Unidad I - Microprocesadores (Intel 8086).

1.1 Introducción a los Microprocesadores

1.1.1 Definición de microprocesador

Microprocesador es un circuito integrado que incluye todas o casi todas las funciones de una unidad central de proceso (Central Procesing Unit – CPU). Incluyen la unidad de control (CU), la unidad lógica aritmética (ALU), y los registros. La interfaz con el exterior (subsistema de memoria y subsistema de entrada/salida) se realiza por medio de los buses de datos, direcciones y control.

Históricamente, los primeros microprocesadores aparecieron a principios de los 1970 para ser usados en calculadoras. Solo trabajaban con 4 bits. En la actualidad se encuentran disponibles en diferentes tamaños y velocidades y han convertido en obsoletas todas demás formas de procesamiento.

1.1.2 Diferencia entre microprocesador y microcontrolador

Un microcontrolador (uC) es un circuito integrado que incluye todas las partes de una computadora pequeña, y no solo las partes del CPU, como los microprocesadores. Suelen basarse en un microprocesador, que implementa el CPU y le añaden Memoria volátil, no volátil y periféricos de Entrada y salida. Sin embargo, en general, los microprocesadores (uP) incluyen una mayor capacidad de cálculo y de direccionamiento a memoria.

1.1.3 Tipos de Microprocesadores

Los procesadores pueden clasificarse, entre otras formas, por el tamaño de sus bus de datos y por su arquitectura

Los procesadores pueden ser de 4, 8 , 16, 32 o 64 bits, lo que corresponde con el ancho de sus bus de datos o con el del ALU y los registros. En general, este tamaño coincide con el tamaño de los enteros más grandes con los que puede trabajar el microprocesador en una sola instrucción.

Por el tipo de arquitectura de las CPU, los microprocesadores pueden clasificarse como de Conjunto Complejo de instrucciones (CISC) o de Conjunto de instrucciones reducido (RISC)

Las computadoras de tipo CISC (Complex Instruction Set Computers), tienen un número amplio de instrucciones y modos de direccionamiento. Se implementan instrucciones especiales que realizan funciones complejas. El número de registros del CPU es limitado. Los microprocesadores anteriores a los años 1980 y sus derivados más modernos corresponden a esta categoría.

En las de tipo RISC (Reduced Instruction Set Computers), solo se cuenta con unas pocas instrucciones y modos de direccionamiento, pero se busca implementarlos de forma muy eficiente y que todas las instrucciones trabajen con todos los modos de direccionamiento. Amplio número de registros en el CPU.

Este tipo de CPUs surgió a partir de estudios realizados a mediados de los años ochenta en los que se comprobó que la mayoría de los programas escritos en lenguajes de alto nivel, al ser compilados, solo utilizaban unas pocas instrucciones simples y no las instrucciones más complejas, por lo que los recursos usados en la implementación de estas instrucciones complejas y especializadas no contribuían significativamente a mejorar la velocidad de ejecución de los programas escritos en lenguajes de alto nivel. Así mismo, se observo que una parte importante de una parte importante del tiempo de ejecución se empleaba en pasar los parámetros de las funciones por medio de la pila.

Por lo anterior, entre otras propuestas, se sugirió dedicar una mayor cantidad de recursos a únicamente implementar las instrucciones más utilizadas por los compiladores de la manera más eficiente posible, en lugar de implementar muchas instrucciones que no serian utilizadas. Además, con el objeto de reducir el tiempo consumido por paso de los parámetros de las funciones, se sugiere el pasar los parámetros por medio de un muy grande conjunto de registros en lugar de hacerlo por medio de la pila.

1.2 .- Arquitectura del procesador.

El 8086 fue el segundo procesador de 16 bits en aparecer en el mercado (en 1978) y el primero en lograr un éxito comercial. Intel lo comercializo en dos versiones: el 8086 y el 8088, con las siguientes características.

	8086	8088
Bus de direcciones	20 bits	20 Bits
Bus de datos interno	16 Bits	16 Bits
Bus de datos externo	16 Bits	8 Bits

La razón del bus de datos externo reducido del 8088 fue hacerlo compatible con los periféricos de los procesadores de 8 bits muy populares en esos días, lo que reduciría el costo de un sistema completo.

1.2.1 A través de diagrama a bloques(Estructura Interna)

Internamente, el 8086 se divide en dos unidades funcionales independientes; la Unidad de Interfase con el Bus (BIU, Bus Interface Unit) y la Unidad de Ejecución (EU, Execution Unit) como se muestra en la fig. 1.1.

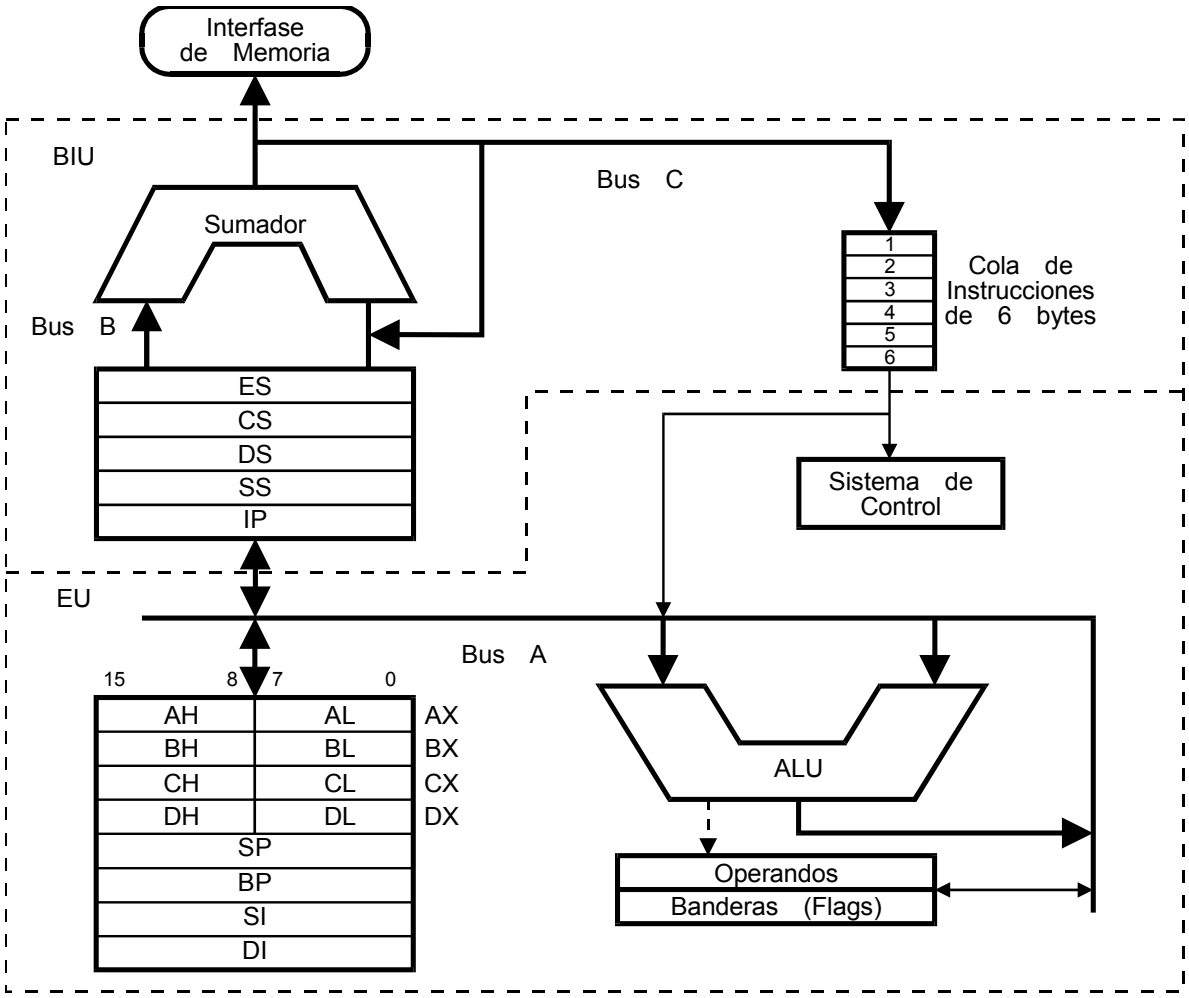


Fig. 1.1 Diagrama a bloques interno del 8086

La unidad de interface con el bus busca las instrucciones en memoria y se encarga de la lecturas y escritura a memoria y puertos. En tanto, la unidad de ejecución se encarga de decodificar y ejecutar las instrucciones. Las dos unidades se comunican por medio de una cola de instrucciones de 6 bytes en donde la BIU coloca los códigos de las instrucciones leídas de memoria. Dichos códigos permanecen en la cola hasta que son extraídos en el momento en que la EU se encuentra lista para ejecutarlos. Esta técnica para aumentar el desempeño de un procesador se conoce como **entubamiento (Pipelining)**. Una comparación entre el desempeño de un procesador sin y con entubamiento se puede observar en la Fig 1.2. Esta técnica es especialmente útil cuando el procesador cuenta con instrucciones que requieren de muchos ciclos de reloj para ejecutarse y que no ocupan ninguna transferencia a través del bus (Ej. multiplicación, división). En dicho caso la BIU puede mantenerse ocupada colocando varias instrucciones en la cola mientras la EU se termina la ejecución.

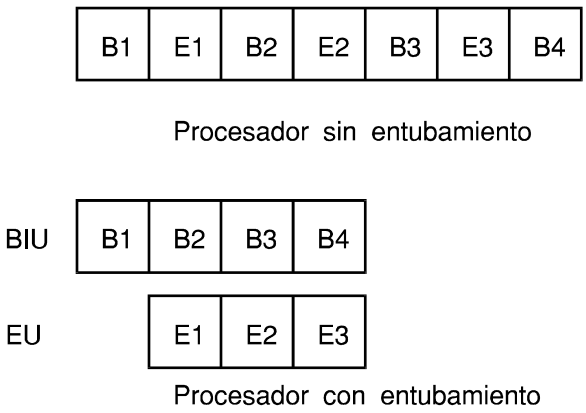


Fig. 1.2 Mejora del desempeño al usar entubamiento

1.2.2 Segmentación

Un uso muy común de un registro es servir como apuntador, guardando una dirección de memoria. En el 8086 todos los registros son de 16 bits, sin embargo, como el bus de direcciones del 8086 es de 20 bits, un solo registro no es suficiente para almacenar una dirección de memoria. Para solucionar este problema, Intel decidió usar una estrategia conocida como segmentación, consistente en formar la dirección de 20 bits a partir de dos partes de 16 bits. La primera, conocida como segmento (Segment), son los 16 bits más altos de la dirección inicial de un bloque de memoria de máximo 64 Kbytes. La segunda, llamada desplazamiento (Offset) es la distancia de la localidad de memoria al inicio del bloque. Es común representar a la pareja segmento - desplazamiento que forma una dirección de memoria usando la notación *segmento:desplazamiento*.

La dirección física de 20 bits se puede calcular por la formula *dir=segmento*16+desplazamiento*. Si se usan números hexadecimales, el calculo de la dirección se simplifica, ya que solo hay que añadir un cero a la derecha del segmento y sumar el desplazamiento. Ej. C4A3:21F8h = C4A30+21F8= C6C28h. Observe que más de una pareja segmento:desplazamiento pueden dar por resultado la misma dirección física. Por ejemplo, . C4A3:21F8h = C6C28h y C2A3:41F8h = C6C28h. Esto implica que los segmentos se traslapan entre si. Además, como el desplazamiento es de 16 bits, el tamaño máximo del sistema es de 64 K Bytes.

El segmento siempre debe estar almacenado en uno de los 4 registros de segmento que se encuentran dentro de la BIU. Cada uno de los 4 registros tiene un propósito especifico y por lo mismo un nombre distinto: CS, Code Segmet, indica el segmento donde se encuentra la siguiente instrucción a ejecutar; SS, Stack Segment, contiene el segmento de la pila del sistema; DS, Data Segment, segmento por defecto en los accesos a datos y ES, Extra Segment, un segmento de datos extra. El calculo de la dirección física a partir de la pareja segmento:desplazamiento no reduce la velocidad del procesador, ya que se cuenta con un sumador dedicado a este propósito en la BIU.

La segmentación tiene la ventaja de que permite tener cada uno de los bloques de memoria empleados por un programa claramente separados uno del otro. Además permite que el código sea relocalizable, es decir, que pueda ser ejecutado en diferentes direcciones de memoria, ya que solo es necesario cambiar el número del segmento y los desplazamientos dentro de cada segmento permanecen fijos. Sin embargo , como el tamaño máximo de un segmento es de 64 K Bytes, esto complica bastante el manejo de variables muy grandes.

1.2.3 Tabla de vectores de interrupción.

Una interrupción es un tipo especial de llamada a subrutina generada por hardware o por software. Las subrutinas llamadas de esta forma son diferentes a las subrutinas normales y reciben el nombre de rutinas de servicio de interrupción. El hardware puede generar una interrupción cuando un dispositivo periférico, tal como un puerto serie o el controlador de disco, requiere de la atención inmediata del procesador. Se detiene la ejecución del programa principal y se llama a una subrutina especial que se encarga de atender al dispositivo que solicito la interrupción. El software puede producir una interrupción como respuesta a algunas situaciones de error, o para acceder a rutinas de servicio del sistema operativo o del BIOS.

El 8086 puede tener en un momento dado hasta 256 rutinas de servicio de interrupción diferentes. El proceso para llamar a una rutina de servicio de interrupción involucra a una tabla que contiene la dirección en donde se encuentra cada rutina de servicio. Dicha tabla se conoce como la tabla de vectores de interrupción y se encuentra en los primeros 1024 Bytes de memoria, de la dirección 00000h a la 003FFh. Cada elemento de la tabla, conocido como vector, es un número de 32 bits, almacenado en cuatro localidades de memoria. Los 16 bits menos significativos contienen el desplazamiento y los 16 bits más significativos contienen el segmento de la dirección en donde se encuentra la rutina de servicio. Existe un total de 256 vectores, cada uno identificado por un número de 8 bits. Siempre que se genere una interrupción se debe indicar el número de vector de interrupción al que se desea llamar. Esta función la realiza el controlador de interrupciones 8259. La tabla1.1 muestra el uso de algunos vectores de interrupción en la PC.

Número	Operación
0	Error de división
1	Ejecución paso a paso
2	Pin NMI (verifica frecuentemente errores de paridad)
3	Punto de detención
4	Desbordamientos
5	Tecla de impresión de pantalla e instrucción BOUND
6	Instrucción ilegal
7	Emulación de coprocesador
8	Intervalo de reloj (18.2 Hz)
9	Teclado
A	IRQ2 (en cascada en los sistemas AT)
B-F	IRQ3 a IRQ7
10	BIOS de vídeo
11	Ambiente del equipo
12	Tamaño de memoria convencional
13	Servicios directos de disco
14	Servicio del puerto de comunicación serial (COM)
15	Misceláneos
16	Servicio de teclado
17	Servicio del puerto paralelo (LPT)
18	ROM BASIC
19	Reiniciar sistema operativo
1A	Servicio de reloj
1B	Controlador de detención
1C	Servicio del temporizador de usuario
1D	Apuntador a la tabla de parámetros de vídeo
1E	Apuntador a la tabla de parámetros de disco

1F	Apuntador a la tabla de patrones de caracteres gráficos
20	Finalizar programa (DOS 1 .0)
21	Servicios del DOS
22	Controlador de finalización del programa
23	Controlador de manejador ctrl-C
24	Controlador de error crítico
25	Lectura de disco
26	Escritura de disco
27	Concluye programa y permanece residente en memoria (TSR)
28	DOS inactivo
2F	Controlador multiplex
31	DPMI (interfaz DOS de modo protegido) ofrecida por Windows
33	Controlador del ratón
67	VCPI (Virtual Control Program Interface) ofrecido por HIMEM.SIS
70-77	IRQ8 a IRQl5

Tabla 1.1.- Vectores de interrupción en la PC

1.4 Lenguaje ensamblador del microprocesador

1.4.1 Registros del 8086.

Los registros del 8086 pueden ser agrupados como se muestra en la fig. 1.3. Los registros de segmento fueron discutidos en la sección anterior. Los registros de propósito general son los más comúnmente usados por el programador. Estos registros pueden usarse con la mayoría de las instrucciones de transferencia de datos, aritméticas y lógicas. Los otros tres registros tienen una función única y especifica.

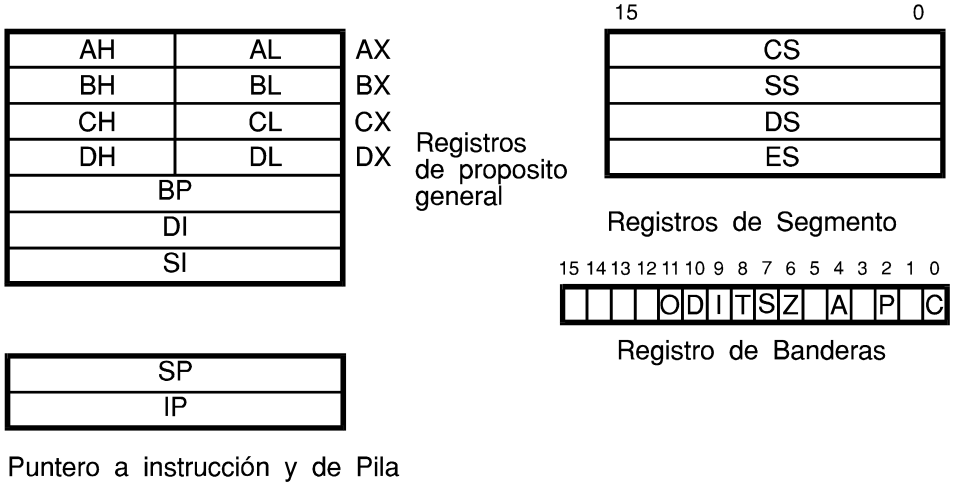


Fig. 1.3 Conjunto de registros del 8086

De los registros de propósito general, AX, BX, CX y DX pueden ser usados como si se tratara cada uno de dos registros de 8 bits cada uno. Cuando se usan de esta manera, la parte alta se designa sustituyendo la X del nombre del registro por una H y la parte baja por una L. Por ejemplo: la parte alta de CX es CH y la baja es CL. Estos registros son usados cuando se desea manipular bytes individuales.

Aunque pueden agruparse como registros de propósito general, algunos de estos registros pueden hacer ciertas tareas mejor que los demás o realizar algunas funciones que los demás no pueden, por lo que se puede decir que cada uno tiene su especialidad y un nombre descriptivo de esta:

- AX - Acumulador. Se usa en la multiplicación, división y aritmética BCD.
- BX - Base. Pude ser usado como apuntador a memoria.
- CX - Contador. Se usa para contar el numero de veces que se ejecuta una tarea repetitiva.
- DX - Datos. Puede guardar la dirección del puerto accedido en una operación de entrada o salida.
- BP - Puntero Base. Permite acceder a los parámetros de las funciones de lenguajes de alto nivel.
- SI y DI - Índice Fuente e Índice Destino. También se pueden usar como apuntadores, especialmente en las instrucciones de manejo de cadenas.

Existen además algunos registros de propósito específico, como el apuntador a instrucciones (IP, Instruction Pointer) y el apuntador a la pila (SP- Stack Pointer). La pareja CS:IP se usa para almacenar la dirección lógica de la siguiente instrucción que debe ejecutarse. IP es incrementado de forma automática cada vez que una instrucción es leída. CS e IP solo pueden ser modificados por medio de instrucciones de salto. SS:SP apuntan a la dirección donde se encuentra el tope de la pila. En el 8086 la pila crece hacia abajo, es decir que SP es decrementado cada vez que se almacena un dato en la pila. Como se trata de un microprocesador de 16 bits, solo se pueden almacenar variables de este tamaño en la pila.

El último registro que se va a analizar es el registro de banderas. Más que un registro es en realidad un conjunto de bits independientes, representando cada uno un cierto estado de la maquina. Seis de estos bits, conocidos como banderas de estado, son modificados por el procesador al final de algunas instrucciones, dependiendo de el resultado obtenido. Dentro de este grupo se encuentran:

- CF - Bandera de acarreo (Carry Flag). Indica si el resultado de la última instrucción produjo un acarreo, es decir que el resultado no cabe en el número de bits de los operandos, considerando a estos como números sin signo.
- PF - Bandera de Paridad (Parity Flag). Se pone en 1 si el número de bits iguales a 1 del resultado es par, en cero si el número es impar. En instrucciones de 16 bits solo se considera a los 8 bits menos significativos.
- AF - Bandera de Acarreo Auxiliar (Auxiliary Carry Flag). Muestra si hubo acarreo del nible bajo al nible alto de AL.
- ZF - Bandera de Cero (Zero Flag). Es puesta a 1 si el resultado fue cero, a 0 si el resultado fue distinto de cero.
- SF - Bandera de Signo (Sign Flag). Su valor es igual al del bit más significativo del resultado, el cual indica el signo cuando se interpreta al resultado como un número con signo en complemento a 2.
- OF - Bandera de Sobreflujo. (Overflow Flag). Toma un valor de 1 cuando el resultado se salió del rango de números que pueden ser representados en notación con signo en complemento a 2.

Las ultimas 3 banderas, llamadas banderas de control, son modificadas por el usuario para alterar el comportamiento del micro. Estas son:

- IF - Bandera de Interrupción (Interrupt Flag). Cuando tiene un valor de 1 se habilitan las interrupciones por hardware, con 0 de deshabilitan.
- TF - Bandera de Trampa (Trap Flag). Habilita el modo de trampa o depuración del microprocesador. En este modo, después de ejecutar una instrucción se produce una

interrupción por software que llama a un programa monitor. Es utilizado para correr programas paso a paso y depurarlos.

- DF - Bandera de Dirección (Direction Flag). Se utiliza para controlar la dirección en que la ejecución de una instrucción de manejo de cadenas recorrerá un bloque de memoria. Si DF=0 la dirección será incremental, si DF=1 será decremental.

1.4.2 declaración de segmentos (Estructura de un programa en lenguaje ensamblador)

El esqueleto básico de un programa en lenguaje ensamblador es:

```
.model small
.stack 200h
.data
      (Declaración de variables)
.code
mov  ax,@data
mov  ds,ax
      (Código del programa)
mov ah,4Ch
int 21h
end
```

La directiva .model sirve para seleccionar el modelo de memoria en un programa que utilice las directivas de segmento simplificadas. El modelo de memoria permite la compatibilidad con código escrito en lenguajes de alto nivel. El modelo de memoria tiene que ver con el tipo de punteros a código y a datos que se van ha usar. Existen dos tipos de punteros tanto a código como a datos: los de tipo NEAR (cercanos) que consisten solo en el desplazamiento de 16 bits, y los FAR (lejanos), que contienen tanto al segmento como al desplazamiento de una dirección de memoria y tienen una longitud de 32 bits. Los punteros NEAR son más pequeños y rápidos, pero no permiten tener más de un segmento y por lo tanto, limitan el tamaño de los datos y / o código a 64 K Bytes. Los modelos de memoria se forman a partir de diversas combinaciones de tipos de punteros usados para acceder el código y los datos. Los modelos de memoria existentes son:

- **tiny**. En este modelo, tanto el código como los datos deben caber en un único segmento de 64K como máximo. Se usan punteros NEAR tanto para código como para datos. Y CS=DS=SS.
- **small**. Como en el modelo tiny, los punteros a código y a datos son tipo NEAR, pero el segmento de código y el de datos son distintos. Sin embargo, todo el código del programa deben caber en un único segmento de 64K, y los datos del programa deben caber dentro de un segmento separado de 64K.
- **medium**. El código del programa puede ser mayor de 64K, pero los datos deben caber en un solo segmento. Los punteros a código son tipo FAR mientras que los de datos son NEAR.
- **compact**. El código del programa debe caber en un solo segmento de 64K, pero los datos del programa pueden ocupar más de un segmento de 64K. El código es de tipo NEAR mientras que los datos son tipo FAR.
- **large**. Tanto el código como los datos pueden ser mayores de 64K, ya que para ambos se usan punteros FAR.
- **huge**. Al igual que en el modelo large, tanto el código como los datos son tipo FAR, pero como se usan punteros normalizados, que permiten la comparación entre punteros.

La directiva .stack 200h sirve para indicar el inicio del segmento de pila. El parámetro 200h indica la cantidad de memoria que será reservada para la pila (512 Bytes). .data indica el comienzo del segmento de datos. En la mayoría de los casos es aquí donde se declaran todas las variables del programa. La directiva .code marca el inicio del segmento de código y en el se colocan todas las instrucciones del programa. Es muy común que las primeras instrucciones que se encuentran en cualquier programa sean MOV AX, @data y MOV DS, AX, las que inicialisan el registro de segmento de datos. Esto es necesario ya que al arrancar el programa solo CS y SS son inicialisados por el cargador de programas del sistema operativo. Las ultimas instrucciones del programa deben regresar el control al sistema operativo. Una forma de hacerlo es por medio del servicio 4Ch de la interrupción 21h del DOS. Finalmente, la directiva END hace que el ensamblador termine de ensamblar el archivo fuente, puede tener como parámetro opcional una etiqueta que indique cual será la primera instrucción a ejecutarse al iniciar el programa.

Declaración de variables en ensamblador.

La declaración de variables en ensamblador se hace con una línea de la siguiente forma: *{Nombre} (Directiva de definición de Memoria) inicializador*

El nombre de la variable es una etiqueta que se coloca al inicio de la línea de declaración de la variable. Este se interpreta como un nombre simbólico para representar la dirección de las localidades de memoria que se reservan. Es decir , el nombre de una variable es un símbolo que sustituye al valor numérico de la dirección que es el que se utilizara al formar los códigos de instrucción correspondientes.

Las directivas de reservado o definición de memoria sirven para definir una o más variables de un cierto tamaño. Las directivas existentes son:

Directiva	Tipo de Variable	Tamaño en bits	Tamaño en Bytes
DB	BYTE	8	1
DW	WORD	16	2
DD	DWORD	32	4
DF,DP	FWORD,PWORD	48	6
DQ	QWORD	64	8
DT	TBYTE	80	10

El inicializador sirve tanto para dar un valor inicial a la variable, como para indicar cuantas variables del mismo tipo serán definidas. Un inicializador puede ser:

1. Una constante entera con o sin signo de 8, 16, 32 o 64 bits.
2. Una constante de punto flotante de 32, 64 o 80 bits..
3. Una cadena de caracteres entre comillas, en este caso, por cada caracter de la cadena se definirá una variable inicializada con el código ASCII del caracter.
4. Un inicializador duplicado N veces, usando la forma N DUP (inicializador), donde se definen N variables inicializadas con el mismo inicializador.
5. Una lista de inicializadores separada con comas.
6. Un signo de interrogación para que la variable no se inicialice.

Ejemplos:

```
v1    db 0BAh, 0CAh ; declara dos localidades de tamaño byte
                    ; con valor inicial. La etiqueta v1 representa la
```

```
                    ; dir. de la primera localidad
v2    dd 5 DUP (?)  ; Declara 5 variables de 32 bits no inicializadas
```

1.4.3 Modos de direccionamiento.

Un modo de direccionamiento es la forma como se obtienen los operandos requeridos por una instrucción. El 8086 cuenta con muchos modos de direccionamiento muy versátiles tanto a datos como a código. Para ejemplificar el uso de estos modos de direccionamiento se usara la instrucción MOV, que realiza la asignación o transferencia de datos. La sintaxis de esta instrucción es MOV *destino,fuente* y su función es copiar el contenido del operando fuente en el operando destino. Es equivalente al enunciado *destino=fuente* en un lenguaje de alto nivel.

1.4.3.1 Modos de direccionamiento a Datos.

Modo de registro

El o los operandos están contenidos en uno o dos registros. La ejecución de la instrucción es muy rápida. El tamaño del dato se indica por el tamaño del registro.
Ejemplos:

```
MOV AL,AH   ;Cargar la parte alta de AX en la parte baja del mismo.
MOV BX,AX   ;Almacena en el registro BX el contenido de AX.
MOV ES,AX   ;Carga el Registro de segmento extra con el contenido de AX.
```

En toda instrucción que contenga 2 operandos, la longitud de los operandos destino y fuente debe ser la misma.

Modo Inmediato.

El operando fuente es una constante de 8 o 16 Bits que se encuentra inmediatamente después del código de operación y formando parte de la instrucción. La ejecución de una instrucción así es muy rápida, pues se hace únicamente a partir de los códigos almacenados en la cola de instrucciones.
Ejemplo:

```
MOV AL,50h        ;Carga el número 50h en el registro AL
MOV BX,7050h      ;Carga la constante 7050h en el registro BX.
```

Modos de direccionamiento a memoria.

Es una forma más general de los modos de direccionamiento directo e indirecto de otros procesadores. Se forma de la siguiente manera:

Registro de Segmento.	: [	Reg. Base	+	Reg. Índice	+	Desplazamiento]
CS, DS, SS, o ES		BX o BP		SI o DI		Cte. de 8 o 16 Bits

La dirección de memoria se maneja en la forma segmento:desplazamiento. El segmento es el contenido del registro de segmento y el desplazamiento se forma sumando el contenido de el registro base más el contenido de el registro índice más el desplazamiento. Algunas de las partes de este modo de direccionamiento son opcionales. La primera es el registro de segmento. Cuando se omite este, se toma por defecto a DS, con excepción de cuando se usa BP. En este caso, el segmento por defecto es SS. En el desplazamiento, tanto el registro base como el índice y la constante de desplazamiento pueden omitirse, pero al menos debe quedar una de ellas. Todas las combinaciones posibles son validas, con

excepción de [BP] y [Cte. 8 bits]. En la mayoría de los ensambladores para PC, los desplazamientos de variables se representan por el nombre simbólico de estas y es equivalente escribir `[variable]` que `variable`. También se puede escribir `[BX + DI + variable]`, `[BX][DI][variable]` o `variable[BX][DI]`.
Ejemplos:

```
MOV AL,[BX]       ;Carga en AL el contenido de la localidad de memoria
            ;cuya dirección se forma con DS como segmento y el contenido
            ;de BX como desplazamiento
MOV AX,ES:[BX+DI+50h]   ;Carga en AX el contenido de la variable de 16
            ;bits cuya dirección de memoria se forma con ES como segmento
            ;y el contenido de BX más el contenido de DI más 50h como
            ;desplazamiento.
MOV [BP+1000h],CH       ;Almacena el contenido de CH en la localidad de
            ;memoria cuya dirección se forma tomando SS como segmento y
            ;el contenido de BP más 1000h como desplazamiento
MOV DX,tabla[DI]  ;Carga en DX el contenido de la variable de 16
            ;bits cuya dirección de memoria se forma con DS como segmento
            ;y el contenido de DI más el desplazamiento de la variable
            ;tabla como desplazamiento.
```

Modos de direccionamiento a puertos.

Se utiliza para acceder localidades del espacio de direccionamiento a puertos. Es utilizado únicamente por las instrucciones IN y OUT. Existen dos formas:

- **Inmediato**: La dirección accesada se encuentra en el ultimo byte del código de instrucción. Solo se pueden acceder con esta forma las primeras 256 localidades de puerto. Ejemplo: IN AL,45h ;Lee el puerto 45h y lo guarda en AL
- **Indirecto**: La dirección de 16 bits de encuentra en el registro DX. Permite acceder a todas las localidades de puertos. Ejemplo: OUT DX,AX ;Escribe el contenido de AX en el puerto cuya dirección se encuentra en DX.

1.4.3.2 Modos de Direccionamiento a Código.

Estos modos de direccionamiento son usados en todas las instrucciones de brinco y llamadas a subrutinas. Existen tres Formas distintas:

- **Inmediato absoluto**. La dirección a la que se va ha brincar se incluye como parte de la instrucción. La dirección puede ser cercana (16 bits) o lejana (32 Bits). La cercana se almacena directamente en IP, en el caso de la lejana, los 16 bits menos significativos se cargan en IP y los más significativos en CS. Un brinco lejano es la única forma de modificar CS y por lo tanto de brincar de un segmento a otro. Casi siempre se utiliza una etiqueta simbólica para referirse a la dirección destino.
- **Inmediato Relativo**. En el código de instrucción se incluye una constante de 8 bits con signo. Esta constante es el desplazamiento relativo a la dirección apuntada por IP. La dirección a la que se va a brincar se calcula por IP=IP+Cte. Esta constante es calculada por el ensamblador a partir de una etiqueta simbólica. Como solo se modifica IP el salto siempre es dentro del mismo segmento.
- **Indirecto**. La dirección se guarda en un registro de 16 bits o localidad de memoria de 16 o 32 bits. Cuando la dirección se encuentra en memoria, se utiliza un modo de direccionamiento a datos para indicar donde esta almacenada. Para diferenciar entre los saltos indirectos cortos y los largos, se utilizan las directivas del ensamblador WORD

PTR y DWORD PTR, que indican que se trata de un puntero a palabra y doble palabra respectivamente. Los saltos cercanos se identifican con WORD PTR, y el contenido de la variable de 16 bits apuntada por el modo de direccionamiento a datos se carga en IP. Los saltos largos se identifican con DWORD PTR, la palabra baja de la variable de 32 bits se carga en IP y la palabra alta reemplaza el contenido de CS. Ejemplos: JMP BX; JMP WORD PTR[BX+DI] ; JMP DWORD PTR tabla[si].

1.4.5 Conjunto de Instrucciones.

Instrucciones de transferencia y manipulación de datos.

Todas las instrucciones de transferencia no afectan las banderas de estado.

MOV

Realiza la transferencia del operando fuente al operando destino. La fuente nunca se modifica. Los tamaños de los dos operandos deben ser iguales. Sintaxis:

```
MOV DESTINO, FUENTE
MOV {REG/MEM}, {REG/MEM/INMEDIATO}
```

Restricciones: No se pueden hacer transferencias de memoria a memoria, tampoco de memoria a un registro de segmento. No se permiten transferencias entre dos registros de segmento, ni que el registro de código sea el operando destino.

Se pueden usar los operadores de ensamblador WORD PTR o BYTE PTR para forzar al ensamblador a interpretar un nombre de variable como la dirección de una variable de tamaño palabra o byte respectivamente.

Ejemplos:

```
MOV word ptr var1, 10   ;Almacena 000Ah en la palabra que se
       ; encuenta en la dir. apuntada por la  etiqueta var1

MOV AL,var2[BX]         ;Carga en AL el contenido de la localidad de
                        ;memoria cuya dirección se forma sumando el
                        ;desplazamiento representado por var2 más el
                        ;contenido de BX
MOV DS, AX        ;Carga en el registro DS el contenido de AX
MOV BX,[BP+5]           ;Carga en BL el contenido de la localidad de
                        ;memoria del segmento de pila cuyo
                        ;desplazamiento se forma sumando el el
                        ;contenido de BP más 5 y BH con el contenido
                        ;la localidad con desplazamiento BP+6
```

XCHG

Esta instrucción intercambia el contenido de los operandos fuente y destino. XCHG no se puede ejecutar en registros de segmento ni con datos de memoria a memoria. Los operandos son de tamaño byte o palabra y se utiliza cualquiera de los modos de direccionamiento, excepto el direccionamiento inmediato.

Sintaxis:

```
XCHG DESTINO, FUENTE
XCHG {REG/MEM}, {REG/MEM}
```

Ejemplos:

```
XCHG AX,BX  ;Intercambia el contenido del registro AX con el de BX.
XCHG AL,var ;Intercambia el contenido del registro AL con el de
            ;la localidad de memoria apuntada por la etiqueta var.
```

XLAT

La instrucción XLAT reemplaza el contenido del registro AL por un byte almacenado en una tabla. Esta instrucción, primero suma el contenido de AL con BX para formar el desplazamiento de una dirección de memoria en el segmentos de datos. Luego transfiere el contenido de esa dirección a AL. El contenido de AL juega el papel de índice y en BX se almacena la dirección inicial (base) de la tabla. No requiere operandos.

AL ← DS: [BX+AL]

Sintaxis:

```
XLAT
```

Ejemplo: El siguiente programa convierte un digito hexadecimal a el código ASCII correspondiente usando una tabla que contiene los códigos ASCII. La tabla con los códigos ASCII mostrando la dirección relativa de cada elemento y su contenido se muestra a continuación:

tabla	+0	+1	+2	+3	+4	+5	+6	+7	+8	+9	+10	+11	+12	+13	+14	+15
	30h	31h	32h	33h	34h	35h	36h	37h	38h	39h	41h	42h	43h	44h	45h	46h

Se hace que BX contenga la dirección de inicio de la tabla y al sumar AL se obtiene la dirección del elemento correspondiente al digito hexadecimal en la tabla.

```
        .model small
        .stack 200h
        .data
tabla   db "0123456789ABCDEF" ;tabla con los codigos
digito  db 10
ascii   db ?
        .code
        mov ax,@data   ;se hace que ds apunte al segmento de
        mov ds,ax     ;datos
        mov bx, offset table ;bx apunta al inicio de la tabla
        mov al, digito ;al escoge el elemento de la tabla
        xlat          ;se lee el elemento de la tabla
        mov ascii, al ;almacena el resultado
        mov ah,4ch    ;termina el programa
        int 21h
        end
```

SAHF

La instrucción SAHF transfiere el registro AH hacia los bits correspondientes de la parte baja del registro de estado, o sea, a los indicadores SF, ZF, AF, PF y CF. SHAF solo modifica al registro de banderas y no requiere operandos.

AH → Banderas 0-7

Sintaxis:

```
SAHF
```

LAHF

La instrucción LAHF transfiere los indicadores SF, ZF, AF, PF y CF del registro de estado a las posiciones 7,6,4,2 y 0 respectivamente del registro AH. No requiere operandos.

AH ← Banderas 0-7

Sintaxis:

```
LAHF
```

CBW

La instrucción CBW extiende el signo del byte del registro AL a cada uno de los bits del registro AH. CBW convierte un número con signo de tamaño byte almacenado en AL a uno de tamaño palabra en AX. No requiere operandos.

Sintaxis:

```
CBW
```

Ejemplos:

```
si    AX = 1011 0010 0110 0111 b
después de  CBW
      AX = 0000 0000 0110 0111 b
si    AX = 1011 0010 1110 0111 b
después de  CBW
      AX = 1111 1111 1110 0111 b
```

CWD

CWD es la instrucción de extensión de signo de una palabra contenida en el registro AX a todos los bits del registro DX. Esta instrucción puede utilizarse para obtener un dividendo de doble longitud (doble palabra, o sea, 32 bits) antes de realizar la división de 2 palabras con signo. La instrucción CWD convierte un número con signo de tamaño palabra almacenado en AX a uno de doble palabra almacenando la palabra baja en AX y la alta en DX. No requiere operandos.

AX → DX,AX con extensión de signo**.**

Sintaxis:

```
CWD
```

LEA

(Load Efective Address). La instrucción LEA carga cualquier registro de 16 bits, indicado en la instrucción, con el desplazamiento de la dirección efectiva de un modo de direccionamiento a memoria.

Sintaxis:

```
LEA REG16 , MEM
```

Ejemplo:

```
LEA AX, NUMB      ;Se carga AX con el desplzamiento de la dirección
                  ;en que se almacena la variable NUMB.
LEA BX, VAR[SI]   ;Almacena en BX el resultado de sumar el contenido
                  ;de SI  más el desplazamiento de VAR
```

LDS

La instrucción LDS sirve para cargar un apuntador lejano almacenado en memoria, en una variable de 32 bits. Copia la palabra baja de la variable al registro especificado en la instrucción. Después copia la palabra alta de la variable al registro DS. En esta instrucción se utiliza un modo de direccionamiento a memoria para indicar la dirección de la variable de 32 bits que contenga el segmento y el desplazamiento.

Sintaxis:

```
LDS REG16, MEM
```

Ejemplos:

```
LDS DI, VAR ;Se carga DI y DS con el puntero almacenado en VAR.
LDS BX,[DI] ;Carga BX con la palabra apuntada por [DI] y DS con la
            ;apuntada por [DI+2].
```

LES

La instrucción LES es idéntica a la instrucción LDS, con la excepción de que en vez de almacenar el segmento del puntero en DS lo hace en ES.

Sintaxis:

```
LES REG16, MEM
```

Transferencias con la pila.

PUSH

La instrucción PUSH siempre transfiere 2 bytes de datos a la pila. La fuente de los datos puede ser cualquier registro interno de 16 bits, datos inmediatos, cualquier registro de segmento o un palabra de la memoria. Esta instrucción decrementa SP en 2 unidades y luego transfiere el operando indicado en la instrucción a la pila a partir de la dirección apuntada por SS:SP.

Sintaxis:

```
PUSH {REG16/MEM16}
```

Ejemplos:

```
PUSH BX     ;Decrementa en 2 a SP, copia BX a la dirección apuntada
            ;por SS:SP.
PUSH VAR    ;Decrementa en 2 a SP, copia el contenido de VAR
            ;al segmento de pila.
```

PUSHF

(Push Flags) La instrucción PUSHF transfiere el contenido del registro de banderas a la pila. Decrementa a SP en 2 unidades y luego transfiere el contenido del registro de estado a la pila en la dirección apuntada por SP. No requiere operandos.

Sintaxis:

```
PUSHF
```

PUSHA

(Push All) La instrucción PUSHA salva a los registros en la pila en el siguiente orden: AX, CX, DX, BX, SP, BP, SI y DI. Decrementa a SP en 2 unidades antes de almacenar cada registro. No requiere operandos.

Sintaxis:

```
PUSHA
```

POP

La instrucción POP recupera de la pila los datos almacenados por las instrucciones PUSH. Esta instrucción transfiere el dato de la pila apuntado por SP al operando destino y luego incrementa a SP en dos. El operando destino puede ser cualquier registro de propósito general, registro de segmento o una localidad de memoria.

Sintaxis:

```
POP {REG16/MEM16}
```

Ejemplos:

```
POP AX      ; recupera la palabra del tope de la pila
            ;y la almacena en AX
```

POPF

(Pop flags) La instrucción POPF hace que la palabra apuntada por el registro SP se transfiera al registro de estado, y a continuación se incrementa SP en 2 unidades.

Sintaxis:

```
POPF
```

POPA

(Pop All) La instrucción POPA recupera los registros de la pila en el siguiente orden: DI, SI, BP, SP, BX, DX, CX y AX. Incrementa SP en 2 unidades después de recuperar cada registro. No requiere operandos.

Sintaxis:

```
POPA
```

Transferencias con puertos.

IN

La instrucción IN (lectura de un puerto) transfiere datos de un dispositivo de E/S a los registros AL o AX. Si se lee a AL el puerto se maneja de tamaño byte, si se lee a AX es de tamaño palabra y se leen dos direcciones de puerto consecutivas. La dirección que se lee puede ser una constante de 8 bits o el contenido de DX, dependiendo de si se usa direccionamiento a puertos inmediato o indirecto. Los puertos arriba de 00FFh solo se pueden leer por medio de DX usando direccionamiento a puertos indirecto.

Sintaxis:

```
IN {AL/AX}, {DX/BYTE}
```

Ejemplos:

```
IN AX,3Fh   ;Lee el puerto de 8 bits de la localidad 3Fh en AL
            ;y el de la localidad 40h (3Fh+1)en AH.
IN AL,DX    ;Lee el puerto de 8 bits cuya dirección se
            ;encuentra en DX
```

OUT

Escribe a un puerto el contenido del lo registro AL o AX a un dispositivo de E/S apuntado por una constante de 8 bits o por DX.

Sintaxis:

```
OUT {DX/BYTE}, {AL/AX}
```

Manipulación de Bits de Bandera.

Estas instrucciones permiten modificar algunas de las banderas de manera independiente. Ninguna recibe operandos.

CLD

CLD borra la bandera de dirección: DF ← 0

CLC

CLC borra la bandera de acarreo: CF ← 0

CLI

CLI borra la bandera de habilitación de interrupciones: IF ← 0.

STD

STD Pone a uno la bandera de dirección: DF ← 1

STC

STC Pone a uno la bandera de acarreo: CF ← 1

STI

STI Pone a uno la bandera de habilitación de interrupciones: IF ← 1

1.6.5 Instrucciones aritméticas.

Todas las instrucciones aritméticas modifican las banderas de estado de acuerdo al resultado de la operación.

ADD DESTINO ← DESTINO + FUENTE

La instrucción ADD suma 2 operandos de 8 ó 16 bits con o sin signo. El resultado toma el lugar del operando destino y se afecta a las siguientes banderas: CF, AF, PF, ZF, SF, OF. Los únicos tipos de sumas que no se permiten son las de memoria a memoria y registros de segmento.

Sintaxis:

```
ADD {REG/MEM}, {REG/MEM/INMED}
```

Ejemplo:

```
ADD AX, 2          ; Suma 2 al contenido del registro AX
ADD CX,VAR         ; Suma el contenido de VAR al contenido de CX
ADD VAR2, AL       ; Suma el byte contenido en AL a localidad
                   ;de memoria VAR2
```

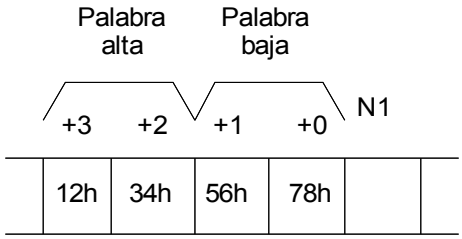
ADC DESTINO ← DESTINO + FUENTE + CF

Suma con acarreo. Similar a la instrucción ADD, pero suma también la bandera de acarreo. Se usa para implementar sumas de números de más de 16 bits.

Sintaxis:

```
ADC {REG/MEM}, {REG/MEM/INMED}
```

Ejemplo: El siguiente programa utiliza la instrucción ADC para sumar las partes altas de dos números de 32 bits. Para conocer la dirección relativa al inicio de las variables en donde se encuentran la palabra alta y baja, se usa el siguiente diagrama que muestra como se almacena el número 12345678h en memoria:




```
;Programa que suma dos numeros de 32 bits N3=N1+N2
     .model small
     .stack 200h
     .data
N1   dd 12345678h
N2   dd 9abcdef0h
n3   dd ?
```

```
.code
mov ax,@data
mov ds,ax

mov ax, word ptr N1 ;carga la palabra baja de N1
add ax, word ptr N2 ;suma la palabra baja de N2
mov word ptr N3, ax ;almacena la parte baja
                    ;del resultado en N3

mov ax, word ptr [N1+2];Carga la parte alta de N1
adc ax, word ptr [N2+2];suma la parte alta de N2 y CF
mov word ptr [N3+2], ax ;almacena la parte alta
                    ;del resultado en N3


mov ah,4ch ;termina el programa
int 21h
end
```

SUB DESTINO ← DESTINO - FUENTE

Resta sin acarreo, almacena el resultado en el operando destino.

Sintaxis:

```
SUB {REG/MEM}, {REG/MEM/INMED}     8/16 bits
```

SBB DESTINO ← DESTINO - FUENTE - CF

Resta con préstamo. Similar a la instrucción SUB, pero también resta la bandera de acarreo. Se usa para implementar restas de números de más de 16 bits.

Sintaxis:

```
SBB {REG/MEM}, {REG/MEM/INMED}
```

CMP DESTINO - FUENTE

Comparación. La instrucción para comparación es una resta que sólo afecta los bits de bandera, es decir, no modifica al operando destino. La comparación es útil para comparar el contenido de un registro o una localidad de memoria contra otro valor. La instrucción CMP suele ir seguida por una instrucción de brinco condicional (Jcond) que prueba el estado de los bits de bandera.

Sintaxis:

```
CMP {REG/MEM}, {REG/MEM/INMED}
```

INC DESTINO ← DESTINO + 1

Incremento. Suma una unidad al operando. Esta instrucción afecta a las siguientes banderas: AF, OF, PF, SF y ZF. Al no afectar a CF no se puede usar para aritmética de más de 16 bits.

Sintaxis:

```
INC {REG/MEM}     8/16 bits
```

DEC DESTINO ← DESTINO - 1

Decremento. Resta una unidad al operando. Esta instrucción afecta a las mismas banderas que la instrucción INC.
Sintaxis:

```
DEC {REG/MEM}     8/16 bits
```

NEG DESTINO ← 0 - DESTINO

Negar. La instrucción NEG hace que el operando indicado en la instrucción se reemplace por su complemento a dos. Esta instrucción invierte el signo del operando destino. Esta instrucción afecta a todas las banderas.
Sintaxis:

```
NEG {REG/MEM}     8/16 bits
```

MUL

Multiplicación sin signo de números enteros de 8 o 16 bits. El producto después de una multiplicación es siempre del doble de tamaño que los operandos.

En la multiplicación de 8 bits el multiplicando está siempre en el registro AL, el multiplicador puede ser cualquier registro de 8 bits o cualquier localidad de la memoria; el producto se guarda en AX. No se permite la multiplicación por una constante inmediata.

La multiplicación de palabras es muy semejante a la multiplicación de bytes. La diferencia es que el multiplicando está en AX en vez de AL y el producto aparece en DX-AX en lugar de AX. El registro DX siempre contiene los 16 bits más significativos del producto y AX los 16 bits menos significativos.

Las banderas de PF, ZF, SF, AF quedan indefinidas al efectuarse la multiplicación. las banderas CF y OF se ponen a uno si los bits más significativos (AH para operandos de 8 bits y DX para operandos de 16 bits) no son igual a cero.
Sintaxis:

```
MUL {REG/MEM}     8/16 bits
```

Tamaño	Multiplicando	Multiplicador	Producto
8 Bits	AL	REG8 / MEM	AX
16 Bits	AX	REG16 / MEM	DX-AX

IMUL

La instrucción IMUL es la multiplicación de dos números con signo de 8 ó 16 bits. Esta instrucción maneja los operandos de manera similar a MUL, pero los considera como números con signo
Sintaxis:

```
IMUL {REG/MEM}    8/16 bits
```

DIV

División sin signo de números de 8 o de 16 bits. El dividendo siempre es del doble de tamaño que el divisor. Esto significa que en una división de 8 bits se divide un número de 16 bits entre uno de 8 bits; en una división de 16 bits se divide un número de 32 bits entre uno de 16 bits. No está disponible la división por una constante inmediata. Ninguno de los bits de bandera tiene un cambio predecible en una división.

El único operando que tiene es el divisor, el cual indica si se trata de una instrucción de 8 o de 16 bits.

Una división de 8 bits utiliza el registro AX para almacenar el dividendo, que se divide entre el contenido de cualquier registro o localidad de memoria de 8 bits. El cociente se transfiere al registro AL después de la división y el registro AH contiene el residuo.

La división de 16 bits es semejante a la división de 8 bits, excepto que toma un dividendo de 32 bits que se encuentra en DX-AX (DX palabra alta, AX palabra baja) y el divisor es el contenido de un registro o variable de 16 bits. El cociente aparece en AX y el residuo en DX después de una división de 16 bits.

Sintaxis:

```
DIV {REG/MEM}
```

Tamaño	Dividendo	Divisor	Cociente	Residuo
8 Bits	AX	REG8 / MEM	AL	AH
16 Bits	DX-AX	REG16 / MEM	AX	DX

IDIV

División con signo. Similar en sintaxis y operación a DIV pero considera a todos los operandos como números con signo.

Sintaxis:

```
IDIV {REG/MEM}
```

1.6.6 Instrucciones de bifurcación.

JMP

La instrucción JMP efectúa un salto incondicional a una dirección. Hay tres tipos de instrucciones para brinco incondicional disponibles en el conjunto de instrucciones del microprocesador: salto corto relativo, cercano y lejano.

El salto corto (short) o relativo es una instrucción de 2 bytes que permite transferir el control del programa a localidades de memoria que están dentro del intervalo de +127 bytes y -128 bytes de la localidad de la memoria después de la instrucción de salto

El brinco cercano (near), de 3 bytes permite transferir el control del programa a cualquier dirección dentro del segmento de código actual, ya que solo modifica a IP. Puede usar los modos de direccionamiento a código inmediato absoluto o indirecto.

El brinco lejano (far), de 5 bytes permite saltar a cualquier localidad en la memoria dentro de todo el espacio de memoria, ya que se modifican tanto a CS como a IP, pudiéndose saltar a cualquier segmento. También pueden usar los modos de direccionamiento a código inmediato absoluto o indirecto.

En ensamblador normalmente se marca la instrucción a donde se quiere brincar con una etiqueta y el ensamblador escoge el tipo de brinco más conveniente para saltar a esa etiqueta. Esto funciona con los modos de direccionamiento a código absoluto y relativo, no con el indirecto.

Sintaxis:

```
JMP {REG/MEM/DIR}       16 bits
JMP {MEM/DIR}           32 bits
```

Saltos condicionales

Las instrucciones para brinco condicional prueban una cierta condición. Si la condición probada es verdadera, se transfiere el control del programa a la etiqueta relacionada con la instrucción de brinco. En caso contrario, se ejecuta la siguiente instrucción en la secuencia del programa. La condición probada puede ser el estado de uno de los siguientes bits de bandera: signo (S), cero (Z), acarreo (C), paridad (P) y sobreflujo (O). En tal caso, los mnemónicos de las instrucciones que saltan si la bandera esta en 1 se forman con J y la letra de la bandera, los mnemónicos de las que saltan si la bandera esta en cero se forman con JN y la letra de la bandera. Ejemplo: JC y JNC.

También se puede probar si se cumple un operador relacional entre los operandos de una instrucción de comparación ejecutada justo antes del brinco condicional. Los mnemónicos para dichos operadores se dan en la siguiente tabla:

Operador	Para números con signo	Para números sin signo
>	G, NLE	A, NBE
<	L, NGE	B, NAE
>=	GE, NL	AE, NB
<=	LE, NG	BE, NA
=	E	E
≠	NE	NE

Sintaxis:

```
Jcond byte
donde: cond = 1,2 o 3 letras de condición
       byte = etiqueta relativa
```

Ejemplos:

```
JC Etiqueta       ;Brinca a Etiqueta si CF=1, si no ejecuta la
                  ; siguiente instrucción

JNP Etiqueta      ;Brinca a Etiqueta si PF=0

CMP AX,BX
JGE Etiqueta      ;Brinca a Etiqueta si AX es mayor igual que BX

CMP VAR,5
JNB Etiqueta      ;Brinca a Etiqueta si VAR no es menor que 5
                  ;como número sin signo
```

LOOP

LOOP es una instrucción de ciclo. Permite repetir un grupo de instrucciones el número de veces contenido en CX. Al ejecutarse, primero decrementa CX en una unidad. Si el contenido de CX es diferente de cero brinca a la dirección indicada por la etiqueta, usando modo de direccionamiento relativo (-128 a +127 Bytes). Si CX es cero se ejecuta la siguiente instrucción. Esta instrucción no afecta a los bits de bandera.

Sintaxis:

```
LOOP etiqueta
```

Ejemplo:

```
          mov cx,5    ;Número de veces que se va repetir el ciclo
ciclo:    mov ax,bx   ;esta instrucción se ejecuta 5 veces
          loop ciclo  ;Decrementa CX y brinca si CX<>0
```

LOOPE/LOOPZ

Ciclo mientras es igual. Se usa para implementar ciclos, al igual que LOOP. Básicamente se ejecuta como LOOP, pero además se revisa el estado de la bandera ZF como parte de la condición de salida del ciclo. Si ZF=0 (No igual después de CMP) el salto no se realiza y el ciclo se termina antes de que CX llegue a cero.
Sintaxis:

```
LOOPE etiqueta
```

LOOPNE/LOOPNZ

La instrucción LOOPNE (repite mientras no sea igual) repite un ciclo mientras CX sea diferente de cero y ZF = 0 (No igualdad). Saldrá del ciclo si la condición es de igualdad (ZF=1) o si el registro CX es cero después de decrementarlo.
Sintaxis:

```
LOOPNE etiqueta
```

CALL

La instrucción CALL es de llamada a un subprograma o subrutina, también denominado procedimiento. CALL transfiere el flujo del programa al procedimiento. La instrucción CALL difiere de las instrucciones para brinco porque CALL salva una dirección para retorno en la pila. Existen dos versiones de la instrucción, una cercana (NEAR) y otra lejana (FAR). En la cercana, se brinca a un procedimiento dentro del mismo segmento, por lo que solo se almacena IP como dirección de retorno. En la lejana se puede llamar a un procedimiento en otro segmento, por lo que se debe almacenar tanto CS como IP en la pila para tener la dirección de retorno completa. Con la instrucción CALL puede usar cualquier modo de direccionamiento a código.
Sintaxis:

```
CALL {REG/MEM/INMED}
```

RET

La instrucción RET es el retorno de subprograma o subrutina. Cuando se ejecuta la instrucción RET, la dirección de retorno devuelve el control a la instrucción que sigue a CALL. Existen retornos cercanos y lejanos, que corresponden con las llamadas cercanas y lejanas. La instrucción de retorno (RET) extrae un número de 16 bits (retorno cercano) de la pila y lo coloca en IP o bien un número de 32 bits (retorno lejano) y lo coloca en IP y CS.
Sintaxis:

```
RET
```

INT

Llamada a Interrupción por software. Esta instrucción causa que el 8086 llame a un procedimiento lejano en una manera similar a la que corresponde a una señal de interrupción en las terminales de entrada INTR o NMI. La llamada se realiza de modo indirecto, a través de la tabla de vectores de interrupción. El parámetro de la instrucción es un número de 8 bits entre 0 y 255 que identifica a la interrupción. Los pasos que toma el 8086 para ejecutar una instrucción INT son:

- Decrementa el puntero de pila en 2 y almacena el registro de banderas en la pila.
- Decrementa SP en 2 y almacena el contenido de CS en la pila.
- Decrementa SP en 2 y almacena el contenido de IP en la pila, el cual corresponde con el desplazamiento de la siguiente instrucción después de la interrupción.

- Obtiene un nuevo valor para IP de la dirección física igual a cuatro veces el número de vector especificado en la instrucción. Como ejemplo, para INT 8h, el nuevo valor de IP se leería de la dirección 00020h.
- Obtiene un nuevo valor para CS de la dirección física igual a cuatro veces el número de vector más 2. Como ejemplo, para INT 8h, el nuevo valor de CS se leería de la dirección 00022h.
- Pone en cero las banderas IF y TF. Ninguna otra bandera se modifica.

Sintaxis:

```
INT Byte
```

IRET

Cuando un 8086 responde a una interrupción generada por hardware o software, almacena en la pila el registro de banderas, CS e IP. Entonces carga CS e IP con la dirección inicial de la rutina de servicio de interrupciones. La instrucción IRET es usada al final de la rutina de servicio para regresar el control al programa interrumpido. Para realizar esto el procesador copia el valor de IP que fue almacenado en la pila de regreso a IP. Posteriormente recupera el valor almacenado en la pila de CS y finalmente el de el registro de banderas. Las banderas tendrán el valor que tenían antes de la interrupción.