison-Wesley 1993

6. Ortega Lopez Fernando; "Depurador de programas para el microprocesador 8086" E.P.N. 1989

7. Dutenmann, Jeff; "La Biblia del Turbo Pascal" ANAYA 1989

8. Zelenovsky Ricardo; " IBM PC para Ingenieros" ESPE 1996

9. Rodríguez-Roselló Miguel Angel; "8088 - 8086/8087 Programación Ensamblador en Entorno MS DOS" ANAYA MULTIMEDIA 1987

10. Norton Peter; "Guía del Programador para el IBM PC" ANAYA MULTIMEDIA 1987.

APENDICES A - H

El contenido de los apendices A - F se encuentra en el Tomo 2 de la presente tesis.

El contenido de los apendices G - H se encuentra en el Tomo 3 de la presente tesis.