ENSAMBLADOR

1. Sistemas Numéricos

Un *sistema numérico* es un conjunto de reglas y símbolos que nos permiten escribir números.

Números: reales negativos y positivos, y enteros negativos y positivos.

Representación de números enteros no negativos

Sea R (base o radio) un número mayor o igual que 2, entonces pueden representarse números enteros como una cadena de dígitos escogidos entre 0, 1, 2, ..., R-1. Donde la cadena es la representación en base R del entero.

La base de un sistema numérico es el número de dígitos que pueden aparecer en cada posición en el sistema numérico.

Ejemplo: R=3 dígitos={0,1,2}

Base 10 → 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, ...
Base 3 → 0, 1, 2, 10, 11, 12, 20, 21, 22, 100, 101, 102, 110, 111, 112, 120, 121, 122,

Conversión entre bases

Sea el número $a_k a_{k-1} \ldots a_1$, un entero en base R.
Para convertir este número de base R a base Q utilizamos la conversión:

$$a_k R^{k-1} + a_{k-1} R^{k-2} + \ldots + a_1 R^0 \qquad \textbf{(Expresión uno)}$$

Donde R es la base en la que se encuentra el número (base actual), k es el número de dígitos que conforman el número y Q es la nueva base (se debe trabajar con aritmética en base Q).

Ejemplos:
1) Convertir $(100110)_2$ a $(\quad)_{10}$
R=2 k=6 Q=10 $a_6=1$, $a_5=0$, $a_4=0$, $a_3=1$, $a_2=1$, $a_1=0$
$1*2^{6-1}+0*2^{6-2}+0*2^{6-3}+1*2^{6-4}+1*2^{6-5}+0*2^{6-6} = 1*2^5+0*2^4+0*2^3+1*2^2+1*2^1+0*2^0 =$
$1*2^5+1*2^2+1*2^1 = 32+4+2 = 38$

Finalmente obtenemos que: $(100110)_2 \rightarrow (38)_{10}$

2) Convertir $(4302)_5 \rightarrow (\quad)_3$
R=5 Q=3 k=4 $a_4=4$, $a_3=3$, $a_2=0$, $a_1=2$

Se debe trabajar con aritmética en base 3, por lo tanto necesitamos las tablas de suma y multiplicación en base 3.

$4*5^3+3*5^2+0*5^1+2*5^0 = 11*12^3+10*12^2+2*12^0 = 11*11122+10*221+2*1 =$
$200112+2210+2 = 210101$

Realizando las sumas y multiplicaciones debidas en base 3, obtenemos:

$(4302)_5 \rightarrow (210101)_3$

Conversión de números de base R a base 10 (donde R≠10) Algoritmo 1.
(Regla de Horner para la evaluación de polinomios)

1. i <– k, num <– 0
2. Mientras i<=1 hacer
   num <– num*R+a_i
   i <– i-1
   fin_mientras
3. Fin

Ejemplo:

Convertir $(4302)_5 \rightarrow (\ )_{10}$

Conversión de números de base 10 a base S (donde S ≠ 10) Algoritmo 2.

1. i <– 1, q <– 0, p <– 0
2. Repetir
   q <– [x/s] (parte entera)
   p <– x-q*s (residuo)
   a_i <– p, i <– i+1, x <– q
   hasta q=0
3. Fin

Ejemplo:

Convertir $(577)_{10} \rightarrow (\ \ )_3$

Conversión de números de base X a base 10 (donde X ≠ 10)
Números fraccionarios Algoritmo 3.

1. i <– m, num <– 0
2 .Mientras i<=1hacer
   num <– (num+bi)/X
   i <– i-1
   fin_mientras
3. Fin

Ejemplo:

Convertir $(.A06)_{16} \rightarrow (\quad)_{10}$

Conversión de números de base 10 a base S (donde S ≠ 10)
Números fraccionarios Algoritmo 4.

1. i <– 1
2. Mientras i <= m hacer
   x <– x*s
   y <– [x] (parte entera)
   x <– x-y, b_i <– y, i <– i+1
   fin_mientras
3. Fin

Donde m es el número de dígitos que se desean obtener, x es el número a convertir inicialmente, S es la nueva base y b_i es el i-ésimo dígito del número en base s tomando el orden $b_1 b_2 \ldots b_m$

Conversión de potencias de 2

Para convertir números de base 2 a base k, donde k puede expresarse como una potencia de 2, es decir, $k=2^x$ donde x>1 y es un número entero, se llevan a cabo los siguientes pasos:

1. Se agrupan de x en x los dígitos que se encuentran a la izquierda del punto, comenzando a partir de él y aumentando ceros a la izquierda cuando es necesario.
2. Se agrupan de x en x los dígitos que se encuentran a la derecha del punto comenzando a partir de éste y aumentando ceros a la derecha cuando sea necesario.
3. Se sustituyen los grupos por los dígitos correspondientes en la base k.

Ejemplo:

$(1110010100.011011)_2$ a $(\ )_{16}$ Donde $16=2^4$

0011 1001 0100 . 0110 11**00** Resultado:

3 9 4 . 6 C $(1110010100.011011)_2$ a $(394.6C)_{16}$

Conversión de potencias de 2

Para convertir números de base $k=2^x$ a base 2, se sustituye cada dígito en base k por los x dígitos binarios correspondientes.

Ejemplo:

$(7402.61)_8$ a $(\ )_2$ Donde $8=2^3$

7 4 0 2 . 6 1 Resultado:

111 100 000 010 110 001 $(7402.61)_8$ a $(111100000010.110001)_2$

Complemento

El complemento es una forma de representar números negativos.

Si la base es 2, existen dos clases de complementos: complemento a 1 y complemento a 2.

- **Complemento a 1**. Se obtiene cambiando 1's por 0's y 0's por 1's. Ejemplo: Sea el número 00111100, su complemento a 1 es: 11000011.

- **Complemento a 2**. Se aplica complemento a 1 al número y luego se suma 1 al resultado. Ejemplo: Sea el número 0110110 aplicando complemento a 1 obtenemos: 1001001, después se le suma 1, obteniéndose 1001010

Algoritmo de suma utilizando la representación de números negativos mediante signo y magnitud.

1. Sean $a_n a_{n-1} \ldots a_0$ y $b_n b_{n-1} \ldots b_0$ 2 números binarios con signo y magnitud.
2. Tienen signos iguales ? ($a_n = b_n$)
   Si: sumar magnitudes quedando el resultado en $c_{n-2} c_{n-2} \ldots c_0$, $c_n = b_n = a_n$
   No: Comparamos magnitudes y dejamos en c_n el signo del mayor. Restamos a la magnitud mayor la menor y el resultado queda en $c_{n-1} c_{n-2} \ldots c_0$
3. La magnitud de $c_{n-1} c_{n-2} \ldots c_0$ excede el rango?
   Si: Indicar error (overflow – sobreflujo)
   No: El resultado esta en $c_n c_{n-1} \ldots c_0$

Ejemplo: Obtener el resultado de las siguientes sumas binarias a 4 dígitos

1) 5+(-3)=0101+1011 Los signos son diferentes, y la magnitud del primer número es mayor que la del segundo, así que restamos 011 de 101 y el signo del resultado será positivo.
   =0010 Su equivalente decimal es 2

2) (-4)+(-6)=1100+1110 Los signos son iguales, así que se suman magnitudes
   Error ! Existe overflow

Algoritmo de suma algebraica en complemento a 1

1. Tomar el complemento a 1 de los números negativos
2. Sumar los operandos
3. Existe carry? Si: sumar 1 al resultado
4. Existe overflow?
   Si: indicar error
   No: Escribir el resultado

Ejemplo: Utilizar 4 dígitos
(-4) + (-3)= $(0100)^{c1}+(0011)^{c1}$= 1011+1100= 1 0111 Existe carry
=0111+1=1000 No existe overflow

Algoritmo de suma algebraica en complemento a 2

1. Tomar el complemento a 2 de los números negativos
2.Sumar los operandos
3.Existe overflow? Si: mensaje de error
No: Se toman las primeras n posiciones de derecha a izquierda como resultado ignorando el carry si es que lo hay.

Ejemplo: (4 dígitos)
7+(-5)= 0111+$(0101)^{c2}$= 0111+1011= 1 0010 Existe carry, así que el resultado es 0010

Códigos

Toda la información en la computadora es almacenada en dígitos binarios, los cuales son representados por medio de un agrupamiento de los mismos como caracteres de instrucciones o números. Estos agrupamientos definen códigos, de ahí que un código pueda definirse como un conjunto de reglas para interpretar grupos de bits.

Existen tres tipos de códigos:
1. *Caracteres.* EBCDIC
2. *Instrucciones.*
3. *Numéricos.* BCD (Binary Code Decimal) y GRAY

BCD. Es una técnica para representar números decimales en las computadoras. Se refiere al almacenamiento de números decimales en un byte, esto es, 4 dígitos binarios para cada dígito decimal.

0=0000	2=0010	4=0100	6=0110	8=1000
1=0001	3=0011	5=0101	7=0111	9=1001

Ejemplo: 64=01100100 17=00010111

GRAY. Es una técnica de almacenamiento en grupos de 4 bits, donde pueden representarse 16 números diferentes.

0=0000	4=0110	8=1100	12=1010
1=0001	5=0111	9=1101	13=1011
2=0011	6=0101	10=1111	14=1001
3=0010	7=0100	11=1110	15=1000

EBCDIC (Extended Binary Coded Decimal Interchange Code). En una clave de intercambio de código decimal cifrada en binario donde se utilizan 8 bits para representar 256 posibles caracteres.

‘A’ --- c1	0 --- f0	‘.’ --- 4b
‘b’ --- c2	1 --- f1	‘(‘ --- 4d
....	...	‘+’ --- 4e ….

ASCII (American Standard Code for Information Interchange). Es una clave Americana, se define como una clave para intercambio de información donde se utilizan 7 bits para representar 128 caracteres existentes en este código, 32 de los cuales son caracteres de control para las comunicaciones. (pantalla, impresora, unidad de disco, etc.)

'.' --- 2e	'A' --- 41	'a' --- 61	0 --- 30
	'B' --- 42	'b' --- 62	1 --- 31

Descripción de una Computadora

De manera general una computadora puede definirse como un rápido y exacto sistema de manipulación de datos, diseñada y organizada para aceptar y almacenar información, procesarla y producir información de salida. Una computadora consta de dos partes: el *hardware y el software*.

Hardware. Son los componentes electrónicos que forman a la computadora, esto es, el teclado, el monitor, los circuitos integrados.

Software. Es el conjunto de instrucciones que ejecuta una computadora.

Existen dos tipos de computadoras: *digitales y analógicas*.

Esto se debe a que los datos que las computadoras manejan se dividen en *continuos y discretos*.

Discretos. Son aquellos que resultan de un conteo, por ejemplo, el total de alumnos en un salón de clases.

Continuos. Son aquellos que resultan de la medición, por ejemplo, la velocidad de un automóvil, obtenida a través de un velocímetro.

De acuerdo a las definiciones anteriores tenemos que:

Una computadora digital es un dispositivo de cálculo que procesa datos discretos.

Una computadora analógica es un dispositivo de cálculo que procesa datos continuos.

A través del tiempo se han desarrollado más las computadoras digitales y su principal característica es la velocidad, además de ser poco costosas.

Arquitectura de un computadora digital.

El diseño básico operacional de un sistema de cómputo se conoce como arquitectura A John Von Neumann, un pionero del diseño de la computadora, se le da crédito de la mayoría de las arquitecturas de las computadoras actuales. Por ejemplo la familia 80x86 usa una arquitectura Von Neumann (VNA). Un sistema típico de NVA tiene tres componentes: memoria, una unidad central de procesamiento (CPU) y dispositivos de entrada/salida (I/O).

En las máquinas VNA, el CPU es donde toda la acción tomar lugar. Todos los cálculos ocurren dentro del CPU. Tanto los datos como las instrucciones residen en memoria hasta que son requeridas por el CPU.

El bus del sistema.

El bus del sistema conecta varios componentes de una máquina VNA. La familia 80x86 tiene 3 buses principales: el bus de direcciones, el bus de datos, y el bus de control. Un bus es una colección de cables sobre los cuales las señales eléctricas pasan entre los componentes en el sistema, estos buses pueden variar.

El bus de datos. En los procesadores 80x86 se usan para intercambiar datos entre los diferentes componentes de las computadoras. El tamaño del bus varía, así en el 8086 tiene 16 bits, en el 80386DX, 80486 y Pentium Overdrive tienen un bus de 32 bits; el Pentium y el Pentium Pro tienen un bus de datos de 64 bits. El bus de datos en la familia 80x86 transfiere información entre una localidad de memoria particular o I/O y el CPU.

El bus de dirección. Localiza la memoria o el dispositivo de I/O de donde va a transferir información el bus de datos.

El bus de control. Es una colección de señales que controlan como el procesador se comunica con el resto del sistema. El bus de control dirige el flujo de cómo se maneja la información dentro del sistema. En el bus de control existen dos líneas, una de lectura y otra de escritura los cuales especifican la dirección del flujo de datos.

El CPU.

La **ALU** es el lugar donde se realiza realmente el trabajo de procesamiento. Esta unidad se encarga de efectuar sobre los datos operaciones aritméticas, lógicas, de comparación, de movimiento de bits, etc. Para efectuar su operación esta unidad requiere que alguien le diga lo que debe hacer, ese alguien es la Unidad de Control (UC).

Para el manejo de los datos la ALU utiliza pequeñas unidades de almacenamiento llamados registros, donde también puede guardar resultados.

Existen varios tipos de registros:

El *contador de programa* es un registro que contiene la dirección de la próxima instrucción a ser ejecutada.

Registro de instrucción. Unidad donde se interpreta o traduce la instrucción a ser ejecutada, por lo tanto contiene la instrucción actual.

Registros de trabajo. Son aquellos que son utilizados para llevar a cabo la traducción o la interpretación de la instrucción.

Registros generales. Utilizados por el programador como unidades de almacenamiento auxiliares, donde básicamente pueden almacenarse datos y direcciones.

La **UC** es la parte administrativa de la computadora, ésta se encarga de decir a los demás componentes, cómo y cuándo deben efectuar las operaciones. La UC indica a la ALU qué operación va a realizar, de dónde va a tomar los datos, para lo cual interpreta conjuntos de instrucciones que recibe codificadas en binario desde la memoria.

Las instrucciones poseen dos tipos de información, el *código* de operación y la *dirección* de los operandos.

El primero indica lo que se va a realizar y el segundo indica sobre quién o qué se va a efectuar la operación.

La **memoria** es la unidad de almacenamiento de la computadora, esta es una secuencia ordenada de lugares de almacenamiento denominados localidades, en la memoria se almacenan los programas, es decir, los conjuntos de instrucciones que la UC debe interpretar y los datos sobre los cuales se trabajará, es decir, lo que serán procesados. La memoria se divide en:

1. *Memoria interna, principal o primaria.* Es la memoria que contiene la computadora.

2. *Memoria secundaria o externa.* Se maneja en discos flexibles, cintas magnéticas y en discos duros básicamente.

La *memoria interna* es usualmente un recurso escaso pero muy veloz.

La *memoria externa* es mucho más grande en almacenamiento que la interna; pero en tiempo es mucho más lenta.

La capacidad de memoria es un tamaño que generalmente es medido en Mbytes.

La memoria interna se divide en dos grupos:

1. *Volátil o RAM* (Random Access Memory). Memoria de acceso aleatorio, en ella se puede escribir o leer, pero al apagar la máquina la información se pierde.
2. *No volátil o ROM* (Read Only Memory). Es de solo lectura, no se puede escribir en ella, la información se conserva.

La memoria interna siendo volátil o no se divide en algo que se conoce por palabras. Una palabra puede definirse como una unidad de almacenamiento de la memoria interna y esta puede estar formada por 2, 4, u 8 bytes.

Una localidad es un byte de memoria, un agrupamiento de 8 bits.

Dispositivos periféricos. Son medios de comunicación con la computadora. Se dividen en dispositivos de entrada y dispositivos de salida.

Los dispositivos de entrada permiten al usuario proporcionar datos a la computadora entre lo que se encuentran programas que serán procesados. Por ejemplo, teclado, unidad de discos flexibles, scanner, etc.

Los dispositivos de salida permiten al usuario conocer resultados obtenidos a través del procesamiento de programas y datos como por ejemplo, unidad de discos flexibles, pantalla, impresora, etc.

Todas las partes de un computadora están relacionados, ligados por un bus, el cual puede definirse como un medio de comunicación entre los distintos componentes de la computadora.

Operaciones básicas y funciones.

Memoria:

Operaciones básicas: escribir, leer

Funciones: almacenar programas, almacenar datos, almacenar resultados.

Los programas son conjuntos de instrucciones.

Datos. Información a procesar, la cual se puede utilizar o modificar.

CPU:

Funciones: leer y escribir información de la memoria y a la memoria.

Llevar y traer información a los diferentes componentes de la computadora.

Decodificar o traducir y ejecutar las instrucciones almacenadas en la memoria.

Reconocer y responder a algunas señales externas.

Un *ciclo de máquina* es un conjunto de pasos que se llevan a cabo para ejecutar instrucciones.

Ciclo de máquina:

1. Solicita dato (Dirección)
2. Se manda dato (Instrucción)
3. Se traduce o decodifica el dato
4. Se ejecuta instrucción

Estos pasos se han dividido en tres:

1. *Fetch (alimentar 1 y 2)*
2. *Decode (Decodificar 3)*
3. *Execute (Ejecutar 4)*

Arquitectura 8088

La memoria del procesador 8088 se encuentra fraccionada en lo que se conoce como segmento.

Un *segmento* es una porción de memoria que ocupa 64 kbytes de longitud.

Existen cuatro segmentos:

Segmento de datos (ds). Es la porción de memoria que contiene la información a ser procesada, es decir, los operandos. Este segmento también puede ser definido como el área de datos del procesador ya que cuando éste necesita un operando o necesita guardar un resultado lo toma o lo deposita en algún lugar de este segmento.

Segmento de código (cs). Es la porción que contiene los conjuntos de instrucciones a ser ejecutadas.

Segmento de stack (pila - ss). Generalmente contiene direcciones de retorno a subprogramas.

Subprograma. Es un programa usualmente pequeño que funciona de manera independiente, que puede ser utilizado por otro u otros programas cuando es necesario. Los subprogramas en los lenguajes de programación de alto nivel se conocen como procedimientos y para lenguajes de bajo nivel, como ensamblador, se conocen como subrutinas o subprogramas.

Segmento extra (es). Es direccionado a través del registro es y contiene al igual que el registro de datos información para procesar, los datos principalmente son cadenas. Es un segmento para usos especiales.

Conjunto de Registros para el procesador 8088

La offset es un desplazamiento relativo al inicio del segmento.

Base (cs, ds, ss, es)

<u>+ offset</u>

Dirección efectiva

Registros.

Existen 14 registros en total. 4 segmentados, 4 de propósito general, 2 registros apuntadores, 2 registros índica y 1 registro de banderas y el registro apuntador de instrucción.

Registros segmentados.

DS. Contiene la dirección inicial del segmento de datos.

SS. Contiene la dirección inicial del segmento de pila.

ES. Contiene la dirección inicial del segmento extra.

CS. Contiene la dirección inicial del segmento de código, por lo que puede ser visto como el registro base dentro del segmento de código.

Registros de propósito general.

Son registros de 16 bits.

AX. Llamado acumulador principal, es utilizado en todas las operaciones de entrada y salida, en algunas operaciones de cadenas y en operaciones aritméticas.

BX. Llamado registro base, es utilizado como apuntador o índice para el manejo de datos.

CX. Llamado registro contador, es utilizado para controlar el número de veces que un ciclo debe repetirse, también utilizado en operaciones aritméticas y en corrimientos (movimientos de bits).

DX. Llamado registro de datos, utilizado en algunas operaciones de entrada y salida y en operaciones aritméticas que requieren de 16 bits para el manejo de datos.

Registro apuntadores.

SP y BP, ambos utilizados para el manejo de información dentro del stack.

El registro SP es el offset dentro del stack.

Registros indexados o índice.

SI (Indice fuente). Utilizado para manejar datos dentro del segmento de datos o para manejar información.

DI (Indice Destino). Utilizado para manejar información dentro del segmento extra.

Registro de banderas.

A este registro se le conoce también como registro de estado. Consta de 16 bits y únicamente 9 de ellos contiene información, indican el estado de la máquina y el estado de la ejecución de los programas.

Seis de ellas (C, P, A, Z, S, O) son utilizadas para verificar o indicar una condición producida por alguna instrucción, y tres de ellas son utilizadas para el control de algunas operaciones (D, I, T).

C *Carry*. Indica si hubo carry en operaciones aritméticas. C=1 si existe acarreo, 0 sino existe.

P *Paridad*. Indica la paridad de un dato. P=1 si es par, P=0 sino lo es.

Z *Cero*. Indica si el resultado de una operación aritmética o de comparación es cero. Z=0 el resultado no es cero, Z=1 el resultado es cero.

S *Signo*. Indica el signo del resultado. S=0 si el resultado es positivo, S=1 si el resultado es negativo.

O *Overflow*. Indica overflow en la magnitud de un dato. O=0 no existe overflow, O=1 si existe overflow.

A *Carry auxiliar*. Contiene el acarreo del bit 3 al bit 4 en un operando de 8 bits. A=1 si existe carry. A=0 no existe carry.

D *Dirección*. Es utilizada en operaciones de cadena, indica la dirección que el procesador debe seguir para tomar la información.

I *Interrupción*. Indica al procesador si las interrupciones deben ser atendidas o no.

T *Trap*. Indica al procesador que debe ejecutar una por una las instrucciones de un programa (procesamiento de un solo paso).

Unidades lógicas del procesador 8088.

El procesador 8088 contiene dos unidades lógicas: *la unidad de ejecución y la unidad de interfase de bus.* Cada una de bus. Cada una de ellas tiene una función específica y contiene registros del procesador.

La unidad de ejecución, ejecuta las instrucciones que le proporcione la unidad interfase de bus (BIU), con lo cual debe procesar los datos que también esta le proporciona, la BIU es la unidad lógica más activa, esta tiene tres funciones importantes.

La primera es controlar los buses que transfieren información a la EU, a la memoria y a los dispositivos externos de entrada y salida.

La segunda es el direccionamiento de memoria a través de los registros de segmento, los cuales pueden direccionar hasta un millón de bytes de memoria.

Para poder direccionar la memoria el procesador utiliza 16 bits. Así que el número máximo de bytes que puede direccionar es 2^{16}=65536, es decir, 64 Kb de memoria; pero se requiere de más k de memoria, entonces el procesador para ampliar el direccionamiento de memoria utiliza 20 bits, de esta manera maneja desde 0000 hasta fffff , el procesador suma la base del segmento y el offset para obtener una dirección absoluta, pero recorre el offset 4 posiciones a la derecha y los suma, así se obtienen 5 dígitos con 20 bits, de otra forma serían 4 dígitos con 16 bits.

Ejemplo: A098

```
+   8005
  A8985
```

La tercera función es acceder las instrucciones, esto es, tomas las instrucciones de la memoria, llevarlas a la cola de instrucción.

La cola de instrucción está formada de 4 bytes con los cuales es posible que la BIU tome las instrucciones antes de tiempo (prefetch) para que cuando la EU solicite la instrucción esta esté lista para ser ejecutada.

Modos de direccionamiento y cálculo de la dirección efectiva.

Los modos de direccionamiento son medios que facilitan la tarea de programación a la vez que permiten el acceso a los datos y a los cuerpos de entrada y salida de una manera natural y eficiente, su modo de direccionamiento permite obtener la dirección efectiva, que es aquella información donde se encuentran los datos e información requerida para la ejecución de alguna instrucción. Existen 7 modos de direccionamiento para el procesador 8088.

1. *Direccionamiento de registro*. En este direccionamiento sus operandos se encuentran almacenados en cualquiera de los registros de propósito general o en los registros de segmento.

Ejemplo: add ax,bx

2. *Direccionamiento inmediato*. En este direccionamiento el operando se encuentra almacenado inmediatamente después de la instrucción por lo cual no se necesita calcular la dirección efectiva.

Ejemplo: cmp ah,6

3. *Direccionamiento directo*. En este direccionamiento la dirección del operando se encuentra contenida en la instrucción, esta dirección es sumada a la base del segmento de datos para obtener la dirección efectiva.

Ejemplo: mov cx, [100]

4. *Direccionamiento indirecto*. Se utilizan los registros SI, DI o DX para proporcionar de manera indirecta la dirección del operando. Esta dirección es tomada de cualquiera de los registros ya mencionados y es sumada a la base del segmento de datos para obtener la dirección efectiva sel operando.

Ejemplo: mov bx, [si]

5. *Direccionamiento de datos*. En este caso se establece una base a través del registro BX o del registro BP, a partir de la cual se tomarán los datos del segmento de datos o del segmento de stack. El contenido de cualquiera de los dos registros es una

dirección que será sumada a un desplazamiento contenido en la instrucción y al registro que apunta a la dirección inicial del segmento correspondiente. Si se utiliza el registro BX se tomará como base del segmento el registro DS. Si se utiliza el registro BP la base del segmento será el registro SS.

Ejemplo: add ax, [bx+20]

6. *Direccionamiento indexado*. En este direccionamiento se utilizan los registros de índice SI o DI para calcular la dirección efectiva de un operando a través de la suma del registro índice con el desplazamiento contenido en la instrucción, aquí se involucra el registro que apunta a la dirección inicial del segmento de datos.

Ejemplo: add bx, [si+34]

7. *Direccionamiento de base indexada o indexado con base*. En este direccionamiento se utiliza el registro de base BX y el registro de índice SI para el cálculo de la dirección efectiva a través de la suma con un desplazamiento contenido en la instrucción y con la dirección inicial del segmento de datos.

Ejemplo: add dx, [bx+si+10]

Depurador.

Es un programa que sirve para localizar errores a través de un cierto número de técnicas interactivas, es decir, a través de la interacción con el usuario.Además de ser una valiosa herramienta de depuración, un depurador es una valiosa herramienta de aprendizaje utilizada para conocer paso a paso la ejecución de cada instrucción, para llevar a cabo la interacción con el usuario, el depurador cuenta con comandos que son órdenes para el programa, existe un depurador específico para cada lenguaje, para el ensamblador 8088 el programa depurador se llama debug, el cual cuenta con 18 comandos que incluyen manejo de memoria de registros y de archivos, para utilizar el programa debug hay que llamarlo por su nombre y oprimir enter.

Comandos del debug.

Comando A (Assemble). Se encarga de ensamblar mnemónicos 8086, 8087 y 8088, directamente en la memoria, esto es, permite meter instrucciones a partir de una dirección especificada ensamblándolas inmediatamente para ser ejecutadas. La sintaxis es:

A[dirección]

Ejemplo:

```
-A 100
dirección cs:100 mov ah, 0
dirección cs:102 add ah,bh
dirección cs:104
```

Para terminar de ensamblar solo dar enter y se retornará al prompt esperando una nueva orden.

El comando A verifica los errores de sintaxis línea a línea, si existiera un error en una línea no le permitirá al usuario escribir la siguiente línea sin haber corregido la anterior.

Ejemplo:

```
-A 200
dirección cs:200 mov 5,ah
                     ^Error
dirección cs:200
```

Comando D (Dump). Este comando muestra el contenido de la memoria del rango especificado en el comando, o bien muestra 128 bytes a partir de la dirección inicial especificada en el comando.

Sintaxis:

D [rango]

` donde rango puede ser una sola dirección o dirección inicial, dirección final

Ejemplo:

Código hexadecimal Carácter asociado

```
-d 100  Solo usando una dirección inicial
1074:0100  F8 8B CF 81 E9 82 00 26-88 0E 80 00 C3 8B 1E 92   .......&........
1074:0110  DE BE 1A D4 BA FF FF B8-00 AE CD 2F 34 00 63 10   ...........4.c.
1074:0120  DB E2 0A C0 74 09 56 57-E8 2A 21 5F 5E 73 0A B9   ....t.VW.*!_^s..
1074:0130  04 01 FC 56 57 F3 A4 5F-5E C3 50 56 33 C9 33 DB   ...VW.._^.PV3.3.
1074:0140  AC E8 5F 23 74 19 3C 0D-74 15 F6 C7 20 75 06 3A   .._#t.<.t... u.:
1074:0150  06 0C D3 74 0A 41 3C 22-75 E6 80 F7 20 EB E1 5E   ...t.A<"u... ..^
1074:0160  58 C3 A1 E1 D7 8B 36 E3-D7 C6 06 25 D9 00 C6 06   X.....6....%....
1074:0170  21 D9 00 8B 36 E3 D7 8B-0E E1 D7 8B D6 E3 42 51   !...6.........BQ
-d 100,120  Especificando un rango
1074:0100  F8 8B CF 81 E9 82 00 26-88 0E 80 00 C3 8B 1E 92   .......&........
1074:0110  DE BE 1A D4 BA FF FF B8-00 AE CD 2F 34 00 63 10   ...........4.c.
1074:0120  DB                                                .
-
```

Comando E (Enter). Muestra el contenido de la memoria permitiendo la modificación de dicho contenido. La memoria es modificada únicamente en la porción correspondiente al segmento de datos.

Sintaxis:

E Dirección [Lista de datos] Esta última puede omitirse

Uso:

Cuando se le proporciona únicamente la dirección muestra el contenido de la memoria en esa dirección y espera cuatro posibles opciones:

1) Que el usuario oprima la barra espaciadora con lo cual mostrará el contenido de la siguiente localidad.

Ejemplo:

```
-e 100
1074:0100  F8.  8B.  CF.  Se oprimió dos veces la barra espaciadora.
```

2) Que el usuario oprima la tecla correspondiente al guión, en cuyo caso mostrará el contenido de la localidad anterior.

Ejemplo:

```
-e 100
1074:0100  F8.  8B.  CF.-
1074:0101  8B.
```

3) Si el usuario oprime enter, saldrá del comando

Ejemplo:

```
-e 100
```

1074:0100 F8. 8B. CF.

-

4) Modificar la información, esto es, dar un nuevo dato y oprimir enter.

Ejemplo:

-e 100

1074:0100 F8.54 8B.45 En este caso en la dirección 100 en lugar de tener un F8 ahora se tendrá un 54 y en la siguiente dirección (101) en lugar de tener un 8B se tendrá ahora un 45. Para verificar esto se debe utilizar el comando D con la dirección 100.

Otra forma de usar el comando es además de dar la dirección, dar una cadena encerrada entre comillas (“”). Esta información será almacenada a partir de la dirección dada en el segmento de datos.

Ejemplo:

-e 100 "Lenguaje ensamblador"

-

Ahora verificamos que realmente tenemos esta cadena en el segmento de datos a partir de la dirección 100, para ello utilizamos el comando D.

-d 100,130
1074:0100 4C 65 6E 67 75 61 6A 65-20 65 6E 73 61 6D 62 6C Lenguaje ensambl
1074:0110 61 64 6F 72 BA FF FF B8-00 AE CD 2F 34 00 63 10 ador...../4.c.
1074:0120 DB E2 0A C0 74 09 56 57-E8 2A 21 5F 5E 73 0A B9 ....t.VW.*!_^s..
1074:0130 04 .
-

Comando R (Register). Muestra el contenido de los registros y permite modificarlos.

Sintaxis:

R [nombre del registro]

Ejemplo:

-r En este caso solo muestra el contenido de los registros
AX=0000 **BX=0000** CX=0000 DX=0000 SP=FFEE BP=0000 SI=0000 DI=0000
DS=1074 ES=1074 SS=1074 CS=1074 IP=0100 NV UP EI PL NZ NA PO NC
1074:0100 4C DEC SP
-
-r bx
BX 0000
:**0**
-r ds
DS 1074 Este es el valor anterior del registro ds
:**1000** Aquí estamos cambiando el valor de los registros bx y ds.
-r
AX=0000 **BX=0000** CX=0000 DX=0000 SP=FFEE BP=0000 SI=0000 DI=0000
DS=1000 ES=1074 SS=1074 CS=1074 IP=0100 NV UP EI PL NZ NA PO NC
1074:0100 4C DEC SP Banderas
- Aquí podemos comprobar que sí se realizaron nuestros cambios

Las banderas que podemos visualizar con el comando r son 8, éstas están representadas por 2 caracteres y tienen dos estados. Aquí no aparece la bandera de Trap.

Bandera	Encendido	Apagado
OF (Overflow)	OV	NV
DF (Direction)	DN (Decremento)	UP (Incremento)
IF (Interrupt)	EI (Habilitado)	DI (Deshabilitado)
SF (Sign)	NG (Negativo)	PL (Positivo)
ZF (Zero)	ZR	NZ
AF (Auxiliary Carry)	AC	NA
PF (Parity)	PE (Par)	PO (Impar)
CF (Carry)	CY	NC

Comando T (Trace). Permite ejecutar una a una las instrucciones de un programa, a partir de la dirección especificada, o de la dirección contenida en el registro IP.

Sintaxis:

T[=dirección] [numero]

Si se omite la dirección el depurador ejecuta la instrucción cuya dirección esta contenida en el registro IP.

Si no se omite el número, este comando ejecutará número de instrucciones a partir de la dirección dada o contenida en el registro IP.

Si se omite numero, solo se ejecuta una sola instrucción.

Ejemplo:

-t=100 2 Se ejecutaron 2 instrucciones a partir de la dirección 100

```
AX=0000 BX=0000 CX=0000 DX=0000 SP=FFE5 BP=0000 SI=0004 DI=0000
DS=1000 ES=1074 SS=1074 CS=1074 IP=0101   NV UP EI NG NZ NA PO NC
1074:0101 65          DB      65

AX=0000 BX=0000 CX=0000 DX=0000 SP=FFE5 BP=0000 SI=0005 DI=0000
DS=1000 ES=1074 SS=1074 CS=1074 IP=0167   NV UP EI NG NZ NA PO NC
1074:0167 E3D7        JCXZ    0140  Esta es la siguiente instrucción
-           Esta es la dirección de la siguiente instrucción a ejecutarse
-t Se ejecuta una sola instrucción

AX=0000 BX=0000 CX=0000 DX=0000 SP=FFE5 BP=0000 SI=0005 DI=0000
DS=1000 ES=1074 SS=1074 CS=1074 IP=0140   NV UP EI NG NZ NA PO NC
1074:0140 AC          LODSB
-
```

Comando U. Permite ver el contenido del segmento de código (el conjunto de instrucciones) a partir de la dirección especificada y se esta se omite, a partir de la dirección contenida en el registro IP.

Sintaxis;

U [dirección inicial, dirección final]

Si se omite la dirección final se muestran 1f bytes del segmento de código, o menos si es que alguna instrucción sobrepasa los 1f bytes.

Ejemplo:

-u 100,104

```
1074:0100 4C          DEC     SP
1074:0101 65          DB      65
1074:0102 6E          DB      6E
1074:0103 67          DB      67
1074:0104 7561        JNZ     0167
-r
AX=0000  BX=0000  CX=0000  DX=0000  SP=FFE5  BP=0000  SI=0005
DI=0000
DS=1000  ES=1074  SS=1074  CS=1074  IP=0140  NV UP EI NG NZ NA PO
NC
1074:0140 AC          LODSB   El registro IP contiene 140
-u  Se despliega a partir de la dirección 140
1074:0140 AC          LODSB
1074:0141 E85F23      CALL    24A3
1074:0144 7419        JZ      015F
1074:0146 3C0D        CMP     AL,0D
1074:0148 7415        JZ      015F
1074:014A F6C720      TEST    BH,20
1074:014D 7506        JNZ     0155
1074:014F 3A060CD3    CMP     AL,[D30C]
1074:0153 740A        JZ      015F
1074:0155 41          INC     CX
1074:0156 3C22        CMP     AL,22
1074:0158 75E6        JNZ     0140
1074:015A 80F720      XOR     BH,20
1074:015D EBE1        JMP     0140
1074:015F 5E          POP     SI
-u300  Se despliegan 1f bytes a partir de la dirección 300
1074:0300 00CD        ADD     CH,CL
1074:0302 217303      AND     [BP+DI+03],SI
1074:0305 E942FF      JMP     024A
1074:0308 8BD8        MOV     BX,AX
1074:030A B0FF        MOV     AL,FF
1074:030C 864718      XCHG    AL,[BX+18]
1074:030F A21900      MOV     [0019],AL
1074:0312 C3          RET
1074:0313 50          PUSH    AX
1074:0314 33C9        XOR     CX,CX
1074:0316 FC          CLD
1074:0317 AC          LODSB
1074:0318 41          INC     CX
1074:0319 0AC0        OR      AL,AL
1074:031B 75FA        JNZ     0317
1074:031D 2BF1        SUB     SI,CX
1074:031F 58          POP     AX
-Direccion  Código    Instrucciones
```

Comando Q (Quit). Nos permite salir del depurador.
Sintaxis:
Q
Comando G. Permite ejecutar un conjunto de instrucciones.
Sintaxis:
G=dir_inicial, dir_final
Comando w. Permite guardar un conjunto de instrucciones a disco.
Sintaxis:
w dir_inicial
Antes hay que colocar el número de bytes de que consta el programa en los registrosBXCX.

Conjunto de instrucciones

Sintaxis, Mnemónicos, Operandos
Mnemónico. Nombre de la instrucción, generalmente es la abreviación de la acción que se quiere realizar.
Abreviaciones: (operandos)
reg registro de 8 o 16 bits (no incluidos los registros segmento y los índice)
regp1 registro de propósito general de 16 bits
regseg registro segmento
regind registro índice
mem localidad de memoria [dir], byte [reg], word [reg]
dato operando de 8 o 16 bits
dir dirección de 16 bits
con contador que puede ser 1, cl o cx
[ ] contenido de

Instrucciones de transferencia
MOV (move). Realiza el movimiento (copia) de la información.
Sintaxis: MOV destino, origen
a) MOV reg1, mem/reg2/regind/regseg/dato
b) MOV mem, reg/regseg/regind
c) MOV regseg, mem/regind/regseg/regp1
Ejemplo:
MOV AX,BX
MOV CH, [100]
MOV AX, [100]
MOV [300], BX
MOV [SI], BX
MOV DS, [AX]

MOV DS, BX

LEA. Carga un registro de 16 bits con una dirección específica.
Sintaxis: LEA reg, dir
Ejemplo: LEA BX, [100] BX←100
MOV AX, 128 es equivalente a LEA AX, [128]
XCHG. Intercambia el contenido de la información.
Sintaxis:
a) XCHG reg/regind, reg/mem/regind
b) XCHG mem, reg/regind
Ejemplo:
XCHG AX, CX
XCHG [100], SI

PUSH. Coloca en el tope del stack el contenido de un registro de 16 bits.
Sintaxis: PUSH regp1/regseg/mem/regind
Ejemplo:
PUSH AX
PUSH CS
PUSH [SI]

POP. Coloca en algún registro de 16 bits el contenido del stack.
Sintaxis: POP regp1/regseg/regind/mem
Ejemplo:
POP AX
POP SI
POP [100]

LAHF. Coloca las banderas del orden bajo en el registro AH.
Sintaxis: LAHF
SAHF. Coloca el contenido de AH en la parte baja del registro de banderas.
Sintaxis: SAHF
PUSHF. Coloca el contenido del registro de banderas en el stack (primero la parte alta y después la parte baja).
Sintaxis: PUSHF
POPF. Coloca el contenido del stack en el registro de banderas (el primer byte del stack en la parte baja y el Segundo byte en la parte alta).
Sintaxis: POPF
Instrucciones aritméticas.
ADD. Realiza la suma entre dos operandos.
Sintaxis:
a) ADD reg/regind, reg/mem/regind/dato
b) ADD mem, reg/regind
Ejemplo:
ADD AX, BX
ADD [100], AH

ADD SI, [BX]

ADC. Realiza la suma entre dos operandos y el carry, la sintaxis es equivalente a la del ADD. (ADD op1+op2+cy)

Ejemplo:
ADC BH, F0

SUB. Realiza la resta entre dos operandos, la sintaxis es equivalente a la del ADD.

Ejemplo:
SUB AX, BX
SUB [BX], CX

SBB. Realiza la resta entre dos operandos y el carry, la sintaxis es equivalente a la del ADD. (op1-op2-cy).

Ejemplo:
SBB BH, CL
SBB CX, [SI]

INC. Incrementa en 1 un operando.
Sintaxis: INC reg/regind/mem
Ejemplo:
INC CX
INC SI
INC BYTE[100]
INC WORD[200]

DEC. Decrementa en uno el operando, la sintaxis es equivalente a la de la instrucción INC.

Ejemplo:
DEC WORD[200]
DEC BX

NEG. Obtiene el complemento a 2 del operando.
Sintaxis: NEG reg/regind/mem
Ejemplo:
NEG AX
NEG BYTE[100]

MUL. Multiplica dos datos, esta multiplicación es sun signo.
Sintaxis: MUL reg/regind/mem
La multiplicación se lleva a cabo de la siguiente forma:
1) AX ← AL * reg (de 8 bits)
2) DXAX ← AX * reg (de 16 bits)

Ejemplo:
MUL BH
MUL WORD[SI]
MUL CX

IMUL. Multiplica dos datos con signo, la sintaxis es equivalente a la del MUL.
Ejemplo:
IMUL DL
IMUL BYTE[200]
Instrucciones lógicas.
DIV. Divide un operando que esta en AX o DX:AX entre su único argumento.
Sintaxis:
a) DIV reg_8bits AL<- AX/reg_8bits y AH<- residuo
b) DIV reg_16bits AX<-DX:AX/reg_16bits y DX <-resto
AND. Realiza la operación AND entre dos operandos.
Sintaxis:
a) AND reg/regind, reg/mem/regind/dato
c) AND mem, reg/regind
Ejemplo:
AND AX, [SI]
AND DL, BH
AND CX, 1

OR. Realiza la operación OR entre dos operandos, su sintaxis es equivalente a la de la instrucción AND.
Ejemplo:
OR AX, BX
OR SI, [DI]

NOT. Niega el valor del operando. Obtiene el complemento a 1 del operando.
Sintaxis: NOT reg/regind/mem
Ejemplo:
NOT AX
NOT WORD[SI]
XOR. Obtiene el or exclusivo de dos operandos (valores iguales da 0, valores diferentes da 1), su sintaxis es equivalente a la del AND.
Ejemplo:
XOR AH, CL
XOR [100], SI
Instrucciones de comparación y salto.
CMP. Realiza la comparación entre dos operandos, esta instrucción puede modificar el registro de banderas.
Sintaxis:
a) CMP reg/mem/regind, reg/dato
b) CMP mem, reg/dato/regind
La instrucción CMP realiza una resta del primer operando menos el segundo operando, alterando el valor de las banderas.
Ejemplo:
CMP BYTE[100], 6
CMP AH, 7 (ah – 7)

CMP [CX], BX

JMP. Realiza la transferencia del programa, esto es, cambia la secuencia de ejecución. Al registro IP se le asigna la dirección del argumento del JMP.

Sintaxis: JMP dir/regp1/regind/mem

Ejemplo:

JMP 120

JMP [BX]

JMP SI

JA o JNBE. Salta si la bandera del carry es uno.

JBE o JNA. Salta si la bandera del carry o del cero tienen valor 1.

JCXZ. Salta si el registro CX es cero.

JE o JZ. Salta si la bandera del cero es 1.

JG o JNLE. Salta si la bandera de cero es cero y la bandera de signo es igual a la bandera del overflow. Salta si es mayor o si no es menor o igual.

JGE o JNL. Salta si la bandera del signo es igual a la del overflow. Salta si es mayor o igual.

JL o JNGE. Salta si la bandera del signo es diferente a la del overflow. Salta si es menor.

JLE o JNG. Salta si la bandera del cero es uno o si la bandera del signo es diferente a la del overflow. Salta si es menor o igual.

JNE o JNZ. Salta si la bandera de cero es igual a cero. Salta si no es igual.

JNO. Salta si la bandera del overflow es cero.

JNP o JPO. Salta si la bandera de paridad es cero.

JNS. Salta si la bandera de signo es cero.

JO. Salta si la bandera del overflow es uno.

JP o JPE. Salta si la bandera de paridad es uno.

JS. Salta si la bandera de signo es uno. (negativo)

Nota: Todas las instrucciones de salto tienen su sintaxis equivalente a la de JMP.

Loop. Repite un bloque de instrucciones cx veces, por lo tanto es necesario un valor inicial en cx. En cada iteración LOOP en forma automática disminuye 1 de cx. Si cx es cero termina el ciclo. Al ejecutarse la instrucción LOOP se decrementa cx y se verifica si ya llego a cero, si es asi termina, sino continua el ciclo.

Sintaxis. LOOP dirección

Ejemplo:

```
        mov ax,0
        mov cx,10
Ciclo:  inc ax
        loop ciclo
```

Procedimientos (subrutinas)

Definición. Conjunto de instrucciones que pueden ser ejecutadas desde otro programa.

En el 8088/86 se tienen dos clases de subrutinas:

Cercanas (near). Son aquellas que se encuentran definidas en el mismo segmento que el programa que los llamó.

Lejanas(far). Son aquellas que se encuentran definidas en un segmento distinto al programa que las llama. Por lo tanto existen dos tipos de llamados y dos tipos de regreso.

CALL NEAR y CALL FAR son llamados

RET NEAR y RET FAR son retornos

Interrupciones

Definición. Una interrupción es el rompimiento en la secuencia de un programa para ejecutar un programa especial llamando una rutina de servicio cuya característica principal es que al finalizar regresa al punto donde se interrumpió el programa.

Dentro de una computadora existen dos clases de interrupciones:

Interrupciones por software. Son aquellas programadas por el usuario, es decir, el usuario decide cuándo y dónde ejecutarlas; generalmente son utilizadas para realizar entrada y salida.

Interrupciones por hardware. Van a ser aquellas que son provocadas por dispositivos externos al procesador, su característica principal es que no son programados, esto es, que pueden ocurrir en cualquier momento en el programa. Existen dos tipos de esta clase de interrupciones:

Interrupciones por hardware mascarables. Aquellas en las que el usuario decide si quiere o no ser interrumpido.

Interrupciones por hardware no mascarables (NMI). Aquellas que siempre interrumpen al programa.

En el procesador 8088/86 las instrucciones por software se ejecutan con ayuda de las instrucciones INT e IRET. Se tienen 256 interrupciones diferentes. Desde la interrupción 0 hasta la interrupción 255 (FF).

Asociado con el concepto de interrupciones se tiene un área de memoria llamada vector de interrupciones la cual contiene las direcciones de las rutinas de servicio de cada interrupción. Esta área se encuentra en el segmento 0000:0000

Operaciones que realiza la instrucción INT.

1. Salvar el registro de banderas
2. Salvar el cs de la dirección de regreso
3. Salvar el IP de la dirección de regreso
4. Calcula el área donde está la dirección de la rutina de servicio tipo*4 en el vector de interrupciones.
5. Ejecuta la rutina de servicio

Operaciones que realiza la instrucción IRET

1. Desempila dirección de regreso
2. Desempila banderas

Tabla de interrupciones del sistema

Tipo	*Dirección*	*Uso*	*Sistema*
0	0000	División por cero	BIOS
1	0004	Ejecución paso a paso	DEBUG
2	0008	NMI	BIOS
3	000C	Puntos de ruptura	DEBUG
4	0010	Overflow	BIOS

5	0014	Print Screen	BIOS
6-7		No usadas	
8	0020	Timer	BIOS
9	0024	Teclado	BIOS
A-D		No usadas	
E	0038	Disco	BIOS
F	003C	Impresora	BIOS
10	0040	E/S video	BIOS
11	0044	Lista del equipo	BIOS
12	0048	Tamaño de memoria	BIOS
13	004C	E/S disco	BIOS
14	0050	E/S Serial	BIOS
15	0054	E/S casette	BIOS
16	0058	Entrada Teclado	BIOS
17	005C	Salida impresora	BIOS
18	0060	ROM BASIC	BASIC
19	0064	Boot Strap (reset)	BIOS
1A	0068	Fecha y hora	BIOS
1B	006C	Break (Teclado)	BIOS
1C	0070	Int. de timer	BIOS
1D	0074	Tabla del video	BIOS
1E	0078	Tabla de disco	BIOS
1F	007C	Tabla del video	BIOS
20	0080	Termina programa	DOS
21	0084	Funciones	DOS
22	0088	Dir. De regreso	DOS
23	008C	Control-C	DOS
24	0090	Errores críticos	DOS
25	0094	Lectura absoluta del disco	DOS
26	0098	Escritura absoluta del disco	DOS
27	009C	Termina programa (Deja residente)	DOS

Las interrupciones del BIOS siempre están disponibles al usuario (ROM), en cambio las del DOS solo si el sistema se ha cargado en memoria.

Funciones de la interrupción 21h que permiten E/S

Función

01h Lee un carácter del teclado con ECHO. Regresa en AL el ascii dek carácter (lo lee y lo escribe en pantalla)

02h Escribe un carácter en pantalla. Enviar en DL el ascii del carácter que se quiere escribir

03h Lee un carácter de la entrada auxiliar (serial)

04h Escribe un carácter en el dispositivo auxiliar (serial)

05h Escribe un carácter en la impresora. Enviar en DL el carácter a escribir.

06h E/S directa de la consola (teclado + video)

Si al llamar a esta función DL=FFh entonces implica que se quiere una lectura de carácter. Si hubo un carácter presionado ZF=0 y en AL esta el carácter.

Si no hay carácter presionado ZF=1. Si DL<>0FFh entonces escribe dicho carácter en pantalla.
07h Lee un carácter sin eco. Regresa en AL el carácter.
09h Despliega una cadena en pantalla. Enviar DS:DX=dirección de la cadena. Debe ser una cadena de códigos ascii que termina con "$"
0ah Lee un buffer de caracteres. Enviar DS:DX=apuntador al buffer. Mc = máximo número de caracteres a leer. Nc=número de caracteres leídos.
Buffer | mc nc caracteres |
La interrupción 21 nos permite leer del teclado, escribir en video, escribir en impresora, leer y escribir en el dispositivo auxiliary (Puerto serial).

Tipos de Ensambladores

Definición. Un ensamblador es un programa que traduce mnemónicos de un procesador a si correspondiente lenguaje de máquina.
Por la forma en que trabajan existen dos tipos de ensambladores:
Ensambladores de línea. Son aquellos que reciben una sola línea de un programa y la ensambla independientemente del resto del programa. Ejemplo: el comando *a* del debug.
Ensambladores de archivo. Son aquellos que ensamblan todo un programa almacenado en un archivo.
Por el tipo de información que manejan los ensambladores se dividen también en:
Ensambladores propios. (Residentes) Ensamblan programas escritos en lenguaje del procesador con el que trabaja la máquina. Ejemplo MASM
Ensambladores cruzados. (Crossassembler) Ensamblan programas escritos en lenguaje de un procesador diferente al de la computadora de trabajo, pero no puede ejecutarse.
Macroensambladores. Ensambladores propios o cruzados que permiten definición y expansión de MACROS.
Facilidades de los ensambladores de archivo.

1. Nos permite definir etiquetas (nombre que nos marca una dirección importante)
2. Nos permite reservar memoria con una etiqueta asignada.
3. Nos permite ensamblar programas almacenaos en archivos.
4. Nos permite definir constantes.
5. Nos permite dar números en diferentes bases.
6. Nos permite evaluar expresiones aritméticas. Ejemplo: mov ax, 30+2

MASM (ensamblador del 8088/86)

Recibe archivos ascii editados en cualquier editor que contenga programas en lenguaje ensamblador. Tales archivos deben tener extensión ASM y con una forma específica.
Algoritmo → Codificación en ensamblador → Editar → nov.asm → ensamblar (masm archivo.asm;) → código de máquina (archivo.obj) → ligar→ archivo ejecutable (archivo.exe) → ejecutar
El archivo objeto no se puede ejecutar porque no tiene la dirección de memoria donde se ejecutará y será ligado.

Definición. Una pseudoinstrucción es una instrucción para el programa ensamblador, esto es, que solo se ejecuta en el momento de ensamblar, además no generar código.

Pseudoinstrucciones para definir segmentos.

SEGMENT. Define el inicio de un nuevo segmento, el formato es:

Nombre SEGMENT alineación combinación clase

Los parámetros del SEGMENT con información para el ligador.

Alineación. Nos define la dirección a partir de donde puede colocarse el segmento.

PARA (PARAGRAFH). La dirección del segmento es un múltiplo de 16.

PAGE. La dirección inicial del segmento es donde empieza una página (múltiplo de 100h).

WORD. La dirección inicial del segmento es una dirección par.

BYTE. El segmento inicial donde sea.

Si no se especifica la alineación toma por default una alineación tipo PARA y será un múltiplo de 16.

Combinación. Define la forma en que el segmento puede combinarse con otros segmentos que tengan el mismo nombre y la misma clase.

OMITIRLA. Segmento privado, es decir, no puede combinarse.

STACK. Segmento para usarse con el SS:SP

PUBLIC. Este segmento puede unirse con todos los segmentos del mismo nombre y la misma clase para formar uno solo.

COMMON. Todos los segmentos del mismo nombre y clase se colocan a partir de la misma dirección.

Cuando se tienen 2 segmentos con el mismo nombre y clase y PUBLIC al ligar se unirán en un solo segmento no importando que esten en archivos distintos.

Cuando se usa el COMMON van a utilizar el mismo espacio en memoria y si son de diferente tamaño utilizará el tanto de memoria del mayor.

Clase. Indica el tipo de datos que contiene el segmento.

'DATA' datos

'CODE' código

'STACK' stack

Sin embargo se pueden definir otros. Siempre van entre comillas.

El programa LINK puede ligar varios archivos objeto para crear un ejecutable.

ENDS. Define el final de un segmento. Formato:

Nombre ENDS

Ejemplo:

```
DATOS SEGMENT PARA 'DATA'
.....  Define variables y constantes del programa
DATOS ENDS
CODIGO SEGMENT PARA 'CODE'
.....  Programa principal y rutinas
CODIGO ENDS
```

Pseudoinstrucciones (directivas) para reservar memoria y definir constantes.

DB Sirve para reservar un byte en la memoria con un valor determinado. El formato es:

Etiqueta DB val1[,val2,....,valn]
DW Reserva un dato de 2 bytes (una palabra) con un valor inicial
[etiqueta] DW val1[,val2,....,valn]
DD Reserva un dato de 4 bytes (una doble palabra) con un valor inicial
[etiqueta] DD val1[,val2,....,valn]
DQ Reserva 8 bytes de memoria (cuadruple palabra) con un valor inicial
[etiqueta] DQ val1[,val2,....,valn]
DT Reserva un dato de 10 bytes con un valor inicial
[etiqueta] DT val1[,val2,....,valn]
Nota: vali nos representa una expresión formada por números en cualquiera de las siguientes bases:
XXXB binario
XXXO octal
XXX
XXXD decimal
XXXH hexadecimal
O bien expresiones aritméticas de esos números con los siguientes operadores
+ suma, - resta, - negación aritmética (complemento a 2), * multiplicación y / división.
También pueden ser etiquetas o expresiones aritméticas que involucren etiquetas o bien cadenas de ASCII's limitadas por apóstrofes.
Ejemplo:

```
DATOS SEGMENT PARA 'DATA'
        UNO   DB     08H
        DOS   DW     0F48H, 20º, 10111B
        TRES  DD     (45FH+178H)*2
        CUATRO       DQ     78
        CINCO        DW     DOS, 'B'
        SEIS  DB 'ES UNA CADENA'
DATOS ENDS
```

EQU Permite definir constantes
Etiqueta EQU valor
Ejemplo:

```
        CONSTANTE        EQU   34
```

ORG Define el desplazamiento inicial para ensamblar las siguientes líneas en el texto

```
        ORG   valor
```

Pseudoinstrucciones para definir procedimientos .
PROC Define el inicio de una subrutina
Nombre PROC tipo
Si se omite el tipo, por default el procedimiento es de tipo NEAR
ENDP Indica el final de una subrutina
Nombre ENDP
Para compilar un programa con MASM de forma condensada se utiliza:
c:\>masm archivo;

Si no tiene errores el programa entonces se liga para crear el ejecutable:
c:\>link archivo;
sino primero hay que corregir errores.
Estructura básica de un programa en macroensamblador.

```
DATOS SEGMENT PARA 'DATA'
.....
DATOS ENDS
PILA SEGMENT PARA STACK 'STACK'
        DW 100 DUP(0)
PILA ENDS
CODIGO SEGMENT PARA 'CODE'
        ASSUME DS:DATOS, CS:CODIGO, SS:PILA, ES:NOTHING
; PROGRAMA PRINCIPAL
MAIN PROC FAR
        PUSH DS       ; Sirve para que cuando el programa termine
        XOR AX,AX  ; pueda regresar al debug o al sistema operativo
        PUSH AX       ;
        MOV AX,DATOS     ; Actualiza los registros de segmentos de
        MOV DS, AX; datos y extra
        MOV ES, AX ;
        …….. ; código del programa principal
        RET
MAIN ENDP
SUB1 PROC
….. ; código de la primera subrutina
SUB1 ENDP
…….
SUBN PROC
……
SUBN ENDP
CODIGO ENDS
END MAIN
```

1

Nota. El programa principal puede ir al inicio o al final del segmento de código.
DUP Es un operador de MASM que indica que tiene que repetir la pseudoinstrucción n veces con valores iniciales marcados entre paréntesis.
DX n DUP (val1,val2, ..., valm)
Ejemplo:
DB 5 DUP(1,2)
En la memoria encontraríamos lo siguiente:

```
XXX   01
XXX+1  02
XXX+2  01
XXX+3  02
XXX+4  01
```

ASSUME es la pseudoinstrucción que sirve para indicarle al ensamblador cuales segmentos son usados por los registros de segmentos y se pueden calcular correctamente las direcciones de los operandos.

1 Estas instrucciones sirven para indicarle al procesador qué segmentos usará para DATOS y EXTRA, es decir, los únicos registros que se cargan automáticamente son el segmento de código y el de stack.

END Indica al MASM que el ensamble termina

END dir ← dirección del programa principal (una etiqueta)

Ejemplo: Programa que limpia pantalla y escribe una cadena en el renglón12, columna 30.

```
DATOS SEGMENT PARA 'DATA'
        CADENA      DB      'ESTO ES UNA PRUEBA$'
        REN         DB      12
        COL         DB      30
DATOS ENDS

PILA SEGMENT PARA STACK 'STACK'
        DW    100     DUP(0)
PILA ENDS

CODIGO SEGMENT PARA 'CODE'
        ASSUME      DS:CODIGO, DS:DATOS, SS:PILA, ES:NOTHING
        LIMPIA_PANTALLA PROC NEAR
                MOV AX, 0600H
                MOV BH, 71H
                MOV CX, 0000H
                MOV DX, 184FH
                INT 10H
                RET
        LIMPIA_PANTALLA ENDP

POSICIONA_CURSOR PROC NEAR
                MOV AH,02
                MOV BH,00
                MOV DH,REN
                MOV DL, COL
                INT 10H
                RET
        POSICIONA_CURSOR ENDP

LETRERO     PROC NEAR
                MOV AH,09
                LEA DX, CADENA
                INT 21H
                RET
        LETRERO ENDP
```

```
        PRINCIPAL PROC FAR
                PUSH DS
                XOR AX, AX
                PUSH AX
                MOV AX, DATOS
                MOV DS, AX
                MOV ES, AX
                CALL LIMPIA_PANTALLA
                CALL POSICIONA_CURSOR
                CALL LETRERO
                MOV AH,00
                INT 21H
                RET
        PRINCIPAL ENDP
CODIGO ENDS
END PRINCIPAL
```

Operaciones con cadenas de caracteres.

MOVS. Mueve un byte, palabra o palabra doble desde una localidad en memoria direccionada por SI a otra localidad direccionada por DI.

LODS. Carga desde una localidad de memoria direccionada por SI un byte en AL, una palabra en AX o una palabra doble en EAX.

STOS. Almacena el contenido de los registros AL, AX, o EAX en la memoria direccionada por SI.

CMPS. Compara localidades de memoria de un byte, palabra o palabra doble direccionadas por SI, DI.

SCAS. Compara el contenido de AL, AX o EAX con el contenido de una localidad de memoria direccionada por SI.