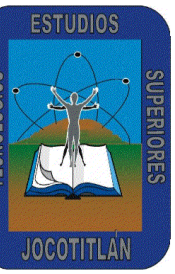

TECNOLÓGICO DE ESTUDIOS SUPERIORES DE JOCOTITLÀN

INGENIERÍA EN SISTEMAS COMPUTACIONALES

ASIGNATURA:

SEMINARIO DE DESARROLLO DE PROYECTOS DE INVESTIGACIÓN

PROGRAMACIÓN EN ENSAMBLADOR PARA PROCESADORES 80x86

DOCENTE:

ISC. JUAN ALBERTO ANTONIO VELÁZQUEZ

PRESENTA:

ALVA HILARIO GUSTAVO

BECERRIL LÓPEZ NANCY

CRUZ MATIAS DEISY

GONZALEZ MALDONADO MARTHA YARELI

ROMERO ORTEGA FRANCISCO JAVIER

SANCHEZ CRUZ GUSTAVO

GRUPO:ISC-801

JOCOTITLÁN MÉXICO, 15 DE ABRIL DE 2008.

INDICE

INTRODUCCION.

El presente trabajo, está enfocado al lenguaje ensamblador de los procesadores 8088, 8086, 80186, 80188 y 80286, asi como todo lo necesario para programar en lenguaje ensamblador y todas las herramientas que nos proporciona este para realizar programas.

Este documento trata de abarcar, de la forma más general, todo aquello que involucra el conocimiento y uso del lenguaje ensamblador.

Ha sido organizado en CUATRO partes importantes que describen detalladamente aspectos relevantes a considerar para el uso de lenguaje ensamblador.

La primera describe los conocimientos básicos que deben poseerse para una mejor comprensión e interpretación de lo que es el lenguaje ensamblador y cómo debe ser usado.

La segunda parte presenta una breve descripción de lo que es el lenguaje ensamblador, ventajas y desventajas de este, instrucciones generales, todo lo que nos proporciona el lenguaje ensamblador.

En el tercer apartado se mencionan los pasos para la creación y depuración ade un programa en lenguaje ensamblador.

En la última parte se explican los tipos de programación en lenguaje ensamblador asi como todas las instrucciones interrupciones, parámetros, procedimientos en cada uno de ellos. .

OBJETIVOS

- Proporcionar información a la comunidad estudiantil que les sirva de apoyo didáctico en la elaboración de investigaciones, realización de practicas y exposición de proyectos en las distintas areas de conocimiento.

- Presentar un proyecto de calidad en la materia de SEMINARIO DE DESARROLLO DE PROYECTOS DE INVESTIGACION EN EL TECNOLOGICO DE ESTUDIOS SUPERIORES DE JOCOTITLAN.

I.-CONCEPTOS BASICOS.
EL LENGUAJE DE MAQUINA Y EL LENGUAJE ENSAMBLADOR.

Todo procesador, grande o pequeño, desde el de una calculadora hasta el de un supercomputador, ya sea de propósito general o específico, posee un lenguaje único que es capaz de reconocer y ejecutar. Por razones que resultan obvias, este lenguaje ha sido denominado Lenguaje de Máquina y más que ser propio de un computador pertenece a su microprocesador. El lenguaje de máquina está compuesto por una serie de instrucciones, que son las únicas que pueden ser reconocidas y ejecutadas por el microprocesador. Este lenguaje es un conjunto de números que representan las operaciones que realiza el microprocesador a través de su circuitería interna. Estas instrucciones, por decirlo así, están grabadas o "alambradas" en el hardware y no pueden ser cambiadas. El nivel más bajo al que podemos aspirar a llegar en el control de un microprocesador es precisamente el del lenguaje de máquina.

Ahora bien, siendo el lenguaje de máquina un conjunto de números, ¿cómo es capaz el microprocesador de saber cuándo un número representa una instrucción y cuándo un dato? El secreto de esto reside en la dirección de inicio de un programa y en el estado del microprocesador. La dirección de inicio nos indica en qué localidad de memoria comienza un programa, y en consecuencia que datos deberemos considerar como instrucciones. El estado del microprocesador nos permite saber cuándo éste espera una instrucción y cuándo éste espera un dato.

Obviamente, el lenguaje de máquina de un microprocesador no puede ser ejecutado por otro microprocesador de arquitectura distinta, a menos que haya cierto tipo de compatibilidad prevista. Por ejemplo, un 80486 es capaz de ejecutar lenguaje de máquina propio y soporta el código generado para microprocesadores anteriores de la misma serie (desde un 8086 hasta un 80386). Por otra parte, un PowerPC es capaz de ejecutar instrucciones de los microprocesadores Motorola 68xxx y de los Intel 80xx/80x86. En ambos casos, el diseño de los microprocesadores se hizo tratando de mantener cierto nivel de compatibilidad con los desarrollados anteriormente. En el segundo caso, este nivel de compatibilidad se extendió a los de otra marca. Sin embargo, un 8088 no puede ejecutar código de un 80186 o superiores, ya que los procesadores más avanzados poseen juegos de instrucciones y registros nuevos no contenidos por un 8088.

Un caso similar es la serie 68xxx, pero de ninguna manera podemos esperar que un Intel ejecute código de un Motorola y viceversa. Y esto no tiene nada que ver con la compañía, ya que Intel desarrolla otros tipos de microprocesadores como el 80860 y el iWARP, los cuales no pueden compartir código ni entre ellos ni entre los 80xx/80xxx.

Ahora bien, mientras que con el lenguaje de máquina, nosotros obtenemos un control total del microprocesador, la programación en este lenguaje resulta muy difícil y fácil para cometer errores. No tanto por el hecho de que las instrucciones son sólo números, sino porque se debe calcular y trabajar con las direcciones de memoria de los datos, los saltos y las direcciones de llamadas a subrutinas, además de que para poder

hacer ejecutable un programa, se deben enlazar las rutinas de run-time y servicios del sistema operativo.

Este proceso es al que se le denomina ensamblado de código. Para facilitar la elaboración de programas a este nivel, se desarrollaron los Ensambladores y el Lenguaje Ensamblador.

Existe una correspondencia 1 a 1 entre las instrucciones del lenguaje de máquina y las del lenguaje ensamblador. Cada uno de los valores numéricos del lenguaje de máquina tiene una representación simbólica de 3 a 5 letras como instrucción del lenguaje ensamblador. Adicionalmente, este lenguaje proporciona un conjunto de pseudo-operaciones (tambien conocidas como directivas del ensamblador) que sirven para definir datos, rutinas y todo tipo de información para que el programa ejecutable sea creado de determinada forma y en determinado lugar.

INTERPRETES, COMPILADORES Y ENSAMBLADORES.

Aun cuando el lenguaje ensamblador fue diseñado para hacer más fácil la programación de bajo nivel, esta resulta todavía complicada y muy laboriosa. Por tal motivo se desarrollaron los lenguajes de alto nivel, para facilitar la programación de los computadores, minimizando la cantidad de instrucciones a especificar. Sin embargo, esto no quiere decir que el microprocesador ejecute dichos lenguajes. Cada una de las instrucciones de un lenguaje de alto nivel o de un nivel intermedio, equivalen a varias de lenguaje máquina o lenguaje ensamblador.

La traducción de las instrucciones de nivel superior a las de bajo nivel la realizan determinados programas. Por una parte tenemos los interpretes, como DBase, BASIC, APL, y Lisp. En estos, cada vez que se encuentra una instrucción, se llama una determinada rutina de lenguaje de máquina que se encarga de realizar las operaciones asociadas, pero en ningún momento se genera un código objeto y mucho menos un código ejecutable. Por otra parte, tenemos los compiladores, como los desarrollados para Fortran, Clipper, COBOL, Pascal o C, que en vez de llamar y ejecutar una rutina en lenguaje de máquina, éstos juntan esas rutinas para formar el código objeto que, después de enlazar las rutinas de run-time y llamadas a otros programas y servicios del sistema operativo, se transformará en el programa ejecutable.

Finalmente, tenemos los ensambladores— como los descritos en este trabajo —que son como una versión reducida y elemental de un compilador (pero que de ninguna manera deben considerarse como tales), ya que lo único que tienen que hacer es cambiar toda referencia simbólica por la dirección correspondiente, calcular los saltos, resolver referencias y llamadas a otros programas, y realizar el proceso de enlace. Los ensambladores son programas destinados a realizar el ensamblado de un determinado código.

EL PROCESO DE LIGA, LAS RUTINAS DE RUN-TIME Y LOS SERVICIOS DEL SISTEMA OPERATIVO.

Para crear un programa ejecutable a partir de un código objeto se require que se resulevan las llamadas a otros programas y a los servicios del sistema operativo, y agregar

las rutinas o información de run-time para que el programa pueda ser cargado a memoria y ejecutado.

Este proceso es lo que se conoce como Link o proceso de liga, y se realiza a través de un ligador o Linker que toma de entrada el código objeto y produce de salida el código ejecutable.

Las rutinas de run-time son necesarias, puesto que el sistema operativo requiere tener control sobre el programa en cualquier momento, además de que la asignación de recursos y su acceso deben hacerse sólamente a través del sistema operativo. Para los computadores personales, esto no es tan complejo como para otros computadores y sistemas operativos, pero es requerido.

ARQUITECTURA DE LOS MICROPROCESADORES .

Sin importar de que microprocesador se trate, los microprocesadores del 8088 al 80486 usan el modelo de registros del 8086. Los microprocesadores matemáticos 80287 al 80487 utilizan el modelo de registros expandidos del 8087. Para mayor detalle ver los apéndices A y B.

Los microprocesadores matemáticos están diseñados exclusivamente para efectuar operaciones aritméticas de una manera más rápida y precisa bajo el control de otro procesador razón por la cual se denominan coprocesadores. Estos también poseen un juego de instrucciones de lenguaje de máquina propio.

La diferencia entre los diversos microprocesadores de uso general y los coprocesadores reside en el nuevo conjunto de instrucciones, registros y señalizadores agregados con cada nueva liberación de un procesador superior. Estas adiciones se hicieron con el fin de agregar un mayor poder de cómputo sin alterar la estructura básica, para así mantener la compatibilidad con los desarollos anteriores, tanto de software como de hardware.

La diferencia entre los 8086 y 8088 con los 80186 y 80188 no es muy grande, ésta radica en un grupo de instrucciones que fueron agregadas al 80186 y al 80188. La diferencia entre el 8086 y el 8088, lo mismo que entre el 80186 y el 80188, es el modelo de memoria que usan ambos procesadores. El 8088 y el 80188 están diseñados como microprocesadores de 8 bits por lo que el modo de acceso a la memoria es ligeramente distinto pero compatible con el 8086 y el 80186. Esto se verá con más detalle en un tema posterior.

Debido al tipo de microprocesador empleado, la memoria de la PC se encuentra dividida en una serie de blocks denominados segmentos, de 64KB cada uno. La memoria es accesada especificando el segmento y el desplazamiento dentro del segmento (segmento:desplazamiento, para mayor detalle ver el apendice C). Para las PC´s la memoria se clasifica en tres tipos:

- Convencional. Es la memoria de tipo básico y que abarca las direcciones de 0 a 640KB. En ésta es donde se cargan los programas de usuario y el sistema operativo, y es la que está disponible para equipo XT (8088,8086, 80186 y 80188).

- Extendida. Esta memoria sólo está disponible para procesadores 80286 y mayores (equipo AT, 80386 y 80486). Muchos programas que usan la memoria convencional no pueden usar la memoria extendida porque las direcciones en memoria extendida están más alla de las que el programa puede reconocer. Únicamente las direcciones dentro de los 640KB pueden ser reconocidas por todos los programas. Para reconocer la memoria extendida se requiere de un manejador de memoria extendida, como HIMEM.SYS que provee MS-DOS.

- Expandida. Esta es la memoria que se agrega al computador a través de una tarjeta de expansión, y que debe ser administrada por un programa especial, como el EMM386.EXE. A diferencia de la memoria convencional o extendida, la memoria expandida es dividida en bloques de 16K llamados páginas (pages) . Cuando un programa solicita información de memoria expandida el manejador copia la página correspondiente en un área denominada page frame para poder ser accesada en la memoria extendida. Como podremos ver, el 8088, 8086, 80188 y 80186 son capaces de direccionar hasta 1 MB de memoria. Ya hemos indicado que la memoria convencional sólo abarca 640KB, así nos quedan 384KB libres. Esta parte de la memoria es denominada parte alta, y como no está disponible para muchos programas generalmente se usa para cargar drivers del sistema operativo, programas residentes y datos de hardware (ROM y páginas de video).

HISTORIA DE LOS PROCESADORES

Con la aparición de los circuitos integrados, la posibilidad de reducir el tamaño de algunos dispositivos electrónicos se vio enormemente favorecida. Los fabricantes de controladores integrados, calculadoras y algunos otros dispositivos comenzaron a solicitar sistemas integrados en una sola pastilla, esto dio origen a la aparición de los microprocesadores.

Microprocesadores de 4 bits

En 1971, una compañía que se dedicaba a la fabricación de memorias electrónicas lanzó al mercado el primer microprocesador del mundo. Este microprocesador fue el resultado de un trabajo encargado por una empresa que se dedicaba a la fabricación de calculadoras electrónicas. El 4004 era un microprocesador de 4 bits capaz de direccionar 4096 localidades de memoria de 4 bits de ancho. Este microprocesador contaba con un conjunto de 45 instrucciones y fue ampliamente utilizado en los primeros videojuegos y sistemas de control.

Microprocesadores de 8 bits

Con la aparición de aplicaciones más complejas para el microprocesador y el gran éxito comercial del 4004, Intel decidió lanzar al mercado un nuevo microprocesador, el 8008, éste fue el primer microprocesador de 8 bits. Las características de este microprocesador fueron:

- Capacidad de direccionamiento de 16 Kb
- Memoria de 8 bits
- Conjunto de 48 instrucciones

Este microprocesador tuvo tanto éxito, que en cosa de dos años su capacidad de proceso fue insuficiente para los ingenieros y desarrolladores, por lo cual en 1973 se liberó el 8080. Este microprocesador fue una versión mejorada de su predecesor y las mejoras consistieron en un conjunto más grande de instrucciones, mayor capacidad de direccionamiento y una mayor velocidad de procesamiento.

Finalmente, en 1977, Intel anunció la aparición del 8085. Este era el último microprocesador de 8 bits y básicamente idéntico al 8080. Su principal mejora fue la incorporación del reloj temporizador dentro de la misma pastilla.

Microprocesadores de 16 bits

En 1978, Intel lanzó al mercado el 8086 y un poco más tarde el 8088. Estos dos microprocesadores contaban con registros internos de 16 bits, tenían un bus de datos externo de 16 y 8 bits respectivamente y ambos eran capaces de direccionar 1Mb de memoria por medio de un bus de direcciones de 20 líneas.
Otra característica importante fue que estos dos microprocesadores eran capaces de realizar la multiplicación y la división por hardware, cosa que los anteriores no podían.
Finalmente apareció el 80286. Este era el último microprocesador de 16 bits, el cual era una versión mejorada del 8086. El 286 incorporaba una unidad adicional para el manejo de memoria y era capaz de direccionar 16Mb en lugar de 1Mb del 8086.

Microprocesadores de 32 bits

El 80386 marco el inicio de la aparición de los microprocesadores de 32 bits. Estos microprocesadores tenían grandes ventajas sobre sus predecesores, entre las cuales se pueden destacar: direccionamiento de hasta 4Gb de memoria, velocidades de operación más altas, conjuntos de instrucciones más grandes y además contaban con memoria interna (caché) de 8Kb en las versiones más básicas.
Del 386 surgieron diferentes versiones, las cuales se listan a continuación.

Modelo	Bus de Datos	Coprocesador matemático
80386DX	32	Si
80386SL	16	No
80386SX	16	No
80486SX	32	No
80486DX	32	Si

Terminales del microprocesador

En esta sección se realizará una breve descripción del conjunto de terminales del microprocesador más representativo de la familia 80x86.

El microprocesador 8086 puede trabajar en dos modos diferentes: el modo mínimo y el modo máximo. En el modo máximo el microprocesador puede trabajar en forma conjunta con un microprocesador de datos numérico 8087 y algunos otros circuitos periféricos. En el modo mínimo el microprocesador trabaja de forma más autónoma al no depender de circuitos auxiliares, pero esto a su vez le resta flexibilidad.

En cualquiera de los dos modos, las terminales del microprocesador se pueden agrupar de la siguiente forma:

- Alimentación
- Reloj
- Control y estado
- Direcciones
- Datos

El 8086 cuenta con tres terminales de alimentación: tierra (GND) en las terminales 1 y 20 y Vcc=5V en la terminal 40.

En la terminal 19 se conecta la señal de reloj, la cual debe provenir de un generador de reloj externo al microprocesador.

El 8086 cuenta con 20 líneas de direcciones (al igual que el 8088). Estas líneas son llamadas A0 a A19 y proporcionan un rango de direccionamiento de 1MB.

Para los datos, el 8086 comparte las 16 líneas más bajas de sus líneas de direcciones, las cuales son llamadas AD0 a AD15. Esto se logra gracias a un canal de datos y direcciones multiplexado.

En cuanto a las señales de control y estado tenemos las siguientes:

La terminal MX/MN controla el cambio de modo del microprocesador.

Las señales S0 a S7 son señales de estado, éstas indican diferentes situaciones acerca del estado del microprocesador.

La señal RD en la terminal 32 indica una operación de lectura.

En la terminal 22 se encuentra la señal READY. Esta señal es utilizada por los diferentes dispositivos de E/S para indicarle al microprocesador si se encuentran listos para una transferencia.

La señal RESET en la terminal 21 es utilizada para reinicializar el microprocesador.

La señal NMI en la terminal 17 es una señal de interrupción no enmascarable, lo cual significa que no puede ser manipulada por medio de software.

La señal INTR en la terminal 18 es también una señal de interrupción, la diferencia radica en que esta señal si puede ser controlada por software. Las interrupciones se estudian más adelante.

La terminal TEST se utiliza para sincronizar al 8086 con otros microprocesadores en una configuración en paralelo.

Las terminales RQ/GT y LOCK se utilizan para controlar el trabajo en paralelo de dos o mas microprocesadores.
La señal WR es utilizada por el microprocesador cuando éste requiere realizar alguna operación de escritura con la memoria o los dispositivos de E/S.

Las señales HOLD y HLDA son utilizadas para controlar el acceso al bus del sistema.

DIAGRAMA DE COMPONENTES INTERNOS

Descripción de los componentes

La figura 2 muestra la estructura interna del microprocesador 8086 con base en su modelo de programación. El microprocesador se divide en dos bloques principales: la unidad de interfaz del bus y la unidad de ejecución. Cada una de estas unidades opera de forma asíncrona para maximizar el rendimiento general del microprocesador.

Unidad de ejecución

Este elemento del microprocesador es el que se encarga de ejecutar las instrucciones. La unidad de ejecución comprende el conjunto de registros de propósito general, el registro de banderas y la unidad aritmético-lógica.

Unidad de interfaz de bus

Esta unidad, la cual se conoce como BIU (Bus Interface Unit), procesa todas las operaciones de lectura/escritura relacionadas con la memoria o con dispositivos de entrada/salida, provenientes de la unidad de ejecución. Las instrucciones del programa que se está ejecutando son leídas por anticipado por esta unidad y almacenadas en la cola de instrucciones, para después ser transferidas a la unidad de ejecución.

Unidad aritmético-lógica

Conocida también como ALU, este componente del microprocesador es el que realmente realiza las operaciones aritméticas (suma, resta, multiplicación y división) y lógicas (and, or, xor, etc.) que se obtienen como instrucciones de los programas.

Buses internos (datos y direcciones)

Los buses internos son un conjunto de líneas paralelas (conductores) que interconectan las diferentes partes del microprocesador.

Existen dos tipos principales: el bus de datos y el bus de direcciones. El bus de datos es el encargado de transportar los datos entre las distintas partes del microprocesador; por otro lado, el bus de direcciones se encarga de transportar las direcciones para que los datos puedan ser introducidos o extraídos de la memoria o dispositivos de entrada y salida.

Cola de instrucciones

La cola de instrucciones es una pila de tipo FIFO (primero en entrar, primero en salir) donde las instrucciones son almacenadas antes de que la unidad de ejecución las ejecute.

Registros de propósito general

El microprocesador 8086 cuenta con cuatro registros de propósito general, los cuales pueden ser usados libremente por los programadores. Estos registros reciben los nombres siguientes:

AX (Acumulador) Este registro es el encargado de almacenar el resultado de algunas operaciones aritméticas y lógicas.

BX (Base) Este registro es utilizado para calcular direcciones relativas de datos en la memoria.

CX (Contador) Su función principal es la de almacenar el número de veces que un ciclo de instrucciones debe repetirse.

DX (Datos) Por lo general se utiliza para acceder a las variables almacenadas en la memoria.

Registros apuntadores

El 8086 también cuenta con dos registros apuntadores SP y BP. Estos registros reciben su nombre por que su función principal es la de apuntar a alguna dirección de memoria especifica.

SP (Apuntador de pila) Se encarga de controlar el acceso de los datos a la pila de los programas. Todos los programas en lenguaje ensamblador utilizan una pila para almacenar datos en forma temporal.

BP (Apuntador Base) Su función es la de proporcionar direcciones para la transferencia e intercambio de datos.

Registros índices

Existen dos registros llamados SI y DI que están estrechamente ligados con operaciones de cadenas de datos.

SI (Índice Fuente) Proporciona la dirección inicial para que una cadena sea manipulada.

DI (Índice Destino) Proporciona la dirección de destino donde por lo general una cadena será almacenada después de alguna operación de transferencia.

Registros de segmento

El 8086 cuenta con cuatro registros especiales llamados registros de segmento.

CS (Segmento de código) Contiene la dirección base del lugar donde inicia el programa almacenado en memoria.

DS (Segmento de datos) Contiene la dirección base del lugar del área de memoria donde fueron almacenadas las variables del programa.

ES (Segmento extra) Este registro por lo general contiene la misma dirección que el registro DS.

SS (Segmento de Pila) Contiene la dirección base del lugar donde inicia el área de memoria reservada para la pila.

Registro apuntador de instrucciones

IP (Apuntador de instrucciones) Este registro contiene la dirección de desplazamiento del lugar de memoria donde está la siguiente instrucción que será ejecutada por el microprocesador.

Registro de estado

Conocido también como registro de banderas (Flags), tiene como función principal almacenar el estado individual de las diferentes condiciones que son manejadas por el microprocesador. Estas condiciones por lo general cambian de estado después de cualquier operación aritmética o lógica:
CF (Acarreo) Esta bandera indica el acarreo o préstamo después de una suma o resta.
OF (Sobreflujo) Esta bandera indica cuando el resultado de una suma o resta de números con signo sobrepasa la capacidad de almacenamiento de los registros.
SF (Signo) Esta bandera indica si el resultado de una operación es positivo o negativo. SF=0 es positivo, SF=1 es negativo.
DF (Dirección) Indica el sentido en el que los datos serán transferidos en operaciones de manipulación de cadenas. DF=1 es de derecha a izquierda y DF=0 es de izquierda a derecha.
ZF (Cero) Indica si el resultado de una operación aritmética o lógica fue cero o diferente de cero. ZF=0 es diferente y ZF=1 es cero.

IF (interrupción) Activa y desactiva la terminal INTR del microprocesador.
PF (paridad) Indica la paridad de un número. Si PF=0 la paridad es impar y si PF=1 la paridad es par.
AF (Acarreo auxiliar) Indica si después de una operación de suma o resta ha ocurrido un acarreo de los bits 3 al 4.
TF (Trampa) Esta bandera controla la ejecución paso por paso de un programa con fines de depuración.

Funcionamiento interno (ejecución de un programa)

Para que un microprocesador ejecute un programa es necesario que éste haya sido ensamblado, enlazado y cargado en memoria.

Una vez que el programa se encuentra en la memoria, el microprocesador ejecuta los siguientes pasos:

1.- Extrae de la memoria la instrucción que va a ejecutar y la coloca en el registro interno de instrucciones.

2.- Cambia el registro apuntador de instrucciones (IP) de modo que señale a la siguiente instrucción del programa.

3.- Determina el tipo de instrucción que acaba de extraer.

4.- Verifica si la instrucción requiere datos de la memoria y, si es así, determina donde están situados.

5.- Extrae los datos, si los hay, y los carga en los registros internos del CPU.
6.- Ejecuta la instrucción.

7.- Almacena los resultados en el lugar apropiado.

8.- Regresa al paso 1 para ejecutar la instrucción siguiente.

Este procedimiento lo lleva a cabo el microprocesador millones de veces por segundo.

Manejo de memoria

Segmentación

El microprocesador 8086, como ya se mencionó, cuenta externamente con 20 líneas de direcciones, con lo cual puede direccionar hasta 1 MB (00000h--FFFFFh) de localidades de memoria. En los días en los que este microprocesador fue diseñado, alcanzar 1MB de direcciones de memoria era algo extraordinario, sólo que existía un problema: internamente todos los registros del microprocesador tienen una longitud de 16 bits, con lo cual sólo se pueden direccionar 64 KB de localidades de memoria. Resulta obvio que con este diseño se desperdicia una gran cantidad de espacio de almacenamiento; la solución a este problema fue la segmentación.

La segmentación consiste en dividir la memoria de la computadora en segmentos. Un segmento es un grupo de localidades con una longitud mínima de 16 bytes y máxima de 64KB.
La mayoría de los programas diseñados en lenguaje ensamblador y en cualquier otro lenguaje definen cuatro segmentos. El segmento de código, el segmento de datos, el segmento extra y el segmento de pila.

A cada uno de estos segmentos se le asigna una dirección inicial y ésta es almacenada en los registros de segmento correspondiente, CS para el código, DS para los datos, ES para el segmento extra y SS para la pila.

Dirección física

Para que el microprocesador pueda acceder a cualquier localidad de memoria dentro del rango de 1MB, debe colocar la dirección de dicha localidad en el formato de 20 bits.

Para lograr esto, el microprocesador realiza una operación conocida como cálculo de dirección real o física. Esta operación toma el contenido de dos registros de 16 bits y obtiene una dirección de 20 bits.

La formula que utiliza el microprocesador es la siguiente:

Dir. Física = Dir. Segmento * 10h + Dir. Desplazamiento

Por ejemplo: si el microprocesador quiere acceder a la variable X almacenada en memoria, necesita conocer su dirección desplazamiento. La dirección segmento para las variables es proporcionada por el registro DS. Para este caso, supongamos que X tiene el desplazamiento 0100h dentro del segmento de datos y que DS tiene la dirección segmento 1000h, la dirección física de la variable X dentro del espacio de 1Mb será:

Dir. Física = 1000h * 10h +0100h

Dir. Física = 10000h + 0100h

Dir. Física = 10100h (dirección en formato de 20 bits).

Dirección efectiva (desplazamiento)

La dirección efectiva (desplazamiento) se refiere a la dirección de una localidad de memoria con respecto a la dirección inicial de un segmento. Las direcciones efectivas sólo pueden tomar valores entre 0000h y FFFFh, esto es porque los segmentos están limitados a un espacio de 64 Kb de memoria.
En el ejemplo anterior, la dirección real de la variable X fue de 10100h, y su dirección efectiva o de desplazamiento fue de 100h con respecto al segmento de datos que comienza en la dirección 10000h.
Direccionamiento de los datos
En la mayoría de las instrucciones en lenguaje ensamblador, se hace referencia a datos que se encuentran almacenados en diferentes medios, por ejemplo: registros, localidades de memoria, variables, etc.

Para que el microprocesador ejecute correctamente las instrucciones y entregue los resultados esperados, es necesario que se indique la fuente o el origen de los datos con los que va a trabajar, a esto se le conoce como direccionamiento de datos.

En los microprocesadores 80x86 existen cuatro formas de indicar el origen de los datos y se llaman modos de direccionamiento.

Para explicar estos cuatro modos, tomaremos como ejemplo la instrucción más utilizada en los programas en ensamblador, la instrucción MOV.

La instrucción MOV permite transferir (copiar) información entre dos operandos; estos operandos pueden ser registros, variables o datos directos colocados por el programador. El formato de la instrucción MOV es el siguiente:

Mov Oper1,Oper2

Esta instrucción copia el contenido de Oper2 en Oper1.

Direccionamiento directo

Este modo se conoce como directo, debido a que en el segundo operando se indica la dirección de desplazamiento donde se encuentran los datos de origen.

Ejemplo:

```
Mov AX,[1000h]      ;Copia en AX lo que se encuentre almacenado en
                    ; DS:1000h
```

Direccionamiento inmediato

En este modo, los datos son proporcionados directamente como parte de la instrucción.

Ejemplo:

```
Mov AX,34h          ;Copia en AX el número 34h hexadecimal
Mov CX,10           ;Copia en CX el número 10 en decimal
```

Direccionamiento por registro

En este modo de direccionamiento, el segundo operando es un registro, el cual contiene los datos con los que el microprocesador ejecutará la instrucción.

Ejemplo:

```
Mov AX,BX           ;Copia en AX el contenido del reg
```

Direccionamiento indirecto por registro

Finalmente, en el modo indirecto por registro, el segundo operando es un registro, el cual contiene la dirección desplazamiento correspondiente a los datos para la instrucción.

Ejemplo:

```
Mov AX,[BX]         ; Copia en AX el dato que se encuentre en la localidad de
                            ;memoria DS:[BX]
```

Los paréntesis cuadrados sirven para indicar al ensamblador que el número no se refiere a un dato, si no que se refiere a la localidad de memoria.

En los siguientes capítulos se muestran varios programas, en los cuales podrá identificar los diferentes modos de direccionamiento de datos.

EL SISTEMA OPERATIVO MS-DOS.

Junto con todo lo visto anteriormente, y como se mencionó anteriomente, uno de los componentes que caracterizan los computadores personales es su sistema operativo. Una PC puede correr varios sistemas operativos: CP/M, CP/M-86, XENIX, Windows, PC-DOS, y MS-DOS. Lo que los define es la forma en que están integrados sus servicios y la forma en

que se accesa a ellos. Esto es precisamente lo que el linker debe enlazar y resolver. Aquí nos enfocaremos exclusivamente en el sistema operativo MS-DOS, y lo que se mencione aquí será valido para las versiones 3.0 y superiores. Este sistema operativo está organizado de la siguiente manera:

- Comandos Internos (reconocidos y ejecutados por el COMMAND.COM)
  Comandos Externos (.EXEs y .COMs)
  Utilerías y drivers (programas de administración del sistema)
  Shell (Interfaz amigable, sólo versiones 4.0 o mayores)
  Servicios (Interrupciones)

Los servicios, más conocidos como interrupciones o vectores de interrupción, es parte medular de lo que es MS-DOS, y no son mas que rutinas definidas por MS-DOS y el BIOS, ubicadas a partir de una localidad de memoria específica. La manera de accesar a estas rutinas y a los servicios que ofrecen es mediante una instrucción que permite ejecutar una interrupción.

MS-DOS proporciona dos módulos: IBMBIO.COM (provee una interfaz de bajo nivel para el BIOS) e IBMDOS.COM (contiene un manejador de archivos y servicios para manejo de registros). En equipos compatibles estos archivos tienen los nombres IO.SYS y MSDOS.SYS, respectivamente. El acceso a los servicios del computador se realiza de la siguiente manera:

Programa DOS DOS ROM EXTERNO
de usuario Alto nivel Bajo nivel
petición de — IBMDOS.COM — IBMBIO.COM — BIOS — Dispositivo
I/O

ENSAMBLADORES Y MACROENSAMBLADORES.

Existen varios ensambladores disponibles para ambiente MS-DOS: el IBM Macro Assembler, el Turbo Assembler de Borland, el Turbo Editassm de Speedware, por citar algunos. Una breve descripción de cada uno se propociona a continuación.

Macro Ensamblador IBM.- Está integrado por un ensamblador y un macroensamblador. En gran medida su funcionamiento y forma de invocarlo es sumamente similar al de Microsoft. Su forma de uso consiste en generar un archivo fuente en código ASCII, se procede a generar un programa objeto que es ligado y se genera un programa .EXE. Opcionalmente puede recurirse a la utilería EXE2BIN de MS-DOS para transformarlo a .COM. Es capaz de generar un listado con información del proceso de ensamble y referencias cruzadas.

Macro Ensamblador de Microsoft.- Dependiendo de la versión, este ensamblador es capaz de soportar el juego de instrucciones de distintos tipos de microprocesadores Intel de la serie 80xx/80x86. En su versión 4.0 este soporta desde el 8086 al 80286 y los coprocesadores 8087 y 80287. Requiere 128KB de memoria y sistema operativo MS-DOS v2.0 o superior. Trabaja con un archivo de código fuente creado a partir de un editor y grabado en formato ASCII. Este archivo es usado para el proceso de ensamble y

generación de código objeto. Posteriormente, y con un ligador, es creado el código ejecutable en formato .EXE.

Turbo Editassm.- Este es desarrollado por Speddware, Inc., y consiste de un ambiente integrado que incluye un editor y utilerías para el proceso de ensamble y depuración. Es capaz de realizar el ensamble línea a línea, conforme se introducen los mnemónicos, y permite revisar listas de referencias cruzadas y contenido de los registros. Este ensamblador trabaja con tablas en memoria, por lo que la generación del código ejecutable no implica la invocación explícita del ligador por parte del programador. Adicionalmente permite la generación de listados de mensajes e información de cada etapa del proceso y la capacidad de creación de archivos de código objeto.

Turbo Assembler.- De Borland Intl., es muy superior al Turbo Editassm. Trabaja de la misma forma, pero proporciona una interfaz mucho más fácil de usar y un mayor conjunto de utilerías y servicios.

En lo que se refiere a las presentes notas, nos enfocaremos al Microsoft Macro Assembler v4.0. Los programas ejemplo han sido desarrollados con éste y está garantizado su funcionamiento. Estos mismo programas posiblemente funcionen con otros ensambladores sin cambios o con cambios mínimos cuando utilizan directivas o pseudoinstrucciones. Realmente la diferencia entre los ensambladores radica en la forma de generar el código y en las directivas con que cuente, aunque estas diferencias son mínimas. El código ensamblador no cambia puesto que los microprocesadores con los que se va a trabajar son comunes. Así, todos los programas que se creen con un ensamblador en particular podrán ser ensamblados en otro, cambiando las pseudo-operaciones no reconocidas por el equivalente indicado en el manual de referencia del paquete empleado. Los programas que componen el Macro Ensamblador Microsoft v4.0 son los siguientes:

Programa Descripción
MASM.EXE Microsoft Macro Assembler
LINK.EXE Microsoft 8086 object linker
SYMDEB.EXE Microsoft Symbolic Debuger Utility
MAPSYM.EXE Microsoft Symbol File Generator
CREF.EXE Microsoft Cross-Reference Utility
LIB.EXE Microsoft Library Manager
MAKE.EXE Microsoft Program Maintenance Utility
EXEPACK.EXE Microsoft EXE File Compression Utility
EXEMOD.EXE Microsoft EXE File Header Utility
COUNT.ASM Sample source file for SYMDEB session
README.DOC Updated information obtained after the manual
was printed.

El Microsoft Macro Assembler v4.0 crea código ejecutable para procesadores 8086, 8088, 80186, 80188, 80286, 8087 y 80287. Además es capaz de aprovechar las instrucciones del 80286 en la creación de código protegido y no protegido. El término macroensamblador es usado para indicar que el ensamblador en cuestión tiene la capacidad de poder ensamblar programas con facilidad de macro. Una macro es una pseudo-instrucción que define un conjunto de instrucciones asociadas a un nombre

simbólico. Por cada ocurrencia en el código de esta macro, el ensamblador se encarga de substituir esa llamada por todas las instrucciones asociadas y, en caso de existir, se dejan los parámetros con los que se estaba llamando la macro y no con los que había sido definida. Es importante señalar que no se deja una llamada, como a una subrutina o procedimiento, sino que se incorporan todas las instrucciones que definen a la macro.

II.- EL LENGUAJE ENSAMBLADOR.

Importancia del lenguaje ensamblador

El lenguaje ensamblador es la forma más básica de programar un microprocesador para que éste sea capaz de realizar las tareas o los cálculos que se le requieran.

El lenguaje ensamblador es conocido como un lenguaje de bajo nivel, esto significa que nos permite controlar el 100 % de las funciones de un microprocesador, así como los periféricos asociados a éste.

A diferencia de los lenguajes de alto nivel, por ejemplo "Pascal", el lenguaje ensamblador no requiere de un compilador, esto es debido a que las instrucciones en lenguaje ensamblador son traducidas directamente a código binario y después son colocadas en memoria para que el microprocesador las tome directamente.

Aprender a programar en lenguaje ensamblador no es fácil, se requiere un cierto nivel de conocimiento de la arquitectura y organización de las computadoras, además del conocimiento de programación en algún otro lenguaje.

Ventajas del lenguaje ensamblador:

- Velocidad de ejecución de los programas
- Mayor control sobre el hardware de la computadora

Desventajas del lenguaje ensamblador:

- Repetición constante de grupos de instrucciones
- No existe una sintaxis estandarizada
- Dificultad para encontrar errores en los programas (bugs)

UN EJEMPLO

Para comenzar veamos un pequeño ejemplo que ilustra el formato del programa fuente. Este ejemplo está completamente desarrollado en lenguaje ensamblador que usa servicios o funciones de MS-DOS (system calls) para imprimir el mensaje Hola mundo!! en pantalla.

```
; HOLA.ASM
; Programa clasico de ejemplo. Despliega una leyenda en pantalla.
STACK SEGMENT STACK ; Segmento de pila
DW 64 DUP (?) ; Define espacio en la pila
STACK ENDS
DATA SEGMENT ; Segmento de datos
SALUDO DB "Hola mundo!!",13,10,"$" ; Cadena
DATA ENDS
CODE SEGMENT ; Segmento de Codigo
```

```
ASSUME CS:CODE, DS:DATA, SS:STACK
INICIO: ; Punto de entrada al programa
MOV AX,DATA ; Pone direccion en AX
MOV DS,AX ; Pone la direccion en los registros
MOV DX,OFFSET SALUDO ; Obtiene direccion del mensaje
MOV AH,09H ; Funcion: Visualizar cadena
INT 21H ; Servicio: Funciones alto nivel DOS
MOV AH,4CH ; Funcion: Terminar
INT 21H
CODE ENDS
END INICIO ; Marca fin y define INICIO
```

La descripción del programa es como sigue:

1.- Las declaraciones SEGMENT y ENDS definen los segmentos a usar.
2.- La variable SALUDO en el segmento DATA, define la cadena a ser desplegada. El signo del dolar al final de la cadena (denominado centinela) es requerido por la función de visualización de la cadena de MS-DOS. La cadena incluye los códigos para carriage-return y line-feed.
3.- La etiqueta START en el segmento de código marca el inicio de las instrucciones del programa.
4.- La declaracion DW en el segmento de pila define el espacio para ser usado por el stack del programa.
5.- La declaración ASSUME indica que registros de segmento se asociarán con las etiquetas declaradas en las definiciones de segmentos.
6.- Las primeras dos instrucciones cargan la dirección del segmento de datos en el registro DS. Estas instrucciones no son necesarias para los segmentos de código y stack puesto que la dirección del segmento de código siempre es cargado en el registro CS y la dirección de la declaración del stack segment es automáticamente cargada en el registro SS.
7.- Las últimas dos instrucciones del segmento CODE usa la función 4CH de MS-DOS para regresar el control al sistema operativo. Existen muchas otras formas de hacer esto, pero ésta es la más recomendada.
8.- La directiva END indica el final del código fuente y especifica a START como punto de arranque.

EL FORMATO DEL ENSAMBLADOR.

De acuerdo a las convenciones y notación seguidas en el manual del Microsoft Macro Assembler.

Negritas Comandos, símbolos y parámetros a ser usados como se muestra.
Itálicas Todo aquello que debe ser reemplazado por el usuario
ð ð Indican un parámetro opcional
„„ Denota un parámetros que puede repetirse varias veces
¦ Separa dos valores mutuamente excluyentes
letra chica Usada para ejemplos. Código y lo que aparece en pantalla.

Cada programa en lenguaje ensamblador es creado a partir de un archivo fuente de código ensamblador. Estos son archivos de texto que contienen todas las declaraciones de datos e instrucciones que componen al programa y que se agrupan en áreas o secciones, cada una con un propósito especial. Las sentencias en ensamblador tienen la

siguiente sintaxis:
[nombre> mnemónico [operandos> [;comentarios>

En cuanto a la estructura, todos los archivos fuente tienen la misma forma: cero o más segmentos de programa seguidos por una directiva END. No hay una regla sobre la estructura u orden que deben seguir las diversas secciones o áreas en la creación del código fuente de un programa en ensamblador. Sin embargo la mayoría de los programas tiene un segmento de datos, un segmento de código y un segmento de stack, los cuales pueden ser puestos en cualquier lugar.

Para la definición de datos y declaración de instrucciones y operandos el MASM reconoce el conjunto de caracteres formado por letras mayúsculas, letras minúsculas (excluyendo caracteres acentuados, ñ, Ñ), números, y los símbolos: ? @ _ $: . [> ( ) { } + - / * & % ! ´ ~ ¦ \ = # ; , " `

La declaración de números requiere tener presente ciertas consideraciones. En el MASM un entero se refiere a un número entero: combinación de dígitos hexadecimales, octales, decimales o binarios, más una raíz opcional. La raíz se especifica con B, Q u O, D, o H. El ensamblador usará siempre la raíz decimal por defecto, si se omite la especificación de la raíz (la cual se puede cambiar con la directiva .RADIX). Así nosotros podemos especificar un entero de la siguiente manera: digitos, digitosB, digitosQ o digitosO, digitosD, digitosH. Si una D o B aparecen al final de un número, éstas siempre se considerarán un especificador de raíz, e.g. 11B será tratado como 112 (210), mientras que si se trata del número 11B16 debe introducirse como 11Bh.

Para los números reales tenemos al designador R, que sólo puede ser usado con números hexadecimales de 8, 16, ó 20 digitos de la forma digitosR. También puede usarse una de las directivas DD, DQ, y DT con el formato [+¦->digitos.digitos[E[+¦->digitos>. Las cadenas de carácter y constantes alfanuméricas son formadas como ´caracteres´ o "caracteres" . Para referencias simbólicas se utilizan cadenas especiales denominadas nombres. Los nombres son cadenas de caracteres que no se entrecomillan y que deben comenzar con una A..Z ¦ a..z ¦ _ ¦ $ ¦ @ los caracteres restantes pueden ser cualquiera de los permitidos, y solamente los 31 primeros caracteres son reconocidos.

DIRECTIVAS.

El MASM posee un conjunto de instrucciones que no pertenecen al lenguaje ensamblador propiamente sino que son instrucciones que únicamente son reconocidas por el ensambldor y que han sido agregadas para facilitar la tarea de ensablamblado, tanto para el programador como para el programa que lo lleva a cabo. Dichas instrucciones son denominadas directivas. En general, las directivas son usadas para especificar la organización de memoria, realizar ensamblado condicional, definir macros, entrada, salida, control de archivos, listados, cross-reference, direcciones e información acerca de la estructura de un programa y las declaraciones de datos.

*** Conjunto de instrucciones.-** Dentro de las directivas más importantes, tenemos las que establecen el conjunto de instrucciones a soportar para un microprocesador en especial: .8086(default).- Activa las instrucciones para el 8086 y 8088 e inhibe las del 80186 y

8 0 2 8 6 .
.8087(default).- Activa instrucciones para el 8087 y desactiva las del 80287.

.186.- Activa las instrucciones del 80186.

.286c.- Activa instrucciones del 80286 en modo no protegido.
.289p.- Activa instrucciones del 80286 en modo protegido y no protegido.
.287.- Activa las instrucciones para el 80287.

*** Declaración de segmentos.-** En lo que respecta a la estructura del programa tenemos las directivas SEGMENT y ENDS que marcan el inicio y final de un segmento del programa. Un segmento de programa es una colección de instrucciones y/o datos cuyas direcciones son todas relativas para el mismo registro de segmento. Su sintaxis es:

nombre SEGMENT [alineación> [combinación> [´clase´>
nombre ENDS

El nombre del segmento es dado por nombre, y debe ser único. Segmentos con el mismo nombre se tratan como un mismo segmento. Las opciones alineación, combinación, y clase proporcionan información al LINK sobre cómo ajustar los segmentos. Para alineación tenemos los siguientes valores: byte (usa cualquier byte de dirección), word (usa cualquier palabra de dirección, 2 bytes/word), para (usa direcciones de parráfos, 16 bytes/parráfo, deafult), y page (usa direcciones de página, 256 bytes/page). combinación define cómo se combinarán los segmentos con el mismo nombre. Puede asumir valores de: public (concatena todos los segmentos en uno solo), stack (igual al anterior, pero con direcciones relativas al registro SS, common (crea segmentos sobrepuestos colocando el inicio de todos en una misma dirección), memory (indica al LINK tratar los segmentos igual que MASM con public, at address (direccionamiento relativo a address). clase indica el tipo de segmento, señalados con cualquier nombre. Cabe señalar que en la definición está permitido el anidar segmentos, pero no se permite de ninguna manera el sobreponerlos.

*** Fin de código fuente.-** Otra directiva importante es la que indica el final de un módulo. Al alcanzarla el ensamblador ignorará cualquier otra declaración que siga a ésta. Su sintaxis es:

END [expresión> la opción expresión permite definir la dirección en la cual el programa iniciará.

*** Asignación de segmentos**.- La directiva ASSUME permite indicar cuales serán los valores por default que asimirán los registros de segmento. Existen dos formas de hacer esto:

ASSUME registrosegmento:nombre,,,
ASSUME NOTHING
NOTHING cancela valores previos.

*** Etiquetas.-** Las etiquetas son declaradas
nombre:
donde nombre constituye una cadena de caracteres.

*** Declaración de datos.-** Estos se declaran según el tipo, mediante la regla [nombre> directiva valor,,,

donde directiva puede ser DB (bytes), DW (palabras), DD (palabra doble), DQ (palabra cuádruple), DT (diez bytes). También pueden usarse las directivas LABEL (crea etiquetas de instrucciones o datos), EQU (crea símbolos de igualdad) , y el símbolo = (asigna absolutos) para declarar símbolos. Estos tienen la siguiente sintaxis:

nombre = expresion
nombre EQU expresión
nombre LABEL tipo
donde tipo puede ser BYTE, WORD, DWORD, QWORD, TBYTE, NEAR, FAR.

*** Declaración de estructuras.-** Para la declaración de estructuras de datos se emplea la directiva STRUC. Su sintaxis es:

nombre STRUC
campos
nombre ENDS

CONJUNTO DE INSTRUCCIONES.

El juego completo de instrucciones reconocidas por los procesadores intel 8086 a 80286, junto con los coprocesadores 8087 y 80287, se enlistan en el apendice E. Como puede verse en dicho apéndice, la mayoría de las instrucciones requieren algunos operandos o expresiones para trabajar, y lo cual es válido también para las directivas. Los operandos representan valores, registros o localidades de memoria a ser accesadas de alguna manera. Las expresiones combinan operandos y operadores aritméticos y lógicos para calcular en valor o la dirección a acceder.

Los operandos permitidos se enlistan a continuación:

Constantes.- Pueden ser números, cadenas o expresiones que representan un valor fijo. Por ejemplo, para cargar un registro con valor constante usaríamos la instrucción MOV indicando el registro y el valor que cargaríamos dicho registro.

mov ax,9
mov al,´c´
mov bx,65535/3
mov cx,count
count sólo sera válido si este fue declarado con la directiva EQU.

Directos.- Aquí se debe especificar la dirección de memoria a accesar en la forma segmento:offset.
mov ax,ss:0031h

```
mov al,data:0
mov bx,DGROUP:block
```

Relocalizables.- Por medio de un símbolo asociado a una dirección de memoria y que puede ser usado también para llamados.

```
mov ax, value
call main
mov al,OFFSET dgroup:tabla
mov bx, count
```

count sólo será válido si fue declarado con la directiva DW.

Contador de localización.- Usado para indicar la actual localización en el actual segmento durante el ensamblado. Representado con el símbolo $ y también conocido como centinela.

```
help DB ´OPCIONES´,13,10
F1 DB ´ F1 salva pantalla´,13,10
.
.
.
F10 DB ´ F10 exit´,13,10,´$
DISTANCIA = $-help
```

Registros.- Cuando se hace referencia a cualquiera de los registros de propósito general, apuntadores, índices, o de segmento.

Basados.- Un operador basado representa una dirección de memoria relativa a uno de los registros de base (BP o BX). Su sintaxis es:

```
desplazamiento[BP>
desplazamiento[BX>
[desplazamiento>[BP>
[BP+desplazamiento>
[BP>.desplazamiento
[BP>+desplazamiento
```

en cada caso la dirección efectiva es la suma del desplazamiento y el contenido del registro.

```
mov ax,[BP>
mov al,[bx>
mov bx,12[bx>
mov bx,fred[bp>
```

Indexado.- Un operador indexado representa una dirección de memoria relativa a uno de los registros índice (SI o DI). Su sintaxis es:

```
desplazamiento[DI>
desplazamiento[SI>
[desplazamiento>[DI>
[DI+desplazamiento>
[DI>.desplazamiento
[DI>+desplazamiento
```

en cada caso la dirección efectiva es la suma del desplazamiento y el contenido del

registro.
mov ax,[si>
mov al,[di>
mov bx,12[di>
mov bx,fred[si>

Base-indexados.- Un operador base-indexado representa una dirección de memoria relativa a la combinación de los registros de base e índice. Su sintaxis es:
desplazamiento[BP>[SI>
desplazamiento[BX>[DI>
desplazamiento[BX>[SI>
desplazamiento[BP>[DI>
[desplazamiento>[BP>[DI>
[BP+DI+desplazamiento>
[BP+DI>.desplazamiento
[DI>+desplazamiento+[BP>
en cada caso la dirección efectiva es la suma del desplazamiento y el contenido del registro.
mov ax,[BP>[si>
mov al,[bx+di>
mov bx,12[bp+di>
mov bx,fred[bx>[si>

Estructuras.- Su sintaxis es variable.campo. variable es el nombre con que se declaró la estructura, y campo es el nombre del campo dentro de la estructura.
date STRUC
mes DW ?
dia DW ?
aa DW ?
date ENDS
actual date ´ja´,´01´,´84´
mov ax,actual.dia
mov actual.aa, ´85´

Operadores y expresiones.- Se cuenta con los siguientes operadores:
-aritméticos
expresión1 * expresión2
expresión1 / expresión2
expresión1 MOD expresión2
expresión1 + expresión2
expresión1 - expresión2
+ expresión
- expresión
-de corrimiento
expresión1 SHR contador
expresión1 SHL contador
-relacionales
expresión1 EQ expresión2

expresión1 NE expresión2
expresión1 LT expresión2
expresión1 LE expresión2
expresión1 GT expresión2
expresión1 GE expresión2
- de bit
NOT expresión
expresión1 AND expresión2
expresión1 OR expresión2
expresión1 XOR expresión2
-de índice
[expresión1> [expresión2>
ejemplos:
mov al, string[3>
mov string[last>,al
mov cx,dgroup:[1> ; igual a mov cx,dgroup:1
-de apuntador
tipo PTR expresión
tipo puede ser BYTE ó 1, WORD ó 2, DWORD ó 4, QWORD ó 8, TBYTE ó 10, NEAR ó 0FFFFh, FAR ó 0FFFEh. Ejemplos:
call FAR PTR subrout3
mov BYTE ptr [array>, 1
add al, BYTE ptr [full_word>
-de nombre de campo
estructura.campo
ejemplos:
inc month.day
mov time.min,0
mov [bx>.dest
-de propósito especial.

OFFSET expresion.- Regresa el desplazamiento del operando
mov bx, OFFSET dgroup:array
mov bx, offset subrout3

SHORT etiqueta.- Para un salto de menos de 128 bytes
jmp SHORT loop

LENGTH variable.- Regresa el número de elementos de variable según su tipo
mov cx,length array

SIZE variable.- Regresa el tamaño en bytes alojados para variable
mov cx,size array

SEG expresión.- Regresa el valor del segmento para expresión
mov ax, SEG saludo

MACROS Y PROCEDIMIENTOS.

La manera más fácil de modularizar un programa es dividirlo en dos o más partes. Para esto, es necesario que datos, símbolos, y demás valores de un módulo sean reconocidos por el otro u otros módulos. Para este tipo de declaraciones globales existen dos directivas:
PUBLIC nombre,,, que hace la variable, etiqueta o símbolo absoluto disponible para todos los programas.

EXTRN nombre:tipo,,, que especifica una variable, etiqueta o símbolo externos identificados por nombre y tipo (que puede ser BYTE, WORD, DWORD, QWORD, TBYTE, NEAR, FAR, o ABS, éste último para números absolutos).
El siguiente ejemplo ilustra el uso de las directivas. El primer listado corresponde al módulo principal, mientras que el segundo al módulo que contiene una rutina. Ambos módulos son archivos que se editan por separado, se ensamblan por separado, pero se ligan juntos.

```
MODULO PRINCIPAL: MAIN.ASM
NAME main
PUBLIC exit
EXTRN print:near
stack SEGMENT word stack 'STACK'
DW 64 DUP(?)
stack ENDS
data SEGMENT word public 'DATA'
data ENDS
code SEGMENT byte public 'CODE'
ASSUME cs:code, ds:data
start:
mov ax,data ; carga localizacion del segmento
mov ds,ax ; en el registro DS
jmp print ; va a PRINT en el otro modulo
exit:
mov ah,4ch
int 21h
code ENDS
END start
SUBMODULO: TASK.ASM
NAME task
PUBLIC print
EXTRN exit:near
data SEGMENT word public 'DATA'
entrada DB "Entrando a un submodulo....",13,10,"$"
salida DB ".....saliendo del submodulo.",01,07,13,10,"$"
data ENDS
code SEGMENT byte public 'CODE'
ASSUME cs:code, ds:data
print:
mov ah,06h ; Funcion para borrar pantalla
mov al,0 ; todas las lineas
```

```
mov cx,0 ; de 0,0
mov dh,24d
mov dl,79d
mov bh,0 ; atributo en lineas vacias
int 10h ; Servicio de e/s video
mov dx, OFFSET entrada
mov ah,09h
int 21h
mov dx, OFFSET salida
int 21h
jmp exit ; Regresa al otro modulo
code ENDS
END
```

La declaración de macros se hace a través de las directivas MACRO y ENDM. Su sintaxis es:

```
nombre MACRO [parámetros,,,>
declaraciones
ENDM
```

parámetros son los valores que se substituirán en la macro cada vez que se haga referencia a ésta.

Para la definición de procedimientos se emplean las directivas PROC y ENDP. Su sintaxis es:

```
nombre PROC [distancia>
sentencias
nombre ENDP
```

distancia, que puede ser NEAR (default) o FAR permiten indicar el tipo de acciones a realizar en brincos y llamados a subrutinas. nombre se puede usar como dirección en llamados o brincos.

INTERRUPCIONES.

Como se mencionó anteriormente la PC esta constituida lógicamente por su BIOS y sistema operativo. La mayoría de las rutinas que controlan al computador están grabadas en el ROM del BIOS, aunque muchas rutinas son establecidas por el sistema operativo y se cargan en RAM al momento de encender al computador. Estas rutinas son denominadas interrupciones y son activadas mediante la instrucción: INT número. Una interrupción es una operación que invoca la ejecución de una rutina específica que suspende la ejecución del programa que la llamó, de tal manera que el sistema toma control del computador colocando en el stack el contenido de los registros CS e IP.

El programa suspendido vuelve a activarse cuando termina la ejecución de la interrupción y son restablecidos los registros salvados. Existen dos razones para ejecutar una interrupción: (1) intencionalmente como petición para la entrada o salida de datos de un dispositivo, y (2) un error serio y no intencional, como sobreflujo o división por cero.

El operando de una interrupción indica cúal es la rutina a activar. La dirección de la rutina es localizada por medio de una tabla que el sistema mantiene a partir de la dirección 0000:0000h. Existen 256 entradas de 4 bytes de longitud, y cada interrupción

proporciona varias funciones. Las interrupciones de 00h a 1Fh correponden al BIOS y de 20h a FFh son del DOS y BASIC. El apéndice F proporciona una lista de las interrupciones para equipo XT.

III.- CREACION Y DEPURACION DE PROGRAMAS EN LENGUAJE ENSAMBLADOR EDICION.

Los archivos fuente de código ensamblador deben estar en formato ASCII standard. Para esto puede usarse cualquier editor que permita crear archivos sin formato, e.g. Edlin, Edit, Write, El editor del Turbo Pascal, Works, Word, WordStar, etcétera. Las declaraciones pueden ser introducidas en mayúsculas y/o minúsculas. Una buena práctica de programación es poner todas las palabras reservadas (directivas e instrucciones) en mayúsculas y todo lo del usuario en minúsculas para fines de facilidad de lectura del código.

Las sentencias pueden comenzar en cualquier columna, no pueden tener más de 128 caracteres, no se permiten lineas múltiples ni códigos de control, y cada línea debe ser terminada con una combinación de line-feed y carriage-return. Los comentarios se declaran con ; y terminan al final de la línea.

ENSAMBLADO.

El ensamblado se lleva a cabo invocando al MASM. Este puese ser invocado, usando una línea de comando, de la siguiente manera:

MASM archivo [,[objeto>[,[listado>[,[cross>>>>>[opciones>[;>
donde:
archivo.- Corresponde al programa fuente. Por default se toma la extensión .ASM.
objeto.- Es el nombre para el archivo objeto.
listado.- Nombre del archivo de listado de ensamblado.
cross.- Es un archivo de referencias cruzadas.
opciones.- Pueden ser:
/A escribe los segmentos en orden alfabético
/S escribe los segmentos en orden del fuente
/Bnum fija buffer de tamaño num
/C especifica un archivo de referencias cruzadas
/L especifica un listado de ensamble
/D crea listado del paso 1
/Dsym define un símbolo que puede usarse en el ensamble
/Ipath fija path para búscar archivos a incluir
/ML mantiene sensitividad de letras (mayús./minús) en nombres
/MX mantiene sensitividad en nombre publicos y externos
/MU convierte nombres a mayúsculas
/N suprime tablas en listados
/P checa por código impuro
/R crea código para instrucciones de punto flotante
/E crea código para emular instrucciones de punto flotante
/T suprime mensajes de ensable exitoso

/V despliega estadísticas adicionales en pantalla
/X incluir condicionales falsos en pantalla
/Z despliega líneas de error en pantalla
si el ; al final se omite es necesario poner todas las comas que se indican. Si no se quiere poner algún valor basta con dejar la coma.

otra forma de invocar al ensamblador es sólo tecleando MASM y respondiendo a la información que se solicita. Para omitir algún valor sólo basta teclear ENTER si dar ningún valor.

L I N K .

De la misma forma que el ensamblado, la fase de liga se lleva a cabo con el LINK. Este puede ser invocado de la misma forma que el MASM. Los parámetros que este requiere son:

LINK objeto [,[ejecutable>[,[mapa>[,[librería>>>>>[opciones>[;>
donde:
objeto.- Es el nombre para el archivo .OBJ
ejecutable.- Nombre del archivo .EXE
mapa.- Nombre del archivo mapa
librería.- Nombre del archivo biblioteca de rutinas
opciones.- Pueden ser:
/HELP muestra lista de opciones
/PAUSE pausa en el proceso
/EXEPACK empaca archivo ejecutable
/MAP crea mapa se símbolos públicos
/LINENUMBERS copia número de lineas al mapa
/NOIGNORECASE mantiene sensitividad en nombres
/NODEFAULTLIBRARYSEARCH no usa bibliotecas por default
/STACK:size fija el tamaño del stack a usar
/CPARMAXALLOC:número fija alojación máxima de espacio
/HIGH fija la dirección de carga más alta
/DSALLOCATE aloja grupo de datos
/NOGROUPASSOCIATION ignora asociaciones para direcciones
/OVERLAYINTERRUPT:número asigan nuevo número a la INT 03Fh
/SEGMENTS:número procesa un número de segmentos
/DOSSEG sigue la convención de orden de DOS

EJECUCION.

Para la ejecución del programa simplemente basta teclear su nombre en el prompt de MS-DOS y teclear ENTER. Con esto el programa será cargado en memoria y el sistema procederá a ejecutarlo. Lo que se vería en pantalla sería lo siguiente:

C:\DATA\PROGRAMS\ASM>masm main
Microsoft (R) Macro Assembler Version 4.00

```
Copyright (C) Microsoft Corp 1981, 1983, 1984, 1985. All rights reserved.
Object filename [main.OBJ>:
Source listing [NUL.LST>:
Cross-reference [NUL.CRF>:
50966 Bytes symbol space free
0 Warning Errors
0 Severe Errors
C:\DATA\PROGRAMS\ASM>masm task
Microsoft (R) Macro Assembler Version 4.00
Copyright (C) Microsoft Corp 1981, 1983, 1984, 1985. All rights reserved.
Object filename [task.OBJ>:
Source listing [NUL.LST>:
Cross-reference [NUL.CRF>:
51034 Bytes symbol space free
0 Warning Errors
0 Severe Errors
C:\DATA\PROGRAMS\ASM>link main+task
Microsoft (R) 8086 Object Linker Version 3.05
Copyright (C) Microsoft Corp 1983, 1984, 1985. All rights reserved.
Run File [MAIN.EXE>:
List File [NUL.MAP>:
Libraries [.LIB>:
C:\DATA\PROGRAMS\ASM>main
Entrando a un submodulo....
.......saliendo del submodulo.
C:\DATA\PROGRAMS\ASM>
```

DEPURACION.

Para la depuración de un programa en ensamblador tenemos disponibles dos herramientas. Por un lado tenemos el debuger que nos proporciona MS-DOS (DEBUG.EXE) y por otro lado tenemos el que nos proporciona Microsoft (SYMDEB.EXE). Este último trabaja igual que el de MS-DOS pero nos proporciona muchas ventajas más. Una de ellas es la facilidad de desplegar el código fuente correspondiente a la instrucción que se esta ejecutando (si el programa ejecutable fue ensamblado o compilado con un ensamblador o compilador compatible), nos permite ejecutar comandos del S.O. y nos permite obtener información de las interrupciones de manera simbólica.

LA UTILERIA EXE2BIN Y LOS ARCHIVOS .EXE Y .COM .

Para MS-DOS sólo existen dos tipo de archivos ejecutables los .COM y .EXE. Ambos archivos difieren en algunas cosas. Primero, las ventajas de los .EXE son dobles, nos permiten tener archivos reubicables y el uso de hasta cuatro segmentos (STACK, DATA, EXTRA y CODE) de hasta 64KB cada uno. Un archivo .COM sólo puede tener un segmento de 64KB, en el que se tiene tanto código como pila, y datos. La desventaja de los .EXE es que agregan 512 bytes como cabecera con información para la reubicación del código. Un .COM no es reubicable, siempre inicia en la dirección 0100H.

Si nuestro programa no es muy grande 64KB son mas que suficientes. Por lo que conviene crearlo como .COM, para esto se cuenta con la utilería EXE2BIN.EXE que nos proporciona el sistema operativo. y que nos permite crear .COM a partir de .EXE . Las restricciones para esto son las suguientes: el archivo a convertir no debe estar empacado, no debe tener segmento de stack, debe tener sólo segmento de código y su tamaño debe ser menor a 64KB.

III.-PROGRAMACIÓN EN ENSAMBLADOR
PROGRAMACIÓN BÁSICA

Para comenzar con la programación en lenguaje ensamblador, es necesario contar con un conjunto de herramientas de programación. Este conjunto de herramientas consta de un editor de texto capaz de producir archivos en código ASCII, un ensamblador y un enlazador.

Para propósitos de este trabajo, se utilizaran los siguientes programas:

- El ensamblador PASS32
- El editor de texto EDIT proporcionado con todas las versiones de MS-DOS y Windows.

PASS32 es un ensamblador y enlazador capaz de producir código ejecutable de 16 y 32 bits.

Formatos de instrucciones

En el lenguaje ensamblador existen tres tipos de instrucciones: instrucciones con dos operandos, instrucciones con un operando e instrucciones con operandos implícitos.

El campo nemónico es ocupado por cualquiera de las instrucciones que forman parte del conjunto de la familia x86.
Ejemplo: Mov (Transferir)

En los dos campos siguientes Reg significa que el operando puede ser un registro, Mem indica que puede ser una dirección de memoria y dato indica que el operando puede ser un dato colocado directamente por el programador. Los campos dos y tres se encuentran entre paréntesis cuadrados para indicar que son opcionales en algunas instrucciones.

Los siguientes son algunos ejemplos de instrucciones de las tres formas:

Instrucción con dos operandos:

Mov AX,BX

En este caso Mov es el nemónico, AX es el operando 1 y BX es el operando 2.

Instrucción con un operando:

INC BX

En este caso INC es el nemónico y BX es el único operando.

Iinstrucciones con operandos implícitos o sin operandos:

PUSHF

Donde PUSHF es el nemónico y único elemento de la instrucción.

FORMATO DE UN PROGRAMA
El código fuente de un programa en lenguaje ensamblador se constituye:

El siguiente listado se utilizará para mostrar las diferentes partes.

```
.COMMENT
 ************************************************************************
 PRIMERO.ASM     Ejemplo de un programa en lenguaje ensamblador.
************************************************************************
.MODEL TINY; Modelo de memoria para el programa
.DATA; Declaración de variables
Mensaje       db     'Mi primer programa',10,13,'$'
.CODE; Cuerpo del programa
INICIO:; Punto de entrada al programa
mov dx,OFFSET Mensaje   ; Dirección de la cadena de texto
mov ah,9                ; Función para imprimir cadenas
int 21h                 ; Llamada al sistema operativo
mov ah,4ch              ; Función para terminar un programa
int 21h                 ; Llamada al sistema operativo
END INICIO              ; Fin del bloque principal del programa
END
```

Un programa en lenguaje ensamblador se compone de las siguientes partes:

- Área de comentarios
- Definición del modelo de memoria
- Área de datos
- Cuerpo del programa

El área de comentarios sirve para incluir comentarios acerca del programa que se está elaborando, comienza con la directiva .COMMENT y el comentario es colocado entre dos caracteres '*'.

La definición del modelo de memoria es la parte donde se indica que tipo de código se va generar (16 o 32 bits). En este trabajo sólo se escribirán programas ejecutables .COM, por lo que siempre se usa la directiva .MODEL TINY.

El área de datos es el lugar donde deben ser declaradas las constantes y variables del programa. Las variables son declaradas después de la directiva .DATA y las constantes después de .CONST.

En el cuerpo del programa es donde se colocan las instrucciones en lenguaje ensamblador que se encargarán de realizar las tareas deseadas. El cuerpo del programa comienza con la directiva .CODE y termina con la directiva END. Esta parte corresponde al Begin y End de un programa en lenguaje Pascal.

Adicionalmente se debe indicar un punto de entrada al programa. El punto de entrada se indica por medio de una etiqueta antes de la primer instrucción real del programa. En el ejemplo anterior el punto de entrada es INICIO: y el punto final de las instrucciones se indica por medio de la instrucción END INICIO.
Cuando se requiere comentar las instrucciones de un programa, se debe colocar un punto y coma (;) y así el ensamblador interpreta todo lo que sigue como un comentario de una

sola línea. Si requiere comentarios de más de una línea puede usar la directiva .COMMENT.

PROCESO DE ENSAMBLE Y LIGADO DE UN PROGRAMA
Este proceso es muy sencillo y se describe a continuación:

Si está trabajando en MS-DOS siga estos pasos:

1.- Escriba el programa, tal y como aparece en el listado anterior, usando su editor de texto preferido.

2.- Guárdelo con algún nombre y la extensión .ASM.

3.- En el símbolo del MS-DOS escriba lo siguiente

C:\PASS32\BIN\>PASS32 Nombre.ASM –t <Enter>

4.- Ejecute el programa .COM que se genera.

Para probar el programa abra una ventana de MS-DOS y seleccione el programa haciendo doble clic sobre el icono.

Directivas de ensamble (Seudo instrucciones)

Pass32 cuenta con algunas palabras reservadas que cumplen tareas especiales para facilitar la programación en ensamblador, estas palabras son llamadas seudo instrucciones o directivas de ensamble.

La siguiente es una lista de las directivas de ensamble más utilizadas en Pass32:

DB	Reserva un byte en memoria
DW	Reserva una palabra (Word) en memoria o 2 bytes
DD	Reserva una palabra doble (Double Word)
.EQU	Se utiliza para reemplazar símbolos por valores
PROC-ENDP	Se utilizan para declarar procedimientos en los programas
.**MACRO-ENDM**	Se utilizan para declarar macros
DUP	Sirve para inicializar cadenas de caracteres o arreglos numéricos
.INCLUDE	Se utiliza para obtener datos o subrutinas de otros programas
.EXTERN	Declara un símbolo como externo, trabaja en conjunto con .INCLUDE
.PUBLIC	Declara un símbolo como público.

Instrucciones de transferencia de datos

Los microprocesadores 80x86 cuentan con algunas instrucciones básicas de transferencia de información de acuerdo con los modos de direccionamiento. Las instrucciones más representativas del grupo de transferencia son:

MOV.- Transfiere (copia) contenidos. Su formato es MOV OP1,OP2. Esta instrucción copia el contenido de OP2 en OP1. Ejemplo:

```
Mov AX,0      ; AX=0
```

LEA.- Carga un registro con la dirección de desplazamiento de alguna variable en memoria. Su formato es LEA REG,Variable. Ejemplo:

.Data

Mensaje db 'Hola','$'

.Code

Lea DX,Mensaje ;DS:DX->Mensaje

LDS.- Inicializa el registro DS

LES.- Inicializa el registro ES

Nota:

Las instrucciones LDS y LES modifican directamente el contenido de los registros de segmento DS y ES, por lo cual se recomienda que sólo sean utilizadas por programadores avanzados.

XCHG.- Intercambia contenidos. Su formato es XCHG OP1,OP2. El resultado es que el contenido de OP2 se pasa a OP1 y el de OP1 se pasa a OP2.

Ejemplo:

XCHG AX,BX ; AX->BX, BX->AX

El siguiente programa muestra la forma de usar las instrucciones de transferencia, además de algunas directivas de ensamble.

Inicialmente, el programa define un arreglo de 10 elementos llamado Array1, y lo inicializa con ceros. Después, utilizando la instrucción Mov, copia el número 10 en el registro AX y el 5 en el registro BX. Por medio de la instrucción Lea, el registro DX es cargado con la dirección de memoria donde comienza Array1 y finalmente intercambia el contenido de los registros AX y BX por medio de la instrucción XCHG.

Debido a que el ensamblador es un lenguaje de bajo nivel, es decir que el programador se encuentra en contacto directo con los componentes de la computadora, no existen instrucciones que nos permitan ver el contenido de los registros o los resultados de las operaciones en pantalla, es por esto que la mayoría de los programas no muestran datos en pantalla.

```
.COMMENT
*Programa: Trans1.ASM
Descripción: Este programa ilustra es uso de las operaciones para transferencia de datos.
El programa realmente no hace nada que sea visible al usuario, es solo con fines
ilustrativos.
*
.MODEL tiny
.DATA
Array1  db  10 dup (0)      ;Arreglo de 10 elementos iniciali-
;zados en cero.
.CODE
```

```
inicio:              ;Punto de entrada al programa
mov AX,10       ;Copiar el número 10 dentro de AX
mov BX,5        ;Copiar le número 5  dentro de BX
lea DX,Array1    ;DX contiene la dirección efectiva de Array1[0]
xchg AX,BX      ;Intercambiar los valores contenidos en AX y BX
mov ax,4C00h     ;Terminar programa y salir al DOS
int 21h
END inicio
END
```

Instrucciones aritméticas

Existen 8 instrucciones aritméticas básicas: ADD (Suma), SUB (Resta), MUL (Multiplicación sin signo), DIV (División sin signo), IMUL (Multiplicación con signo), IDIV (División con signo), INC (Incremento unitario) y DEC (Decremento unitario).
Las instrucciones ADD y SUB permiten realizar sumas y restas sencillas y tienen el siguiente formato:

ADD Destino, Fuente

SUB Destino, Fuente

Ejemplos:

```
ADD AX,BX              ;AX=AX+BX

ADD AX,10              ;AX=AX+10

SUB  AX,BX             ;AX=AX-BX

SUB  AX,10             ;AX=AX-10
```

En las operaciones de suma y resta el resultado siempre es almacenado en el operando de destino, el cual puede ser un registro o una variable.

Las instrucciones INC y DEC permiten incrementar los contenidos de los registros y de las variables almacenadas en memoria.

Ejemplos:

```
INC AX ;AX=AX+1

INC VAR1        ;VAR1=VAR1+1

DEC AX          ;AX=AX-1

DEC VAR1        ;VAR1=VAR1-1
```

El siguiente programa muestra la forma de utilizar estas instrucciones básicas:

```
.COMMENT
*Programa: Addsub.ASM
Descripción: Este programa ilustra el uso de las instrucciones ADD, SUB, INC y DEC.
*MODEL TINY
.DATA
Var1   DW  10   ;Declaración de una variable de tipo entero
;inicializada con 10.
.CODE
```

```
Inicio:              ;Punto de entrada al programa
Mov AX,5       ;AX=5
Mov BX,10      ;BX=10
Add AX,BX      ;AX=AX+BX
Mov CX,8       ;CX=8
Add CX,Var1    ;CX=CX+Var1
Inc AX         ;AX=AX+1
Dec BX         ;BX=BX-1
Inc Var1       ;Var1=Var1+1
Dec Var1       ;Var1=Var1-1
Mov AX,4C00h   ;Terminar programa y salir al DOS
Int 21h        ;
END Inicio
END
```

Multiplicación

Por otro lado, las operaciones de multiplicación y división son un poco más complejas de utilizar, esto se debe a que debemos tomar en cuenta el tamaño de los operandos para no sobrepasar la capacidad de almacenamiento de los registros del microprocesador.

Existen dos instrucciones para la multiplicación, estas son: MUL e IMUL. MUL permite realizar operaciones de multiplicación entre operandos sin signo e IMUL permite realizar operaciones entre operandos con signo.

La multiplicación se puede efectuar entre bytes (8 bits), palabras (16 bits) o dobles palabras (32 bits). Solamente los microprocesadores 386 y posteriores pueden realizar multiplicaciones entre operandos de 32 bits.

El producto de una multiplicación siempre tiene el doble de ancho. Si se multiplican dos números de 8 bits, el resultado será de 16 bits; si se multiplican dos números de 16 bits, el producto será de 32 bits y, finalmente, si se multiplican cantidades de 32 bits, el resultado será un número de 64 bits.

En la multiplicación de 8 bits, con o sin signo, el multiplicando está siempre en el registro AL. El multiplicador puede ser cualquier registro de 8 bits o cualquier variable. El resultado de la multiplicación se almacena en el registro AX, que es de doble ancho que los operandos involucrados.

Ejemplos válidos de operaciones de multiplicación son los siguientes:

```
MOV BL,5          ;Cargar datos

MOV AL,10         ;

MUL BL            ; AX=AL*BL

MOV AL,10

MUL  número1      ; AX=AL*número1

; donde número1 es una variable de tipo byte.
```

En la multiplicación de 16 bits, el multiplicando debe ser colocado en el registro AX y el resultado siempre aparece en el par de registros DX:AX. El registro DX contiene los 16 bits más significativos de producto, mientras que el registro AX contiene los 16 bits menos significativos del resultado.Ejemplos:

```
MOV AX,400   ;Cargar datos

MOV CX,100   ;

MUL CX          ; DX:AX=AX*CX

MOV AX,400   ;

MUL numero2   ; DX:AX=AX*numero2
```

El siguiente programa muestra la forma de utilizar algunas de estas operaciones de multiplicación en sus diferentes formatos. Debido a que el ensamblador no cuenta con funciones para imprimir información numérica en la pantalla, no es posible mostrar los resultados; considere este ejemplo únicamente con fines ilustrativos.

```
.COMMENT
*
Programa: Mul.ASM
Autor: Juan Carlos Guzmán C.
Descripción: Este programa ilustra el uso de las instrucciones MUL e IMUL.
*
.MODEL TINY
.DATA
NUM1 dw  3
NUM2 db  -5
.CODE
INICIO:
;MULTIPLICACIÓN DE 8 BITS CON REGISTROS
MOV  BH,4    ;BH=4
MUL  BH      ;AX=AL*BH
;MULTIPLICACIÓN DE 16 BITS
MOV  AX,-3   ;AX=-3
MUL  NUM1    ;DX:AX=AX*NUM2
;MULTIPLICACIÓN DE 8 BITS CON VARIABLES
MOV  AL,3    ;AL=3
IMUL NUM2    ;AX=AL*NUM2
MOV  AX,4c00h
INT  21h
END INICIO
END
```

División

Las instrucciones para división permiten realizar divisiones de 8, 16 o 32 bits (esta última sólo está disponible en los microprocesadores 386 y posteriores). Los operandos pueden ser números con signo (IDIV) o números sin signo (DIV). El dividendo siempre tiene el doble de ancho que el operando divisor. Esto significa que en una división de 8 bits se divide un número de 16 bits entre uno de 8; en una de 16 bits se divide un número de 32 bits entre uno de 16, etc.

En la división de 8 bits, el dividendo es almacenado en el registro AX y el divisor puede ser cualquier registro de 8 bits o cualquier variable declarada de tipo byte. Después de la división, el cociente es cargado en el registro AL y el residuo en el registro AH.

Ejemplo de división sin signo:

```
MOV AX,10

MOV BL,5

DIV BL
```

Ejemplo de división con signo:

```
MOV AL,-10

MOV BL,2

CBW

IDIV BL
```

En este último ejemplo, el dividendo es cargado en el registro AL, pero debido a las reglas del microprocesador el dividendo debe ser de 16 bits; para lograr esto, se utiliza una instrucción especial. La instrucción CBW (convertir byte a palabra) permite convertir un número de 8 bits con signo en AL en un número de 16 bits con signo en AX.

En la división de 16 bits se siguen las mismas reglas que en la división de 8 bits, sólo que en ésta, el dividendo se encuentra en los registro DX:AX. Después de la división el cociente es almacenado en el registro AX y el residuo en el registro DX. En el caso de la división con signo, existe una instrucción que permite convertir un número con signo de 16 bits en AX en un número con signo de 32 bits en DX:AX.

El siguiente programa muestra la forma de utilizar algunas de estas operaciones de división en sus diferentes formatos. Debido a que el ensamblador no cuenta con funciones para imprimir información numérica en la pantalla, no es posible mostrar los resultados; considere este ejemplo únicamente con fines ilustrativos.

```
.COMMENT
*
Programa: Div.ASM
Autor: Juan Carlos Guzmán C.
Descripción: Este programa ilustra el uso de las instrucciones DIV e IDIV.
*
.MODEL TINY
.DATA
NUM1 db  3
NUM2 db  -5
.CODE
INICIO:                 ;INICIO DEL PROGRAMA
MOV  AX,100       ;AX=100
MOV  BH,10        ;BH=10
DIV  BH          ;DIVISION DE 8 BITS SIN SIGNO
MOV  AX,100       ;AX=100
DIV  NUM1         ;DIVISION DE 8 BITS SIN SIGNO CON VARIABLES
MOV  AL,-10       ;AX=-10
CBW              ;EXTENSIÓN DE SIGNO A 16 BITS
IDIV num2         ;DIVISION DE 8 BITS CON SIGNO
MOV  AX,4c00h     ;FIN DEL PROGRAMA
```

```
INT 21h        ;
END INICIO
END
```

Instrucciones para la manipulación de banderas

El registro de banderas tiene diferentes funciones en cada uno de sus bits, algunos de estos bits (banderas) pueden ser controlados por instrucciones directas de bajo nivel; sin embargo, se debe tener en cuenta que estas banderas están íntimamente ligadas con funciones internas del microprocesador, por ejemplo la línea INTR (interrupción por hardware), acarreos, etc., y que al manipularlas incorrectamente podemos llegar al extremo de bloquear la computadora. Es por esto que se recomienda que sólo programadores experimentados modifiquen dichas banderas.

En esta sección se explicarán algunas de las instrucciones más comunes y sus aplicaciones, pero no se desarrollarán programas por razones de seguridad.

Control de interrupción

La terminal INTR del microprocesador puede ser activada o desactivada directamente por los programas por medio de las instrucciones STI y CLI. STI carga un 1 en IF, con lo cual INTR queda habilitada; por otro lado, CLI carga un cero en IF, con lo cual las interrupciones externas o por hardware quedan deshabilitadas.

Control de la bandera de acarreo

La bandera de acarreo, CF, es la encargada de indicar cuando ha ocurrido un acarreo o préstamo en operaciones de suma o resta, también indica errores en la ejecución de procedimientos. Existen tres instrucciones básicas para su manipulación: STC (activar acarreo), CLC (desactivar acarreo) y CMC (Complementar acarreo).

Control de la bandera de dirección

La bandera de dirección, DF, es utilizada para establecer el sentido en el que las cadenas de datos serán procesadas por los programas. Un cero en DF indica procesamiento de izquierda a derecha, mientras que un uno indica procesamiento de derecha a izquierda.

Para controlar esta bandera, existen dos instrucciones, CLD (limpiar bandera) y STD (establecer bandera). STD coloca un uno y CLD coloca un cero.

Estas instrucciones serán aplicadas más adelante en el capítulo 3, en el cual se desarrollan varios programas para mostrar su uso.

Instrucciones de comparación y prueba

Existen dos instrucciones especiales en el microprocesador 8086: CMP y TEST. CMP (Comparar) compara si dos valores son iguales o diferentes. Su funcionamiento es similar al de la instrucción SUB (restar), sólo que no modifica el operando de destino, solamente modifica las banderas de signo (SF), de cero (ZF) y de acarreo (CF).

Por ejemplo:

```
CMP AX,1235
```

Esta instrucción compara si el valor almacenado en el registro AX es igual que el valor 1235 en decimal.

Por otro lado, la instrucción TEST realiza la operación AND de los operandos especificados sin que el resultado se almacene en algún registro, modificando únicamente ciertas banderas. Su aplicación más común es la de probar si algún bit es cero.

Ejemplo:

Test AL,1

Esta instrucción prueba si el bit menos significativo de AL es 1 y

Test AL,128

prueba si el bit más significativo de AL es 1.

Por lo general estas instrucciones van seguidas de alguna de las instrucciones de salto, las cuales se estudian en otra sección.

Instrucciones de salto

En los lenguajes de alto nivel como Pascal y C, los programadores pueden controlar el flujo de los programas por medio de instrucciones condicionales compuestas; por ejemplo, en Pascal el siguiente conjunto de instrucciones permite tomar una decisión sobre el flujo del programa:

```
IF A=5 then

write("Error...");

else

A:=A+1;
```

En contraste, el lenguaje ensamblador no proporciona tales mecanismos. Este tipo de decisiones se realizan por medio de una serie de instrucciones que van teniendo un significado consecutivo; es decir, el efecto de la instrucción siguiente depende del resultado anterior.

El lenguaje ensamblador proporciona un conjunto de instrucciones conocidas como instrucciones de salto. Estas instrucciones son utilizadas en conjunto con instrucciones de comparación y prueba para determinar el flujo del programa.

Existen dos tipos de instrucciones de salto: las instrucciones de salto condicional y las de salto incondicional.

Las instrucciones de salto condicional, revisan si ha ocurrido alguna situación para poder transferir el control del programa a otra sección, por ejemplo:

```
CMP AX,0

JE  otro

.....

.....

otro:

.....

.....

End
```

En este ejemplo, la instrucción JE (Salta si es igual) revisa si la prueba implícita en la instrucción anterior resultó positiva, es decir, si la comparación de AX con 0 fue cierta. En caso de que AX sea igual a 0, JE transfiere el control del programa a las instrucciones que se encuentran después de la etiqueta "otro". En caso contrario ejecuta las instrucciones siguientes a JE.

Por otro lado, las instrucciones de salto incondicional (sólo existe una) permiten cambiar el flujo del programa sin verificar si se cumplió alguna condición.

Ejemplo:

```
Mov AX,10

Jmp  otro

.....

.....

otro:

.....

.....
```

En este ejemplo, inmediatamente después de cargar el registro AX con el valor de 10, se transfiere el control del programa a la instrucción que sigue después de la etiqueta "otro".

La siguiente es una lista de las instrucciones de salto condicional y su descripción:

JA o JNBE: Salta si está arriba o salta si no está por debajo o si no es igual (jump if above or jump if not below or equal). El salto se efectúa si la bandera de CF=0 o si la bandera ZF=0.

JAE o JNB: Salta si está arriba o es igual o salta si no está por debajo (jump if above or equal or jump if not below). El salto se efectúa si CF=0.

JB o JNAE: Salta si está por debajo o salta si no está por arriba o es igual (jump if below or jump if not above or equal). El salto se efectúa si CF=1.

JBE o JNA: Salta si está por debajo o es igual o salta si no está por arriba (jump if below or equal or jump if not above). El salto se efectúa si CF=1 o ZF=1.

JE o JZ: Salta si es igual o salta si es cero (jump if equal or jump if zero). El salto se efectúa si ZF=1.

JNE o JNZ: Salta si no es igual o salta si no es cero (jump if not equal or jump if not zero). El salto se efectúa si ZF=0.

JG o JNLE: Salta si es mayor o salta si no es menor o igual (jump if greater or jump if not less or equal). El salto se efectúa si ZF=0 u OF=SF.

JGE o JNL: Salta si es mayor o igual o salta si no es menor (jump if greater or equal or jump if not less). El salto se efectúa si SF=OF.

JL o JNGE: Salta si es menor o salta si no es mayor o igual (jump if less or jump if not greater or equal). El salto se efectúa si SF<>OF

JLE o JNG: Salta si es menor o igual o salta si no es mayor (jump if less or equal or jump if not greater). El salto se efectúa si ZF=1 o SF<>OF.

JC: Salta si hay acarreo (jump if carry). El salto se efectúa si CF=1.
JNC: Salta si no hay acarreo (jump if no carry). El salto se efectúa si CF=0.

JNO: Salta si no hay desbordamiento (jump if no overflow). El salto se efectúa si OF=0.

JNP o JPO : Salta si no hay paridad o salta si la paridad es non (Jump if no parity or jump if parity odd). El salto se efectúa si PF=0.

JNS: Salta si no hay signo (jump if not sign). El salto se efectúa si SF=0.

JO: Salta si hay sobreflujo (jump if overflow). El salto se efectúa si OF=1.

JP o JPE: Salta si hay paridad o salta si la paridad es par (jump if parity or jump if parity even). El salto se efectúa si PF=1.

JS: Salta si hay signo (jump if sign). El salto se efectúa si SF=1.

El siguiente programa muestra la forma de utilizar instrucciones de saltos condicionales:

```
.COMMENT
*
Programa: Jumps1.Asm
Descripción: Este programa ilustra el uso de las instrucciones de salto condicional e
incondicional
*
.MODEL TINY
.DATA
cad1    db  'Las cantidades son iguales...',13,10,'$'
cad2    db  'Las cantidades no son iguales...',13,10,'$'
.CODE
INICIO:             ;Punto de entrada al programa
Mov ax,10     ;AX=10
Mov bx,10     ;BX=10
Cmp ax,bx     ;Es AX=BX?
Je  igual     ;S¡, entonces saltar a la etiqueta igual
Lea dx,cad2    ;No, entonces imprimir Cad2
Mov ah,09h     ;
Int 21h       ;
Jmp salir     ;saltar a la etiqueta salir
igual:
Lea dx,cad1    ;imprimir el mensaje en cad1
Mov ah,09h     ;
Int 21h       ;
salir:
Mov ax,4c00h   ;Salir
Int 21h       ;
END INICIO
END
```

Este programa ilustra de forma básica la utilización de las instrucciones de salto, tanto condicionales como incondicionales.

Primeramente, el programa inicializa los registros AX y BX con el valor 10 en decimal; después utiliza la instrucción CMP para comparar el contenido de ambos registros; la instrucción JE (Salta si es igual) verifica la bandera de cero ZF, si ZF=1 significa que los contenidos son iguales y por lo tanto efectúa el salto hacia la etiqueta "Igual", en caso de que ZF=0 el programa continúa su flujo normal hasta encontrar la instrucción JMP; en este caso la instrucción JMP se utiliza para evitar llegar a la etiqueta "Igual" en el caso de que los datos sean diferentes.

El formato para utilizar las instrucciones de salto es idéntico al mostrado en este programa, solamente hay que identificar cual es la condición que queremos probar, para de esta forma seleccionar adecuadamente la instrucción de salto.

Instrucciones para ciclos

El lenguaje ensamblador cuenta con una instrucción muy poderosa que permite la programación de ciclos finitos, la instrucción **LOOP.**

Esta instrucción trabaja en forma conjunta con el registro contador CX. El formato general de esta instrucción es:

```
Mov CX,No_Veces

Etiqueta:

-------

Loop Etiqueta
```

La instrucción LOOP ejecuta las instrucciones que se encuentran entre la Etiqueta: y Loop Etiqueta el numero de veces que indique el campo No_Veces.

Por ejemplo, el siguiente grupo de instrucciones incrementa en 1 el registro AX, esto lo repite 10 veces.

```
Mov CX,10     ;10 veces

Otro:

Inc AX          ; AX=AX+1

Loop Otro
```

La instrucción Loop decrementa el registro CX en cada iteración y se detiene cuando CX es igual a cero.

El siguiente programa da un ejemplo más ilustrativo:

```
.COMMENT
*
Programa: Loop.ASM
Descripción: Este programa calcula la sucesión de Fibonacci para los 10 primeros términos de la serie, utilizando para ello un ciclo controlado por la instrucción Loop.
La sucesión está formada por números, de modo tal que cada número es la suma de los dos anteriores-
Ejemplo:
1,1,2,3,5,8,13,21,34,55....
```

```
*
.MODEL tiny
.CODE
Inicio:              ;Punto de entrada al programa
Mov AX,0       ;AX=0
Mov BX,1       ;BX=1 Estos son los dos primeros elementos 0+1=1
Mov CX,10      ;Repetir 10 veces
Repite:
Mov DX,AX      ;DX=AX
Add DX,BX      ;DX=AX+BX
Mov AX,BX      ;Avanzar AX
Mov BX,DX      ;Avanzar BX
Loop Repite    ;siguiente número
Mov AX,4C00h   ;Terminar programa y salir al DOS
Int 21h        ;
END Inicio
END
```

Instrucciones lógicas

El microprocesador 8086 cuenta con un grupo de instrucciones lógicas que operan a nivel de bit, estas instrucciones son: AND, OR, XOR y NOT.

A continuación se muestran las tablas de verdad de estas instrucciones:

Las instrucciones que se enlistan antes requieren dos operandos, a excepción de la operación NOT que sólo requiere uno.

En la figura se puede observar que para la operación **AND,** si los dos operandos son 1, el resultado será 1, en cualquier otra situación será 0.

La operación **OR** establece el resultado a 1 si cualquiera de los dos operandos es 1, de lo contrario el resultado será 0.

La instrucción **XOR** coloca en 0 el resultado si los operandos son iguales, de lo contrario establece 1.

Finalmente, la instrucción **NOT** cambia de estado todos los bits del operando, los unos por ceros y los ceros por unos.

La principal aplicación de estas instrucciones es el enmascaramiento de información. La operación AND nos permite poner a cero cualquier bit de un dato; la

operación OR nos permite poner a uno cualquier bit de un dato y la operación XOR permite borrar el contenido de algún registro o localidad de memoria, así como para negar algún bit.

El siguiente programa muestra la forma de utilizar estas instrucciones:

```
.COMMENT
*
Programa: AndOr.ASM
Descripción: Este programa ilustra el uso de las instrucciones
AND, OR, XOR y NOT.
*.MODEL TINY
.DATA
Mascara1    db  11111110b
Mascara2    db  00000001b
Dato1       db  11111111b
Dato2       db  00000000b
.CADE
INICIO:
Mov cx,0000h            ;CX=0;
Mov al,dato1            ;al=dato1
And al,mascara1                 ;al=al and mascara1
Mov ah,dato2            ;ah=dato2
Or  ah,mascara2                 ;ah=ah or mascara2
Xor bx,bx                       ;bx=0
Not cx                  ;cx=not cx
Mov ax,4c00h
Int 21h
END INICIO
END
```

El programa del listado 8 declara cuatro variables de tipo byte: Mascara1, Mascara2, Dato1 y Dato2; después inicializa CX=00h, Al=FFh, Ah=00h; al aplicar una operación and de FFh y FEh, el resultado es FEh, en otras palabras, se apagó el bit menos significativo de al; la siguiente operación es un OR entre 00 y 01, lo cual da como resultado que se encienda el bit menos significativo del Ah, el resultado es 01. La siguiente operación es XOR BX,BX, la cual al ser aplicada sobre el mismo operando da como resultado que dicho operando sea borrado. Por ultimo, la operación NOT CX cambia todos los bits de 0 a 1 y viceversa, por lo cual CX=11h.

Instrucciones de rotación y desplazamiento

El microprocesador cuenta con un conjunto de instrucciones que permiten la manipulación de las posiciones individuales de los bits dentro de un registro o localidad de memoria, estas instrucciones se encuentran divididas en dos grupos: instrucciones de rotación e instrucciones de desplazamiento (también conocidas como instrucciones para corrimientos).

Las instrucciones para rotación son cuatro y nos permiten mover de forma cíclica los bits que forman parte de un registro o localidad de memoria, estas instrucciones **son ROL, ROR, RCL , RCR.**

ROL y ROR funcionan de forma muy semejante; al ejecutar una instrucción ROL, el bit más significativo del dato es desplazado hacia la bandera de acarreo y también hacia la posición del bit memos significativo, por lo cual todos los bits restantes son rotados o

movidos hacia la izquierda. La instrucción ROR funciona igual, sólo que ésta trabaja hacia la derecha.

Las instrucciones **RCL y RCR** permiten la rotación de los bits de una localidad de memoria o registro, considerando también el contenido de la bandera de acarreo. En el caso de RCL, el bit más significativo pasa hacia la bandera de acarreo, el bit que se encontraba en la bandera de acarreo pasa al bit menos significativo y finalmente los bits restantes son rotados hacia la izquierda. La instrucción RCR funciona igual, pero aplica su rotación hacia la derecha.

Para ilustrar el uso de estas instrucciones, tomaremos como ejemplo la instrucción ROL (Rotación a la izquierda).

Las instrucciones de rotación y desplazamiento tienen diferentes formas de utilizarse dependiendo del modelo del microprocesador, los siguientes ejemplos muestran estas formas:

En el microprocesador 8086 existen dos formas, con contador implícito y con contador explícito.

La forma con contador implícito se utiliza para realizar una sola rotación a la vez y tiene el siguiente formato:

```
ROL AX,1    ;Rotar AX un bit
```

La forma con contador explícito se utiliza para realizar rotaciones n veces sobre un registro o localidad de memoria:

```
MOV CL,3     ;Número de rotaciones
ROL AX,CL    ; Rotar AX 3 veces
```

En el microprocesador 80386 y superiores existe una variante de contador implícito, la cual nos permite establecer el contador directamente como un operando, su forma es la siguiente:

```
ROL AX,3     ; Rotar AX 3 veces, sólo en 80386 y posteriores
```

En el caso de las instrucciones de desplazamiento, también existen cuatro: SHL, SHR, SAL, SAR.

SHL y SHR se utilizan para desplazar los bits de un registro o localidad de memoria, sin considerar el signo de su contenido.

SAL y SAR se utilizan para desplazar los bits de un registro o localidad de memoria, considerando su contenido como una cantidad con signo.

Las instrucciones SHL y SHR funcionan de forma idéntica, sólo que en sentidos opuestos. La instrucción SHL inserta un 0 en la posición del bit menos significativo y desplaza todos los demás bits una posición hacia la izquierda, colocando el bit más significativo en la bandera de acarreo.

La instrucción SHR inserta un 0 en la posición más significativa, desplaza todos los bit una posición hacia la derecha y finalmente coloca el bit menos significativo en la bandera de acarreo.

Algunos ejemplos de su uso son los siguientes:

```
SHL AX,1    ; Desplaza el contenido de AX una posición a la izquierda
MOV CX,3    ; Número de veces
SHR AX,CL   ; Desplaza AX 3 veces hacia la derecha
SHL BX,4    ; Desplaza BX 4 veces hacia la izquierda, sólo en 386 y posteriores
```

Las dos instrucciones restantes SAL y SAR son muy parecidas a las instrucciones SHL y SHR, sólo que estas dos instrucciones consideran el contenido de los registros como cantidades con signo, por lo cual el bit en la posición más significativa del dato (bit de signo) se conserva sin cambio.
El siguiente ejemplo muestra el uso de las instrucciones de rotación y desplazamiento, revise el código sólo con fines ilustrativos.

```
COMMENT
*
Programa: Rota.ASM
Descripción: Este programa ilustra el uso de las instrucciones de rotación y
desplazamiento.
.MODEL TINY
.DATA
dato1     dw  10     ; variable de tipo entero
.CODE
INICIO:             ; Punto de entrada al programa
mov ax,1      ; AX=1
mov bx,10     ; BX=10
shl ax,1      ; ax=ax*2
mov cx,3      ; contador igual a 3
shl ax,cl     ; ax=ax*8
shr bx,1      ; bx=bx/2
mov cx,2      ;
shr bx,cl     ; bx=bx/4
shl dato1,1    ; dato1=dato1*2
mov ax,1      ; ax=1
rol ax,1      ; rotar ax 1 vez
mov bx,-10    ; bx=-10
sal bx,1      ; bx=bx*2
mov ax,4c00h   ; Terminar
int 21h       ; Salir al dos
END INICIO
END
```

Instrucciones para la pila

La pila es un grupo de localidades de memoria que se reservan con la finalidad de proporcionar un espacio para el almacenamiento temporal de información.

La pila de los programas es del tipo **LIFO** (Last In First Out, Ultimo en entrar, Primero en salir).

Para controlar la pila el microprocesador cuenta con dos instrucciones básicas: **Push (Meter) y Pop (sacar).**

El formato de estas instrucciones es el siguiente:

Push operando

Pop operando

Cuando se ejecuta la instrucción **Push,** el contenido del operando se almacena en la ultima posición de la pila.

Por ejemplo, si AX se carga previamente con el valor 5, una instrucción Push AX almacenaría el valor 5 en la ultima posición de la pila.

Por otro lado la instrucción **Pop** saca el último dato almacenado en la pila y lo coloca en el operando.

Siguiendo el ejemplo anterior, la instrucción Pop BX obtendría el número 5 y lo almacenaría en el registro BX.

El siguiente ejemplo muestra como implementar la instrucción XCHG por medio de las instrucciones Push y Pop. Recuerde que la instrucción XCHG intercambia el contenido de sus dos operandos.

```
.COMMENT
Programa: PushPop.ASM
Descripción: Este programa demuestra el uso de las instrucciones para el manejo de la pila, implementando la instrucción XCHG con Push y Pop
*
.MODEL tiny
.CODE
Inicio:             ;Punto de entrada al programa
Mov AX,5      ;AX=5
Mov BX,10     ;BX=10
Push AX       ;Pila=5
Mov AX,BX     ;AX=10
Pop BX        ;BX=5
Mov AX,4C00h   ;Terminar programa y salir al DOS
Int 21h       ;
END Inicio
END
```

Manipulación de cadenas (Strings)

Definición de string

En el lenguaje ensamblador el tipo de dato cadena (string) no está definido, pero para fines de programación, una cadena es definida como un conjunto de localidades de memoria consecutivas que se reservan bajo el nombre de una variable.

Almacenamiento en memoria

De acuerdo con la definición anterior, las siguientes líneas en ensamblador declaran cadenas:

```
.DATA

Cadena_ASCII   db 'Cadena',13,10,'$'

Cadena_Enteros dw 5 Dup (0)
```

Las dos líneas anteriores están declarando variables de tipo cadena. En el primer caso, Cadena_ASCII reserva un total de 9 bytes de memoria (1 byte = 1 Carácter ASCII) incluyendo el carácter '$' que indica fin de cadena. En el segundo caso, Cadena_Enteros reserva espacio para almacenar 3 cantidades enteras o lo que es lo mismo 6 bytes de memoria (1 entero = 2 bytes), todas inicializadas con cero.

La diferencia en los casos anteriores radica en el tamaño del dato que compone la cadena, 1 byte para cadenas de caracteres y 2 o más bytes para cadenas de datos numéricos.

El almacenamiento en memoria se vería de la siguiente forma:

Instrucciones para el manejo de strings
El lenguaje ensamblador cuenta con cinco instrucciones para el manejo de cadenas:

MOVS: Mueve un byte o palabra desde una localidad de memoria a otra.

LODS : Carga desde la memoria un byte en AL o una palabra en AX.

STOS : Almacena el contenido del registro AL o AX en la memoria.

CMPS : Compara localidades de memoria de un byte o palabra.

SCAS : Compara el contenido de AL o AX con el contenido de alguna localidad de memoria.

Las instrucciones para cadenas trabajan en conjunto con la instrucción CLD, la cual permite establecer que el sentido en el que las cadenas serán procesadas será de izquierda a derecha.

Otra instrucción importante es el prefijo de repetición REP, el cual permite que una instrucción para manejo de cadenas pueda ser repetida un número determinado de veces.

Los registros índice juegan un papel importante en el procesamiento de cadenas de datos, el par de registros CS:SI indican la dirección de la cadena original que será procesada, y el par ES:DI contienen la dirección donde las cadenas pueden ser almacenadas.

Para comprender realmente como funcionan las instrucciones para cadenas analizaremos varios programas que fueron escritos para este fin. Recuerde que las cadenas en ensamblador no se refieren únicamente a cadenas de caracteres ASCII, sino a cualquier tipo de dato.

```
.COMMENT
Programa: Cad1.ASM
Descripción: Este programa ilustra la forma de utilizar la instrucción MOVS para copiar el
contenido de una cadena dentro de otra.
.MODEL tiny
.DATA
cad1    db     'Esta es la cadena1','$'
cad2    db     'Esta es la cadena2','$'
```

```
.CODE
inicio:           ;Punto de entrada al programa
cld       ;Procesamiento de cadenas de izq->der.
mov cx,18   ;longitud de la cadena original
lea di,cad2  ;ES:DI contienen la dirección de Cad2
lea si,cad1  ;DS:SI contienen la dirección de Cad1
rep movsb   ;DS:SI->ES:DI, SI=SI+1, DI=DI+1
lea dx,cad1  ;Imprimir Cad1 en pantalla
mov ah,09h   ;
int 21h     ;
lea dx,cad2  ;Imprimir Cad2 en pantalla
mov ah,09h   ;
int 21h     ;
mov ax,4c00h ;Terminal programa y regresar al DOS
int 21h     ;
END inicio
END
.COMMENT
Programa: Cad2.ASM
Descripción: Este programa demuestra la diferencia entre el uso de MOVSB y MOVSW.
El programa debe copiar Cad3 dentro de Cad1 usando 18 repeticiones con MOVSB,
después realiza lo mismo con Cad4 y Cad2 pero usando solo nueve repeticiones de la
instrucción MOVSW.
El resultado es el mismo en ambos casos
.MODEL tiny
.DATA
cad1    db     'Cadena de prueba1 ','$'
cad2    db     'Cadena de prueba2 ','$'
cad3    db     18 dup (' ')
cad4    db     18 dup (' ')
.CODE
inicio:           ;Punto de entrada al programa
cld       ;procesamiento de izq->der.
mov cx,18   ;Longitud de la cadena
lea si,cad3  ;DS:SI->Cad3
lea di,cad1  ;ES:DI->Cad1
rep movsb   ;Cad3->Cad1
mov cx,9    ;Longitud de la cadena por pares de bytes
lea si,cad4  ;DS:SI->Cad4
lea di,cad2  ;ES:DI->Cad2
rep movsw   ;Cad4->Cad2
lea dx,cad1  ;
mov ah,09h   ;Imprimir Cad1
int 21h     ;
lea dx,cad2  ;
mov ah,09h   ;Imprimir Cad2
int 21h     ;
mov ax,4c00h ;Terminar programa y regresar al DOS
int 21h     ;
END inicio
END


.COMMENT
```

```
Programa: Cad3.ASM
Descripción: Este programa muestra el uso de la instrucción LODSB.
El programa invierte el orden de los elementos de una cadena y los almacena en otra
cadena que originalmente esta inicializada con espacios. Al final se imprimen las dos
cadenas.
.MODEL tiny
.DATA
cad1    db     'Cadena de prueba','$'
cad2    db     16 dup (' '),'$'
.CODE
inicio:                ;Punto de entrada al programa
cld            ;Procesamiento de izq->der.
  mov cx,16       ;Longitud de la cadena
lea si,cad1     ;DS:SI->Cad1
lea di,cad2+15   ;ES:DI apuntan al final del área reservada para
otro:                  ;almacenar la cadena invertida
lodsb          ;Obtener el primer carácter de Cad1
mov [di],al     ;almacenarlo en la posición actual de DI
dec di          ;Disminuir DI
loop otro       ;Obtener siguiente carácter de Cad1
lea dx,cad1     ;
mov ah,09h      ;Imprimir cadena original
int 21h         ;
lea dx,cad2     ;
mov ah,09h      ;Imprimir cadena invertida
int 21h         ;
mov ax,4c00h     ;Terminar programa y regresar al DOS
int 21h         ;
END inicio
END

COMMENT
Programa: Cad4.ASM
Descripción: Este programa utiliza la instrucción STOSB para rellenar un  rea de memoria
con el contenido del registro AL.
En este caso, el área de memoria reservado para  la variable Cad1 es rellenada con el
carácter
ASCII '*'.
.MODEL tiny
.DATA
cad1    db     'Cadena de prueba',13,10,'$'
CODE
inicio:
lea dx,cad1   ;Imprimir Cad1 antes de que sea borrada
mov ah,09h    ;
int 21h       ;
cld           ;Procesamiento de izq->der
mov al,'*'    ;Inicializar AL con '*'
mov cx,16     ;Longitud de la cadena que se va a rellenar
lea di,cad1   ;ES:DI->Cad1
rep stosb     ;Rellenar 16 bytes de memoria con '*'
lea dx,cad1   ;
mov ah,09h    ;Imprimir Cad1 después de ser borrada
int 21h       ;
```

```
mov ax,4c00h  ;Terminar programa y regresar al DOS
int 21h      ;
END inicio
END
```

PROGRAMACIÓN DE E/S

Definición de interrupción

Una interrupción es un estado en el cual el microprocesador detiene la ejecución de un programa para atender una petición especial solicitada por el propio programa o por un dispositivo físico conectado al microprocesador externamente.

Las interrupciones fueron creadas para facilitar al programador el acceso a los diferentes dispositivos de la computadora (puertos de comunicaciones, terminales, impresoras, etc.).

Ejecución de una interrupción

Cuando durante la ejecución de un programa se produce una interrupción, el microprocesador realiza los siguientes pasos:

1.- Detiene la ejecución del programa

2.- Almacena los registros CS, IP y Banderas en la pila

3.- Modifica el CS y el IP para que apunten a la dirección donde se encuentra la rutina de interrupción.

4.- Ejecuta las instrucciones de la rutina de interrupción.

5.- Restablece usando la instrucción RETI los valores originales de los registros CS, IP y Banderas.

6.- Continua con la ejecución del programa en el punto donde fue interrumpido.

Las rutinas se almacenan en la memoria de la computadora cada vez que ésta es inicializada, a esto se le conoce como vector de interrupciones.

Tipos de interrupciones

El microprocesador puede atender dos tipos de interrupciones: interrupciones por software e interrupciones por hardware.

Las interrupciones por software son llamadas desde los programas y son proporcionadas por el sistema operativo (MS-DOS). Existen dos tipos de estas: las interrupciones del DOS y las interrupciones del BIOS (Basic Input Output System o Sistema Básico de Entrada/Salida). Estas interrupciones son invocadas con la instrucción INT del ensamblador.

Por otro lado, **las interrupciones por Hardware** son proporcionadas por el propio microprocesador y también existen dos tipos: interrupciones por hardware internas y las interrupciones por hardware externas. Las interrupciones internas son invocadas por el microprocesador cuando se produce alguna operación incorrecta, como por ejemplo, un intento de dividir por cero o una transferencia de datos entre registros de diferentes longitudes.

Las interrupciones externas son provocadas por los dispositivos periféricos conectados al microprocesador. Para lograr esto, a cada dispositivo periférico se le asigna

una línea física de interrupción que lo comunica con el microprocesador por medio de un circuito integrado auxiliar, el cual se conoce como controlador programable de interrupciones (PIC).

Las computadoras basadas en el microprocesador 8086/8088 cuentan solamente con un PIC, con lo cual pueden proporcionar hasta 8 líneas de interrupción (IRQ), las cuales son llamadas IRQ0 a IRQ7, por otro lado, las computadoras basadas en el microprocesador 80286 y posteriores cuentan con dos chips controladores, con los cuales pueden proporcionar hasta un máximo de 16 líneas IRQ, las cuales son llamadas IRQ0 a IRQ15.

La siguiente es una lista de las interrupciones por software disponibles por el sistema operativo.

Interrupciones del BIOS

Manejo de dispositivos periféricos

- INT 10H Manejo de la pantalla.
- INT 13H Manejo de unidades de disco.
- INT 14H Manejo de los puertos de comunicaciones(RS232).
- INT 15H Manejo de cinta magnética.
- INT 16H Manejo del teclado.
- INT 17H Manejo de la impresora.

Manejo del estado del equipo

- INT 11H Servicios de la lista de elementos de la computadora.
- INT 12H Servicios para el cálculo del tamaño de la memoria.

Servicios de fecha y hora

- INT 1AH Manejo del reloj.

Impresión de pantalla

- INT 5H Impresión de la información contenida en la pantalla.

Servicios especiales

- INT 18H Activación del lenguaje Interprete Basic de la ROM.
- INT 19H Activación de la rutina de arranque de la computadora.

Interrupciones del DOS

- INT 20H Termina la ejecución de un programa.
- INT 22H Dirección de terminación. Guarda la dirección donde se transfiere el control cuando termina la ejecución del programa.
- INT 23H Dirección de la interrupción que se ejecuta cuando se presiona Ctrl-Break.
- INT 24H Manejo de errores críticos.
- INT 25H Lectura directa de sectores del disco.
- INT 26H Escritura directa de sectores del disco.
- INT 27H Terminar un programa y devolver el control al DOS sin borrar el programa de la memoria.
- INT 21H Esta interrupción proporciona una gran cantidad de funciones, las cuales deben ser invocadas en conjunto con el registro AH.

1	Terminación de un programa.
2	Entrada de carácter con eco.
3	Salida a pantalla.
4	Entrada por el puerto serie.
5	Salida por el puerto serie.
6	Salida a la impresora.
7	E/S directa por pantalla.
8	Entrada directa de carácter sin eco.

9	Entrada de carácter sin eco.
10	Visualizar cadenas de caracteres.
11	Entrada desde el teclado.
12	Comprobación del estado de entrada.
13	Borrar registro de entrada.
14	Inicializar unidad de disco.

A continuación se mostrarán algunos programas que utilizan llamadas a diferentes interrupciones por software tanto del BIOS como del DOS.

El siguiente programa utiliza la función 09h de la interrupción 21 del DOS para mostrar en la pantalla un mensaje.

```
.COMMENT
*
Programa: Int1.ASM
Descripción: Imprime una cadena de caracteres en la pantalla por medio de la función 09h
de la interrupción 21h del DOS.
*
.MODEL tiny
.DATA
Mensaje    db   'Interrupciones 21h del DOS',13,10,'$'
.CODE
Inicio:
Lea DX,Mensaje
Mov Ah,09h
Int 21h
Mov ax,4C00h
Int 21h
END Inicio
END
```

El siguiente programa exhibe dos cadenas de caracteres en la pantalla, pero a diferencia del anterior éste no regresa al DOS inmediatamente, espera a que cualquier tecla sea presionada y entonces termina, para ello se utiliza la función 10h de la interrupción 16h del BIOS.

```
.COMMENT
*
Programa: Int2.ASM
Descripción: Imprime dos cadenas de caracteres en la pantalla por medio de la función
09h de la interrupción 21h del DOS y después espera a que una tecla sea presionada, esto
por medio de la interrupción 16h del BIOS con la función 10h.
*
.MODEL tiny
.DATA
Mensaje    db   'Mas interrupciones',13,10,'$'
Mensaje2   db   'Presione cualquier tecla...',13,10,'$'
.CODE
Inicio:
Lea DX,Mensaje
Mov Ah,09h
Int 21h
```

```
Lea DX,Mensaje2
Mov Ah,09h
Int 21h
Mov Ah,10h
Int 16h
Mov ax,4C00h
Int 21h
END Inicio
END
```

Macros

Una de las principales desventajas de la programación en lenguaje ensamblador es la repetición constante de ciertos grupos de instrucciones. Por ejemplo el siguiente conjunto de instrucciones nos permite imprimir una variable de tipo cadena en la pantalla:

```
Lea DX,Cadena ;Direccionar la cadena
Mov AH,09h        ;Usar la función 09h para imprimir cadenas
Int 21h           ;llamada a la interrupción 21h del DOS
```

Si necesitamos que en nuestro programa se muestren mensajes constantemente, es obvio que debemos duplicar este conjunto de instrucciones por cada mensaje que se desea enviar a pantalla.

El principal problema que esto nos ocasiona es que el tamaño de nuestro programa crece considerablemente, y mientras más grande sea el programa, más difícil será encontrar la causa de algún error cuando éste ocurra.

La mejor solución en estos casos es el uso de las MACROS. Una macro es un conjunto de instrucciones que se agrupan bajo un nombre descriptivo (macroinstrucción) y que sólo es necesario declarar una vez (macrodefinición).

Una vez que la macro ha sido declarada, sólo es necesario indicar su nombre en el cuerpo del programa y el ensamblador se encargara de reemplazar la macroinstrucción por las instrucciones de la macro (expansión de la macro).

El formato general de una macro es el siguiente:

```
.MACRO  Nombre [(parametro1, parametro2, etc)]
INSTRUCCIONES
ENDM
```

Nuevamente, lo que se encuentra entre paréntesis cuadrados es opcional.
De acuerdo con esto, la macro para imprimir cadenas quedaría de la siguiente forma:

```
.MACRO Imprime_Cad(Cadena)
Lea DX,Cadena
Mov Ah,09h
Int 21h
ENDM
```

Parámetros y etiquetas

Dentro de las propiedades más importantes de las macros se deben destacar la posibilidad de utilizar parámetros y etiquetas.

Los parámetros permiten que una misma macro pueda ser usada bajo diferentes condiciones, por ejemplo, se puede crear una macro para posicionar el cursor en

diferentes coordenadas de la pantalla e indicar sus coordenadas por medio de parámetros.

La siguiente macro nos muestra esta propiedad:

```
;Esta macro posiciona el cursor en las coordenadas que se le indican como

;parámetros. Es el equivalente al GotoXY de Pascal.

.MACRO gotoxy (x,y)

xor bh,bh              ;Seleccionar página cero de video

mov dl,x               ;Columna

mov dh,y               ;Renglón

mov ah,02h     ;Función 02h para posicionar cursor

int 10h        ;llamada a la int 10h del BIOS

ENDM
```

También existen situaciones en las que los parámetros no son necesarios, es por esta razón que los parámetros son opcionales en la declaración de la macro.

```
;Esta macro realiza una pausa en el programa hasta que una tecla es ;presionada. Es el equivalente del readkey en Pascal.

.MACRO tecla

mov ah,10h

int 16h

ENDM
```

Por otro lado, las etiquetas también son útiles dentro de las macros. Suponga que se desea crear una macro que imprima una cadena un numero n de veces, esta macro podría ser declarada de la siguiente forma:

```
.MACRO Imprime_nCad (Cadena, Cuantos)

Mov CX,Cuantos         ;Iniciar Contador

Lea DX,Cadena ;Direccionar la cadena que se va a imprimir

Mov Ah,09h             ;Usar la función 09h

Otra:                          ;Etiqueta interna

Int 21h        ;Imprimir la Cadena n veces

Loop Otra              ;Siguiente Impresión

ENDM
```

Ensamble de macros

Como ya se mencionó antes, una macro es declarada una sola vez y puede ser llamada cuantas veces sea necesario dentro del cuerpo del programa.

Cada vez que el ensamblador encuentra una macroinstrucción, verifica si ésta fue declarada; si esta verificación es exitosa, el ensamblador toma las instrucciones del cuerpo de la macro y las reemplaza en el lugar donde la macro fue llamada.

El siguiente programa muestra la declaración y uso de las macros:

```
.COMMENT

Programa: Macros1.ASM

Descripción: Este programa muestra el uso de macros.

.MODEL TINY

; Declaración de variables

.DATA

cad         db     'Ejemplo del uso de macros...',13,10,'$'

cad1    db     'Presiona una tecla...','$'

cad2    db     'Ejemplo del uso de la macro gotoxy...','$'

;Aquí se declaran las macros.

;Esta macro imprime una cadena pasada como parámetro.

;Utiliza la función 09h de la Int 21h del DOS.

.MACRO imprime_cad(cadena)

lea dx,cadena

mov ah,09h

int 21h

ENDM

;Esta macro realiza una pausa en el programa hasta que una tecla se ;presione. Es el equivalente del readkey en Pascal.

.MACRO tecla

mov ah,10h

int 16h

ENDM

;Esta macro posiciona el cursor en las coordenadas que se le indican como

;parámetros. Es el equivalente al GotoXY de Pascal.
```

```
.MACRO gotoxy (x,y)
xor bh,bh
mov dl,x
mov dh,y
mov ah,02h
int 10h
ENDM
;Esta macro limpia la pantalla.
;Utiliza la función 06h de la Int 10h del Bios.
.MACRO limpiar_pantalla
mov ax,0600h
mov bh,17h
mov cx,0000h
mov dx,184fh
int 10h
ENDM
;Aquí comienza el cuerpo del programa principal
.CODE
inicio:                     ;Declaración del punto de entrada
limpiar_pantalla    ;Llamada a la macro
gotoxy (0,0)        ;Colocar el cursor en 0,0
imprime_cad(cad)    ;Imprime el primer mensaje
imprime_cad(cad1)   ;Imprime el segundo mensaje
tecla               ;Espera a que se presione una tecla
gotoxy (30,12)      ;Colocar el cursor en 30,12
imprime_cad(cad2)   ;Imprimir el tercer mensaje
gotoxy (50,24)      ;Colocar el cursor en 50,24
imprime_cad(cad1)   ;Imprimir el segundo mensaje
tecla               ;Esperar por una tecla
mov ax,4c00h        ;Fin del programa y regresar al DOS.
```

int 21h

END inicio

END

Ventajas y desventajas

Si bien es cierto que las macros proporcionan mayor flexibilidad a la hora de programar, también es cierto que tienen algunas desventajas.

La siguiente es una lista de la principales ventajas y desventajas del uso de las macros.

Ventajas:

- Menor posibilidad de cometer errores por repetición.
- Mayor flexibilidad en la programación al permitir el uso de parámetros.
- Código fuente más compacto.
- Al ser más pequeño el código fuente, también es más fácil de leer por otros.

Desventajas:

- El código ejecutable se vuelve más grande con cada llamada a la macro.
- Las macros deben ser bien planeadas para evitar la redundancia de código.

PROGRAMACIÓN MODULAR

Definición de procedimientos

Un procedimiento es un conjunto de instrucciones que tienen la finalidad de ejecutar una tarea especifica dentro de un programa. Los procedimientos son muy similares a las macros.

Un procedimiento se declara una sola vez en el código fuente y cuando el programa se ensambla y ejecuta, el procedimiento se coloca en memoria para que pueda ser utilizado por el programa.

Las principales ventajas en el uso de procedimientos son: permiten una codificación más limpia y compacta, es decir el código fuente es más pequeño; también permiten el ahorro de memoria, esto es porque un mismo procedimiento puede ser llamado varias veces en el mismo programa y sólo requiere memoria una vez.

Los procedimientos tienen la desventaja de que reducen la velocidad de ejecución de los programas, esto se debe a la forma en que los procedimientos se ejecutan. A continuación se presentan los pasos necesarios para ejecutar un procedimiento:

1.- Se encuentra la llamada Call

2.- El microprocesador almacena en la Pila el contenido del IP

3.- Se coloca en el IP el valor del desplazamiento correspondiente al Procedimiento

4.- El microprocesador ejecuta las instrucciones del procedimiento

5.- El procedimiento termina cuando se encuentra la instrucción Ret

6.- Se saca de la pila el valor original del IP y se continua el flujo del programa

Un procedimiento se declara de la siguiente forma:

PROC nombre

instrucción

instrucción ....

RET

ENDP NOMBRE

En donde PROC es una palabra reservada que indica el inicio de un procedimiento, RET es una instrucción que indica la terminación del conjunto de instrucciones de un procedimiento y finalmente ENDP es la palabra reservada para fin de procedimiento.

Paso de parámetros

Los procedimientos en lenguaje ensamblador no cuentan con un mecanismo para el paso de parámetros; por lo cual, la única forma de lograr esto es colocando los parámetros que nos interesan en los registros de propósito general antes de que el procedimiento sea ejecutado.

El siguiente procedimiento coloca el cursor en las coordenadas establecidas en Dl y Dh.

```
Proc GotoXY

xor bh,bh

mov ah,02h

int 10h

Ret

Endp GotoXY
```

En este ejemplo, las coordenadas XY se deben situar en el registro DX antes de que se llame al procedimiento.

Llamada a procedimientos

Los procedimientos son llamados por los programas por medio de la instrucción CALL, seguida del nombre del procedimiento.

Ejemplo:

```
Call GotoXY
```

El siguiente programa muestra la forma de pasarle parámetros a los procedimientos por medio de los registros generales. Este programa declara tres procedimientos:

GotoXY: Coloca el cursor en las coordenadas especificadas

Limpia_Pantalla: Limpia la pantalla

Imprime_Cad: Imprime una cadena en la posición actual del cursor

```
.COMMENT
*
Programa: Proc2.ASM
Descripción: Este programa ilustra la forma de utilizar procedimientos en los programas
por medio de la instrucción Call y la forma de pasarles parámetros.
.MODEL TINY
.DATA
```

```
Cad1 db 'Esta es una cadena de prueba...',13,10,'$'
.CODE
INICIO:                ;Punto de entrada al programa
Mov DL,20          ;X=20
Mov DH,10          ;Y=10
Call Gotoxy        ;GotoXY 20,10
Lea DX,cad1        ;DX->Cad1
Call Imprime_Cad    ;Imprimir Cad1
Mov Ax,04C00h      ;Terminar y regresar al dos
Int 21h            ;
END INICIO
;*********************************************************************
;Procedimiento: GotoXY
;Descripción: Coloca el cursor una posición especifica de la pantalla
;Parámetros: Dl=X, Dh=Y
;*********************************************************************
PROC GotoXY
Xor Bh,Bh
Mov Ah,02h
Int 10h
Ret
ENDP GotoXY
;*********************************************************************
;Procedimiento: Limpia_Pantalla
;Descripción: Imprime una cadena de caracteres en la posición del cursor
;Parámetros: La dirección de la cadena en DX
;*********************************************************************
PROC Imprime_Cad
Mov Ah,09h
Int 21h
Ret
ENDP Imprime_Cad
END
```

Procedimientos internos

Los procedimientos internos son aquellos que se declaran y se llaman dentro del mismo programa, también son llamados procedimientos locales.
El listado anterior muestra la forma de utilizar procedimientos internos.

Procedimientos externos

Los procedimientos externos, a diferencia de los internos, se declaran en módulos o programas separados al programa donde el procedimiento es llamado, en otras palabras, la llamada al procedimiento se encuentra en un programa y el procedimiento en otro.

Para poder utilizar procedimientos externos, es necesario que sean declarados como públicos en el programa donde se encuentran y que sean llamados como externos en el programa donde serán usados. Para lograr esto, Pass32 cuenta con tres directivas de ensamble: .PUBLIC para declarar los procedimientos como públicos, .EXTERN para indicar que el procedimiento que se va a usar está fuera del programa y .INCLUDE para enlazar el programa que contiene los procedimientos con el programa que los llama.

El siguiente programa muestra el uso de las directivas de inclusión. Primeramente, el archivo Proc2.ASM se modificó para que su variable Cad1 fuera declarada como publica,

el programa Proc3.ASM contiene la línea .INCLUDE Proc2.ASM, lo cual indica al ensamblador que, en caso de que se soliciten datos, etiquetas o procedimientos externos, éstos se busquen en el archivo incluido.

Pass32 proporciona grandes facilidades para el manejo de procedimientos; en este caso, solamente Cad1 debe ser declarada como pública, puesto que los procedimientos se buscan y anexan automáticamente al programa que los llama si es que existen.

```
.COMMENT
*
Programa: Proc3.ASM
Descripción: Este programa ilustra la forma de utilizar procedimientos y datos externos
en los programas por medio de las directivas de inclusión include y public.
.MODEL TINY
.INCLUDE proc2.ASM          ;Incluir el archivo proc2.asm
;el cual contiene la variable de cadena
;Cad1 y los procedimientos externos
;usados en este programa.
.DATA
Cad2 db 'Esta es una cadena de prueba 2...',13,10,'$'
.CODE
INICIO:                     ;Punto de entrada al programa
Mov Dl,20          ;X=20
Mov Dh,10          ;Y=10
Call GotoXY        ;GotoXY 20,10
Lea DX,Cad2        ;DX->Cad2 en Proc3.asm
Call Imprime_Cad   ;Imprime Cad2
Lea DX,Cad1        ;DX->Cad1 en Proc2.asm
Call Imprime_Cad   ;Imprime Cad1
Mov AX,04C00h      ;Fin del programa
Int 21h            ;
END INICIO
END
```

Con estas capacidades, es fácil crear bibliotecas de procedimientos y macros que puedan ser utilizados constantemente por los demás programas, ahorrando con ello tiempo de programación al reutilizar código fuente.

El siguiente programa muestra la forma de escribir una biblioteca de procedimientos y la forma de utilizarlos en los programas.

```
.COMMENT
*
Programa: Proc3.ASM
Descripción: Este programa ilustra la forma de utilizar procedimientos y datos externos
en los programas por medio de las directivas de inclusión include y public.
.MODEL TINY
.INCLUDE proclib.inc       ;Incluir el archivo proclib.inc
;el cual contiene la variable de cadena
;Cad1 y los procedimientos externos
;usados en este programa.
.DATA
Cad1 db 'Esta es una cadena de prueba 2...',13,10,'$'
Cad2 db 'Presiona una tecla...','$'
.CODE
INICIO:                    ;Punto de entrada al programa
```

```
Call limpia_Pantalla ;
Mov Dl,20          ;X=20
Mov Dh,10          ;Y=10
Call GotoXY        ;GotoXY 20,10
Lea DX,Cad1        ;DX->Cad1
Call Imprime_Cad   ;Imprime Cad1
Mov Dl,40          ;
Mov Dh,24          ;
Call GotoXY        ;GotoXY 40,25
Lea DX,Cad2        ;
Call Imprime_Cad   ;Imprime Cad2
Call Espera_Tecla  ;Esperar por una tecla presionada
Mov AX,04C00h      ;Fin del programa
Int 21h            ;
END INICIO
END
.COMMENT
Biblioteca de Procedimientos en Lenguaje ensamblador
.CODE
;************************************************************************
;Procedimiento: GotoXY
;  Descripción: Coloca el cursor una posición especifica de la pantalla
;  Parámetros: Dl=X, Dh=Y
;************************************************************************
PROC GotoXY
Xor Bh,Bh
Mov Ah,02h
Int 10h
Ret
ENDP GotoXY
;************************************************************************
;Procedimiento: Imprime_Cad
;  Descripción: Imprime una cadena de caracteres en la posición del cursor
;   Parámetros: La dirección de la cadena en DX
;************************************************************************
PROC Imprime_Cad
Int 21h
Ret
ENDP Imprime_Cad
;************************************************************************
;Procedimiento: Limpia_Pantalla
;  Descripción: Limpia la pantalla de la computadora y coloca el cursor
;  en 0,0.
;   Parámetros: Ninguno
;************************************************************************
PROC Limpia_Pantalla
mov ax,0600h
mov bh,17h
mov cx,0000h
mov dx,184fh
int 10h
Mov dx,0000h
Call Gotoxy
Ret
```

```
ENDP Limpia_Pantalla
;************************************************************************
;Procedimiento: Espera_Tecla
;  Descripción: Detiene la ejecución de un programa hasta que se presiona
; una tecla
;   Parámetros: Ninguno
;************************************************************************
PROC Espera_Tecla
mov ah,10h
int 16h
Ret
ENDP Espera_Tecla
```

PROGRAMACIÓN HÍBRIDA

Pascal y ensamblador

Como ya se mencionó, la programación en lenguaje ensamblador proporciona un mayor control sobre el hardware de la computadora, pero también dificulta la buena estructuración de los programas.

La programación híbrida proporciona un mecanismo por medio del cual podemos aprovechar las ventajas del lenguaje ensamblador y los lenguajes de alto nivel, todo esto con el fin escribir programas más rápidos y eficientes.

En esta sección se mostrará la forma para crear programas híbridos utilizando el lenguaje ensamblador y Turbo Pascal.

Turbo Pascal permite escribir procedimientos y funciones en código ensamblador e incluirlas como parte de los programas en lenguaje Pascal; para esto, Turbo Pascal cuenta con dos palabras reservadas: Assembler y Asm.

Assembler permite indicarle a Turbo Pascal que la rutina o procedimiento que se está escribiendo está totalmente escrita en código ensamblador.

Ejemplo de un procedimiento híbrido:

```
Procedure Limpia_Pantalla;

Assembler;

Asm

Mov AX,0600h

Mov BH,18h

Mov CX,0000h

Mov DX,184Fh

Int 10h

End;
```

El procedimiento del listado 23 utiliza la función 06h de la Int 10h del BIOS para limpiar la pantalla, este procedimiento es análogo al procedimiento ClrScr de la unidad CRT de Turbo Pascal.

Por otro lado, Asm nos permite incluir bloques de instrucciones en lenguaje ensamblador en cualquier parte del programa sin necesidad de escribir procedimientos completos en ensamblador.

Ejemplo de un programa con un bloque de instrucciones en ensamblador:

{ Este programa muestra como se construye un programa híbrido utilizando un bloque Asm... End; en Turbo Pascal.
El programa solicita que se introduzcan dos número, después calcula la suma por medio de la instrucción Add de ensamblador y finalmente imprime el resultado en la pantalla.}

```
Program hibrido;
Uses Crt;
Var
N1,N2,Res : integer;
Begin
Writeln("Introduce un número: ");
Readln(N1);
Writeln("Introduce un número: ");
Readln(N2);
Asm
Mov AX,N1;
Add AX,N2;
Mov Res,AX
End;
Writeln("El resultado de la suma es: ",Res);
Readln;
End.
```

El programa del listado 24 realiza la suma de dos cantidades enteras (N1 y N2) introducidas previamente por el usuario, después almacena el resultado en la variable Res y finalmente presenta el resultado en la pantalla.

El lenguaje ensamblador no cuenta con funciones de entrada y salida formateada, por lo cual es muy complicado escribir programas que sean interactivos, es decir, programas que soliciten información o datos al usuario. Es aquí donde podemos explotar la facilidad de la programación híbrida, en el programa anterior se utilizan las funciones Readln y Writeln para obtener y presentar información al usuario y dejamos los cálculos para las rutinas en ensamblador.

En el siguiente listado nos muestra la forma de escribir programas completos utilizando procedimientos híbridos.

{Este programa solicita al usuario que presione alguna tecla, cuando la tecla es presionada, ésta se utiliza para rellenar la pantalla.
El programa termina cuando se presiona la tecla enter.
El programa utiliza tres procedimientos:
Limpia_Pantalla: Este se encarga de borrar la pantalla

Cursor_XY: Este procedimiento reemplaza al GotoXY de Pascal

Imprime_Car: Este procedimiento imprime en pantalla el carácter que se le pasa como parámetro. }

```
Program Hibrido2;
Uses Crt;
Var
```

```
Car: Char;
i,j : integer;
{Este procedimiento limpia la pantalla y pone blanco sobre azul}
Procedure Limpia_Pantalla;
Assembler;
Asm
Mov AX,0600h
Mov Bh,17h
Mov CX,0000h
Mov DX,184Fh
Int 10h
End;
{Este procedimiento imprime el carácter en la pantalla}
Procedure Imprime_Car(C: Char);
Assembler;
Asm
Mov Ah,02h
Mov Dl,C
Int 21h
End;
{Este procedimiento tiene la misma función que el procedimiento GotoXY de Turbo Pascal}
Procedure Cursor_XY(X,Y: Byte);
Assembler;
Asm
Mov Ah,02h
Mov Bh,00h
Mov Dh,Y
Mov Dl,X
Int 10h
End;
Begin
Limpia_Pantalla;
Repeat
Limpia_Pantalla;
Cursor_XY(0,0);
Write('Introduce un carácter: ');
Car:=ReadKey;
Imprime_Car(Car);
Limpia_Pantalla;
If car <> #13 then
Begin
For i:=0 to 24 do
For j:=0 to 79 do
Begin
Cursor_XY(j,i);
Imprime_Car(Car);
End;
Cursor_XY(30,24);
Write('Presiona enter para salir u otro para seguir...');
Readln;
Until car = #13;
End.
```

CONCLUSIÓN

Quizá el lenguaje ensamblador es el lenguaje de programación mas difícil de comprender, pero en la actualidad es una de las herramientas de programación más utilizadas para obtener un mayor conocimiento acerca del funcionamiento de una computadora personal y programar determinados dispositivos.

Las personas que deseen hacer uso de este lenguaje deberán tener un nivel de comprensión alto ya que como se menciono este lenguaje no es tan fácil, por lo que deberán poner gran empeño para el aprendizaje.

Existe en la actualidad una gran cantidad de programas ensambladores que nos permiten programar en ambientes operativos gráficos como Windows 95/98, Windows NT y Linux.

Aunque algunas personas piensan que el uso de este lenguaje esta ya obsoleto están muy equivocados, hoy en dia existen en internet infinidad de paginas que presentan información acerca de este tema (como es nuestro caso) y que se actualizan diariamente, no por algo existen estas paginas, es decir , si encontramos tanta cantidad de información es por que todavía esta en uso.

BIBLIOGRAFÍA

Abel, P.; Lenguaje Ensamblador para IBM PC y Compatibles; Ed. Prentice Hall; 3ª Edición; 1996.

Brey, B.; Los microprocesadores de Intel: Arquitectura, Programación e Interfaces; Ed. Prentice Hall; 3ª Edición; 1995.

Caballar, J.; El libro de las comunicaciones del PC: técnica, programación y aplicaciones; Ed. Rama-Computec; 1ª Edición; 1997.

Morgan y Waite; Introducción al microprocesador 8086/8088; Ed. Byte Books/Mc Graw Hill; 1ª Edición; 1992.

Pawelczak; Pass32 32 bit Assembler V 2.5 Instruction Manual; 1997.

Rojas, A.; Ensamblador Básico; Ed. Computec; 2ª Edición; 1995.

Socha y Norton; Assembly Language for the PC; Ed. Brady Publishing; 3ª Edición; 1992.

Tannenbaum, A.; Organización de Computadoras un enfoque estructurado; Ed. Prentice Hall; 3ª Edición; 1992.

http://www.cryogen.com/Nasm

http://www.geocities.com/SiliconValley/Bay/3437/index.html