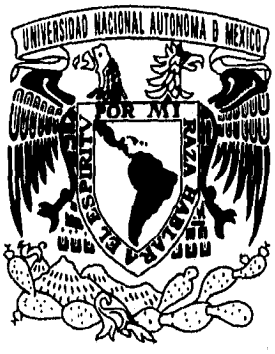
41
2ej



UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

ESCUELA NACIONAL DE ESTUDIOS PROFESIONALES ACATLÁN

"ANÁLISIS Y REFERENCIA SOBRE LA PROGRAMACIÓN EN LENGUAJE ENSAMBLADOR"

TESINA

Que para obtener el título de:
LICENCIADO EN MATEMÁTICAS APLICADAS Y COMPUTACIÓN

Presenta:
ARMIDA JUDITH SALAZAR ARAGÓN

Asesor:
FIS. MAT. JORGE LUIS SUÁREZ MADARIAGA

Naucalpan de Juárez, Estado de México Primavera 1996

TESIS CON
FALLA DE ORIGEN

UNIVERSIDAD NACIONAL AVENOMA DE MEXICO

ESCUELA NACIONAL DE ESTUDIOS PROFESIONALES "ACATLAN"

DIVISION DE MATEMATICAS E INGENIERIA
PROGRAMA DE ACTUARIA Y M.A.C.

SRITA. ARMIDA JUDITH SALAZAR ARAGON
Alumna de la carrera de M.A.C.
P r e s e n t e .

Por acuerdo a su solicitud presentada con fecha 20 de febrero de 1995, me complace notificarle que esta Jefatura tuvo a bien asignarle el siguiente tema de Tesina: "ANALISIS Y REFERENCIA SOBRE-LA PROGRAMACION EN LENGUAJE ENSAMBLADOR", el cual se desarrollará como sigue:

INTRODUCCION
CAP. I Conceptos Fundamentales.
CAP. II Ensambladores.
CAP. III Programación en Lenguaje Ensamblador.
CAP. IV Tópicos Avanzados.
CONCLUSIONES.
BIBLIOGRAFIA.

Asimismo, fué designado como Asesor de Tesina el FIS. MAT. JORGE LUIS SUAREZ MADARIAGA, Profesor de Esta Escuela.

Ruego a usted tomar nota que en cumplimiento de lo especificado-en la Ley de Profesiones, deberá presentar servicio social durante un tiempo mínimo de seis meses como requisito básico para sustentar examen profesional así como de la disposición de la Coordicación de la Administración Escolar en el sentido de que se imprima en lugar visible de los ejemplares de la tesina el título-del trabajo realizado. Esta comunicación deberá imprimirse en el interior de la misma.

E.N.E.P. ACATLAN

A T E [illegible] T E
"POR M[illegible]RA EL ESPIRITU"
Acatlán, d[illegible], junio 6 de 1996.

ACT. [illegible] BECERRA
Jefatura [illegible] de Actuaría
y [illegible]

cg'

A mis queridos padres,
a Toño, Alfredo, Iván, Vladi y
Paquito
con todo mi amor.

Ante la imperdonable posibilidad, involuntaria de mi parte, de omitir el nombre de alguna persona en la lista de quienes de una manera u otra he obtenido mi acervo; opto por otorgar en forma general mi reconocimiento y gratitud a quienes siempre me alimentaron con sus conocimientos, orientación y amistad.

CAPÍTULO III

CAPÍTULO IV

Programar una computadora consiste en especificar una secuencia de tareas para ser ejecutadas por la Unidad Central de Procesamiento (CPU), la cual sólo entiende instrucciones máquina (cadenas de unos y ceros), antes, programar un microprocesador significaba introducir instrucciones en Lenguaje Máquina, hasta que los primeros ensambladores permitieron sustituir el Lenguaje Máquina por Lenguaje Mnemónico (nombre asignado a una instrucción). A menudo los términos de Lenguaje Ensamblador y Lenguaje Máquina son utilizados como sinónimos; sin embargo no es lo mismo, cada sentencia del Lenguaje Ensamblador es traducida a una instrucción en Lenguaje Máquina por el ensamblador.

Para escribir programas en Lenguaje Ensamblador es necesario conocer el conjunto de instrucciones, las reglas de uso y el Hardware en el que se va a ejecutar, ya que un Lenguaje Ensamblador depende de la arquitectura y de la configuración de la computadora.

El propósito de este trabajo es tratar con un enfoque didáctico, analítico y de referencia, la manera de programar en Lenguaje Ensamblador y aportar al estudiante de Matemáticas Aplicadas una referencia de la programación en Lenguaje Ensamblador, de forma tal que estimule su aprendizaje.

Este trabajo esta enfocado al Microprocesador 8088, ya que el conjunto de instrucciones para los diferentes microprocesadores de Intel tienen mucho en común, y la comprensión del funcionamiento de éste puede ser considerada como la base para el aprendizaje de versiones posteriores.

Como requisitos es que los estudiantes tengan la mínima familiaridad con la PC, Sistema Operativo DOS y conocimiento de otros lenguajes.

El Capítulo Uno contiene conceptos de sistemas numéricos (binario, decimal, hexadecimal) y la arquitectura de la computadora. Provee una breve historia de la evolución de los microprocesadores y una descripción del trabajo del 8088 como base.

El Capítulo Dos explica el uso del programa DEBUG de DOS el cual permite ver la memoria, introducir programas, y ejecutarlos paso a paso; introduce a la lógica de las instrucciones y a las interrupciones del DOS y del BIOS

El Capítulo Tres describe el uso del ensamblador MASM de Microsoft para crear programas, los procedimientos para escribir programas, ensamblarlos, ligarlos y ejecutarlos. También se explican los detalles de los programas y de los listados generados en cada paso, los requerimientos para programar en Lenguaje Ensamblador, el uso de un prototipo en general, como una estructura básica para un programa.

El Capítulo Cuatro provee información para el manejo del teclado, la pantalla, el puerto serial, el puerto paralelo. Aunque este trabajo hace énfasis en la programación 16 bits, este Capítulo introduce a algunos conceptos de la programación de 32 bits.

CAPÍTULO I

CONCEPTOS FUNDAMENTALES

CAPÍTULO 1

CONCEPTOS FUNDAMENTALES

Para entender el funcionamiento interno de la computadora es necesario entender algunos conceptos básicos del diseño de las computadoras. En éste capítulo se presentan los fundamentos de los sistemas numéricos, de la arquitectura de la computadora y una historia de la familia de procesadores.

1.1 Sistemas Numéricos

Aunque los seres humanos utilizan aritmética en base 10 (sistema decimal), las computadoras utilizan el sistema en base 2 (binario).

1.1.1 Sistema Numérico Decimal y Binario

Conversión de Decimal a Binario

Un método de convertir del sistema decimal al sistema binario es dividir el número decimal entre 2 repetidamente, el método termina hasta que el cociente es cero, entonces los residuos se escriben en orden inverso para obtener el número binario.

Convertir 25_{10} a binario

Solución

		Cociente	Residuo	
25/2	=	12	1	bit menos significativo
12/2	=	6	0	
6/2	=	3	0	
3/2	=	1	1	
1/2	=	0	1	bit más significativo

$25_{10} = 11001_2$

CONVERSIÓN DE BINARIO A DECIMAL

Para convertir de binario a decimal, es importante entender el concepto del valor asociado con cada posición

8859567_{10}

7×10^0	=	7
6×10^1	=	60
5×10^2	=	500
9×10^3	=	9000
5×10^4	=	50000
8×10^5	=	800000
8×10^6	=	8000000
		8859567

De la misma manera cada posición de un dígito en un número en base 2 tiene un valor asociado a él.

110101_2

				Decimal	Binario
1×2^0	=	1 x 1	=	1	1
0×2^1	=	0 x 2	=	0	00
1×2^2	=	1 x 4	=	4	100
0×2^3	=	0 x 8	=	0	0000
1×2^4	=	1 x 16	=	16	10000
1×2^5	=	1 x 32	=	32	100000
					110101

Sumando el valor de cada bit en un número binario se obtiene su equivalente en decimal.

Convertir 11001_2 a decimal

Solución

Valor	16	8	4	2	1
Dígitos	1	1	0	0	1
Suma	16+	8+	0+	0+	1

$= 25_{10}$

Es posible convertir un número decimal a un número binario asociando el valor de cada dígito binario.

Convertir 39 a binario

Solución

Valor	32	16	8	4	2	1
Dígitos	1	0	1	0	0	1
Suma	32+	0+	8+	0+	0+	1

= 101001

1.1.2 SISTEMA HEXADECIMAL

El Sistema Hexadecimal es usado para representar los números binarios, por ejemplo es mucho más sencillo para el hombre representar una cadena de 1s y 0s como 1000 0101 0001 en su hexadecimal equivalente 851H.

Sistema	Dígitos	
Binario	2	0 1
Decimal	10	0 1 2 3 4 5 6 7 8 9
Hexadecimal	16	0 1 2 3 4 5 6 7 8 9 A B C D E F

Decimal	Binario	Hexadecimal
0	0000	0
1	0001	1
2	0010	2
3	0011	3
4	0100	4
5	0101	5
6	0110	6
7	0111	7
8	1000	8
9	1001	9
10	1010	A
11	1011	B
12	1100	C
13	1101	D
14	1110	E
15	1111	F

CONVERSIÓN ENTRE BINARIO Y HEXADECIMAL

Para representar un número binario en su equivalente en hexadecimal, se empieza a agrupar en 4 bits, desde la derecha, cada grupo equivale a un número hexadecimal

Convertir 100111110101_2 a hexadecimal

Solución

Se agrupa el número en conjunto de 4 bits: 1001 1111 0101

Cada grupo de 4 bits es reemplazado con su equivalente hexadecimal:

1001 1111 0101
9 F 5

$= 9F5_{16}$

Para convertir un número hexadecimal a binario, cada dígito hexadecimal es reemplazado por su equivalente binario

Convertir $F7B1_{16}$ a binario

Solución

1111 0111 1011 0001
F 7 B 1

$= 1111011110110001$

CONVERSIÓN DE DECIMAL A HEXADECIMAL

Se puede hacer de dos maneras:
Un método es primero convertir el número decimal a binario y después a hexadecimal.
Otro método es convertir directamente de decimal a hexadecimal dividiendo repetidamente.

Convertir 45_{10} a Hexadecimal
Solución

		Cociente	Residuo	
45 / 16	=	2	13	(Dígito menos significativo)
2 / 16	=	0	2	(Dígito más significativo)
			$= 2D_{16}$	

Convertir 629_{10} a Hexadecimal
Solución

		Cociente	Residuo	
629 / 16	=	39	5	(Dígito menos significativo)
39 / 16	=	2	7	
2 / 16	=	0	2	(Dígito más significativo)
			$= 275_{16}$	

Convertir 1714_{10} a hexadecimal
Solución

		Cociente	Residuo	
1714 / 16	=	107	2	(Dígito menos significativo)
107 / 16	=	6	11	
6 / 16	=	0	6	(Dígito más significativo)
			$= 6B2_{16}$	

CONVERSIÓN DE HEXADECIMAL A DECIMAL

Se puede hacer de dos maneras:
Un método es primero convertir el número hexadecimal a binario y después a decimal.
Otro método es convertir directamente de hexadecimal a decimal sumando el valor asociado de todos los dígitos.

Convertir los siguientes números hexadecimales a decimales

$6B2_{16}$
Solución

2×16^0	=	2×1	=	2
11×16^1	=	11×16	=	176
6×16^2	=	6×256	=	1536
				1714

$= 1714_{10}$

$9F2D_{16}$
Solución

13×16^0	=	13×1	=	13
2×16^1	=	2×16	=	32
15×16^2	=	15×256	=	3840
9×16^3	=	9×4096	=	36864
				40749

$= 40749_{10}$

1.1.3 Operaciones en Base 10, 2 y 16

En cada base cuando se le suma un uno al dígito más alto, ese dígito se convierte en cero y se acarrea un uno a la siguiente posición. Por ejemplo en decimal 9 + 1 = 0 con un acarreo a la siguiente posición, en binario, 1 + 1 = 0 con un acarreo; similarmente en hexadecimal, F + 1 = 0 con acarreo.

Suma de Números Binarios y Decimales

La suma de números binarios y decimales es un proceso sencillo:

A + B	Acarreo	Suma
0 + 0	0	0
0 + 1	0	1
1 + 0	0	1
1 + 1	1	0

Suma de números binarios y decimales
Solución

	Binario	Decimal
	1101	13
	1001	9
+	10110	22
	101100	44

Las computadoras realizan una suma para implementar la resta de números binarios, es decir para efectuar x - z, toma el complemento a dos de z y lo suma a x.

Complemento a 2

Un número binario negativo contiene un 1 en el primer bit (el más significativo).

+ 3	00000011
+ 2	00000010
+ 1	00000001
0	00000000
- 1	11111111
- 2	11111110
- 3	11111101

Para encontrar el complemento a 2 o el negativo de un número binario es necesario invertir todos los bits y sumar un 1 al resultado. Invertir bits es cambiar 1s por 0s y 0s por 1s.

Complemento a dos de 65
Solución

Número 65	01000001
	10111110
Sumar 1	10111111 (= -65)

+65 -65
Solución

+ 65	01000001
- 65	10111111
00	00000000

SUMA Y RESTA DE NÚMEROS HEXADECIMALES

Suma de números hexadecimales

Se empieza por los dígitos menos significativos, se suman y si el resultado es menor que 16, se escribe en esa posición. Si es más grande que 16, se le resta 16 para obtener el dígito y el acarreo.

Sumar 23D9 + 94BE
Solución

```
        23D9
+       94BE
        B897

        9 + 14      =   23      23 - 16 = 7     Con acarreo al siguiente dígito
        1 + 13 + 11 =   25      25 - 16 = 9     Con acarreo al siguiente dígito
        1 + 3 + 4   =   8
        2 + 9       =   B
```

Resta de números hexadecimales

En la resta de dos números hexadecimales, si el segundo dígito es más grande que el primero, se le suma 16 del dígito anterior.

Restar 59F - 2B8
Solución

```
        59F
-       2B8
        2E7

        8 de 15                 =   7
        11 de 25  (9+16)        =   14      = E
        2 de  4   (5-1)         =   2
```

1.1.4 Sistema Numérico BCD

Los números binarios de 0 al 9 son llamados BCD (Binary Code Decimal)

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

Existen dos términos para los números BCD

BCD empacados
BCD no empacados

BCD empacados

Dos números BCD pueden estar en un sólo byte, uno en los 4 bits altos y otro en los 4 bits bajos.

BCD no empacados

Los 4 bits más bajos representan el número BCD, los otros 4 bits son 0's.

ASCII (hex)	BCD empacado	BCD no empacado
48	0100 1000	0000 0100 0000 1000
57	0101 0111	0000 0101 0000 0111

1.1.5 Código ASCII

Toda la información en las computadoras es representada por la ausencia o presencia de una señal electrónica, por lo que sólo existen dos diferentes valores 1s y 0s. Se utiliza el término bit (binary digit) para referirnos a la unidad mínima de almacenamiento.

En los años 1960's una representación estándar llamada ASCII (American Standard Code for Information Interchange) fue establecida. El código ASCII representa los números del 0 al 9, las letras del alfabeto mayúsculas y minúsculas, códigos de control y signos de puntuación. La ventaja de este sistema es que es utilizado por la mayoría de las computadoras y la información puede ser compartida.

Símbolo	Hexadecimal	Símbolo	Hexadecimal
A	41	a	61
B	42	b	62
C	43	c	63
D	44	d	64
E	45	e	65
F	46	f	66
G	47	g	67
H	48	h	68
I	49	i	69
J	4A	j	6A
K	4B	k	6B
L	4C	l	6C
M	4D	m	6D
N	4E	n	6E
O	4F	o	6F
P	50	p	70
Q	51	q	71
R	52	r	72
S	53	s	73
T	54	t	74
U	55	u	75
V	56	v	76
W	57	w	77
X	58	x	78
Y	59	y	79
Z	5A	z	7A

1.2 ARQUITECTURA DE LA COMPUTADORA

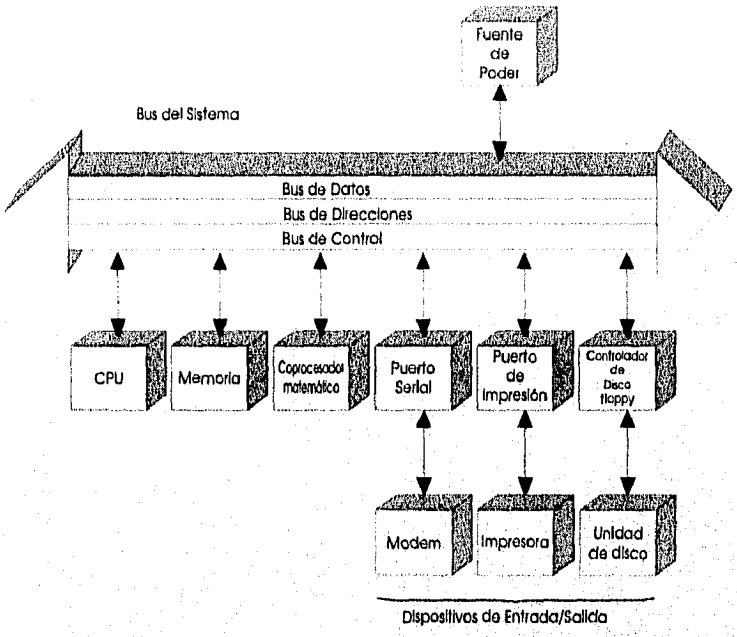
Esta sección es una introducción a la organización y al trabajo interno de las computadoras de manera general.

1.2.1 ORGANIZACIÓN INTERNA DE COMPUTADORAS

El trabajo interno de una computadora puede estar dividido en tres partes:

CPU (Central Processing Unit),
Memoria
Periféricos de E/S

EL CPU ejecuta información grabada en la memoria. Los periféricos de E/S tales como el teclado y el monitor proveen la comunicación con el CPU. El CPU esta conectado a la memoria y a los periféricos a través del bus, el cual lleva información de un lugar a otro.



ESTRUCTURA BÁSICA DE UNA COMPUTADORA

En todas las computadoras existen tres tipos de buses:

bus de direcciones
bus de datos
bus de control

Bus de control

El bus de control se utiliza para proveer las señales de escritura y lectura hacia el periférico para indicar si el CPU esta enviando o requiriendo información.

Bus de datos

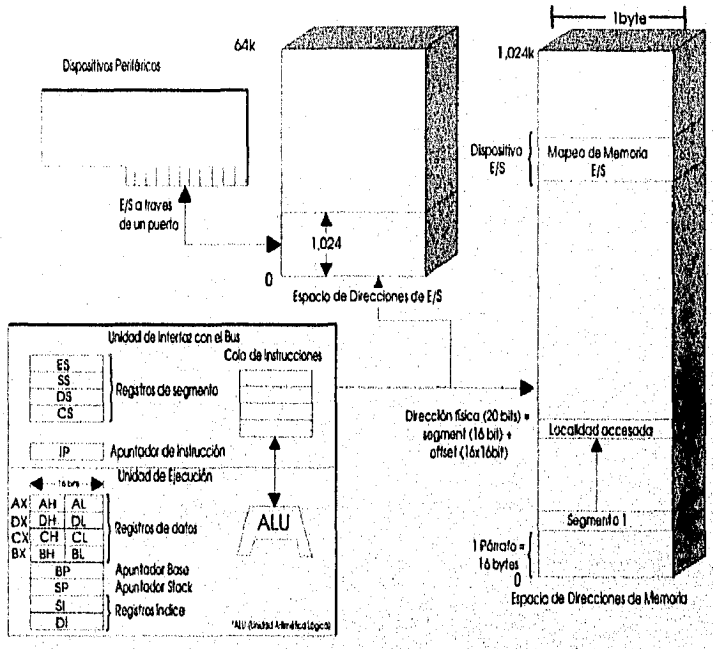
Canal interno a través del cual los datos son enviados hacia y desde el CPU.
El tamaño promedio del bus de datos varia entre 8 y 64 bits.

Bus de direcciones

Canal interno a través del cual se transmiten las direcciones de los datos. El número de cables en el canal de direcciones determina la cantidad de memoria que puede ser direccionable. Por ejemplo, los procesadores 8086/8088 tienen 20 líneas de direcciones y pueden direccionar hasta 1,048,576 bytes

CPU

El 8086 representa la arquitectura base para los microprocesadores de 16 bits de Intel. Aunque cada microprocesador tiene nuevas características, Una vez que se entiende la arquitectura básica de un 8086, es más sencillo programar en 80286, 80386 o 80486.



ARQUITECTURA BÁSICA DE UN MICROPROCESADOR 8086

El Microprocesador 8086 tiene dos unidades básicas:

Unidad de Interfaz del Bus (Bus Interface Unit),
Trae los datos y las instrucciones de la memoria.

Unidad de ejecución (Execute Unit),
Decodifica y ejecuta las instrucciones, y contiene la Unidad Lógica Aritmética (ALU), Unidad de Control y los registros

Estas dos unidades operan en paralelo, la unidad de interface del bus continúa trayendo las instrucciones, mientras el unidad de ejecución procesa la instrucción actual. Existe una área de almacenamiento, conocida como la cola de instrucciones, donde la interface del bus almacena la instrucción recuperada de la memoria. Esta operación paralela mejora la velocidad de ejecución del microprocesador.

Para que el CPU procese la información, los datos deben ser grabados en la memoria RAM (Random Access Memory) o en la memoria ROM (Read Only Memory).

La función de la ROM en las computadoras es proveer información o programas que son esenciales para el funcionamiento de la computadora. La información en la ROM es permanente, no se pierde cuando se apaga la computadora.

La RAM es utilizada para almacenamiento temporal, los datos se pierden cuando la computadora se apaga, por esta razón se le conoce también como memoria volátil.

El CPU utiliza registros para grabar información temporalmente. Los registros pueden ser de 8 bits, 16 bits, 32 bits o 64 bits dependiendo del CPU.

1.2.2 Breve Historia del CPU

En los años 40's, los CPU fueron diseñados utilizando tubos de vacío, eran voluminosos y gastaban mucha energía. En los años 50's los transistores reemplazaron el uso de tubos de vacío en el diseño de las computadoras. En 1959 se inventó el primer circuito integrado, en los años 60's el uso de circuitos integrados en el diseño de tarjetas del CPU se hizo más común, fue hasta los años 70's que un CPU fue puesto en un sólo circuito integrado. El primer CPU en un circuito fue inventado por Intel en 1971. Este CPU es llamado microprocesador. El primer microprocesador, el 4044, tuvo un bus de datos de 4 bits. Los avances en la fabricación de circuitos integrados hicieron posible el diseño de procesadores con bus de datos de 8 bits y bus de direcciones de 16 bits.

CICS y RISC

Hasta los años 80's, todos los CPU's seguían el diseño de CICS (Complex Instruction Set Computer), el cual se refiere a CPU's con un conjunto de instrucciones muy extenso. Después la arquitectura de diseño fue RISC (Reduced Instruction Set Computer), que se refiere a CPU's con un conjunto reducido de instrucciones. La mayoría de los programas utilizan generalmente unas pocas instrucciones, y si se acelera la ejecución de esas instrucciones se mejora el rendimiento. Los chips RISC can más baratos de producir.

Historia de la Familia 80x86

Microprocesadores 8080/8085 y 8086

En 1978, Intel introdujo un microprocesador de 16-bits, llamado 8086.

Microprocesadores 8086 y 8088

El Microprocesador 8086 con un bus de datos de 16 bits internamente y externamente, lo que significa que todos los registros son de 16 bits y el bus de datos es de 16 bits para transferir datos del CPU.

Microprocesador 8088

En 1981

Microprocesadores 80286, 80386 y 80486

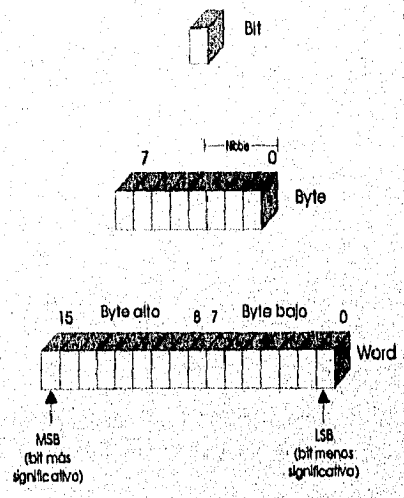
En 1982, Intel introdujo el Microprocesador 80286. Con bus de datos de 16 bits interno y externo; 24 líneas de direcciones, lo cual permiten 16 megabytes de memoria (2^{24}) y la memoria virtual. Este microprocesador puede operar en modo real o modo protegido.

En 1985, Intel introdujo el 80386 (80386DX) con un bus de datos de 32 bits interno y externo, un bus de direcciones de 32 bits, que permite una memoria mayor de 4 gigabytes (2^{32}). La memoria virtual fue incrementada a 64 terabytes (2^{46}).
Después introdujo el 386SX, el cual es internamente idéntico que el bus de 8036 pero tiene bus de datos externo de 16 bits y un bus de direcciones de 24 bits el cual da una capacidad de 16 megabytes de memoria (2^{24}).

En 1989 introdujo 80486 con características adicionales como la memoria cache.

	8080	8085	8086	8088	80286	80386	80486
Año	1974	1976	1978	1979	1982	1985	1989
Memoria física	64 K	64 K	1 M	1 M	16 M	4 G	4 G
Bus de datos interno	8	8	16	16	16	32	32
Bus de datos externo	8	8	16	8	16	32	32
Bus de direcciones	16	16	20	20	24	32	32
Tamaño de datos	8	8	8, 16	8, 16	8, 16	8, 16, 32	8, 16, 32

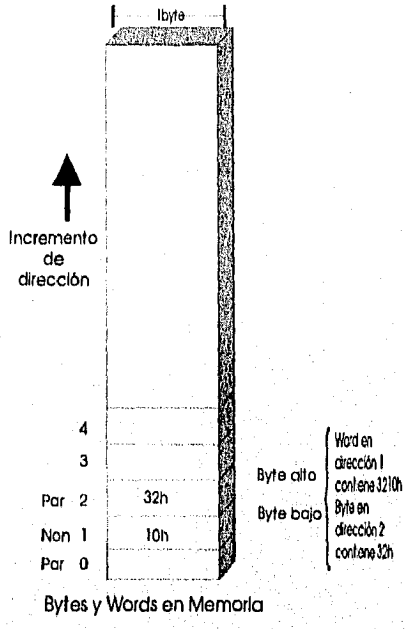
1.2.3 REPRESENTACIÓN DE INFORMACIÓN.



UNIDADES DE ALMACENAMIENTO

Bit	Binary digit
Byte	8 bits
Nible	1/2 bytes = 4 bits
Word	2 bytes = 16 bits

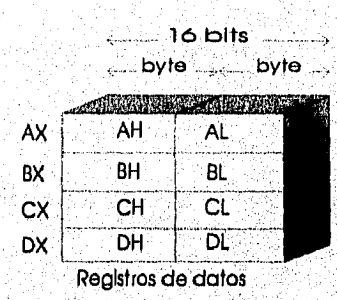
Cada word y cada byte dentro de la memoria tiene una dirección. En el CPU 8086, el byte menos significativo de la palabra está en la dirección más baja.



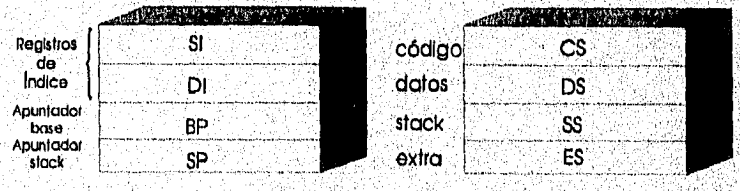
Bytes y Words en Memoria

ALMACENAMIENTO EN MEMORIA

1.2.4 REGISTROS DEL CPU



Registros de datos



CONJUNTO DE REGISTROS

Los registros se utilizan para almacenar información temporalmente, ésta puede ser de uno o dos bytes del dato que va a ser procesado o la dirección del mismo. Se pueden dividir en seis categorías

Generales
- AX Acumulador
- BX Base
- CX Contador
- DX Dato

Apuntadores
- SP Stack Pointer
- BP Base Pointer

Índices
- SI Source Index
- DI Destination Index

Segmentos
- CS Code Segment
- DS Data Segment
- SS Stack Segment
- ES Extra Segment

Instrucción
- IP Instruction Pointer

Banderas
- FR Flag Register

REGISTROS GENERALES

Los registros AX, BX, CX y DX son utilizados para almacenar cualquier tipo de información. Pero en forma particular:

Registro AX

Es el principal registro utilizado en las instrucciones aritméticas, como acumulador de resultados en algún cálculo, por lo que es llamado acumulador.

Registro BX

Es conocido como el registro base ya que puede ser utilizado como índice para direccionar alguna localidad de memoria.

Registro CX

Es conocido como contador, ya que puede contener un valor que controle el número de veces que un ciclo se va a repetir, o en la rotación de bits.

Registro DX

Es conocido como registro de datos, y se utiliza para datos de propósito general

REGISTRO APUNTADORES

Registro SP

El Stack Pointer esta asociado con el registro SS y provee un offset que se refiere a la palabra que está siendo procesada en el stack. EL 80386/486 soporta un registro extendido de 32 bits llamado ESP

Registro BP

Apuntador de propósito general utilizado para direccionar datos dentro del segmento del stack.

REGISTROS ÍNDICES

Registro SI

Es requerido para algunas operaciones de cadena. En este contexto, el registro SI es asociado con el registro DS. EL 80386/486 soporta un registro extendido de 32 bits llamado ESI

Registro DI

Es requerido para algunas operaciones de cadena. En este contexto, el registro SI es asociado con el registro ES. EL 80386/486 soporta un registro extendido de 32 bits llamado EDI

REGISTROS DE SEGMENTOS

Registro CS

Este registro contiene la dirección inicial del segmento de código del programa.

Registro DS

Este registro contiene la dirección inicial del segmento de datos del programa.

Registro SS

Este registro permite la implementación de una stack en la memoria, utilizado para almacenamiento temporal de direcciones y datos. Contiene la dirección de inicio del segmento de stack del programa. Esta dirección más el offset contenido en el registro SP (Stack Pointer) indican la palabra actual del stack que esta siendo direccionada.

Registro ES

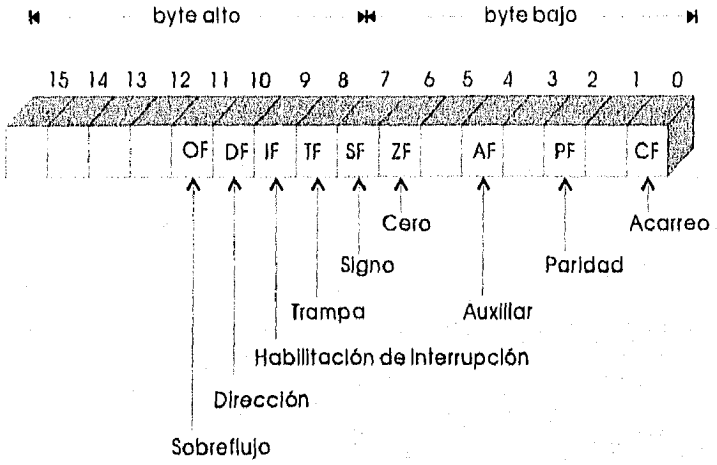
Algunas operaciones de cadenas utilizan este registro. Es asociado con el registro DI.

INSTRUCCIONES

Registro IP

Este registro contiene la dirección de desplazamiento de la instrucción que se va a ejecutar.

REGISTRO DE BANDERAS



REGISTRO DE BANDERAS

El registro de banderas es un registro de 16 bits, pero sólo algunos bits son utilizados, los restantes están indefinidos o reservados por Intel. Seis de las banderas son llamados de estado ya que indican que resulta después de que una instrucción fue ejecutada: CF, PF, AF, ZF, SF, y OF. Las tres restantes son conocidas como de control ya que son utilizadas para controlar la operación de las instrucciones antes de que sean ejecutadas: DF, IF, TF.

BANDERAS DE ESTADO

Bandera de Acarreo

CF (Carry Flag)

Esta bandera se enciende cuando existe un acarreo ya sea del dígito 7 después de una operación de 8 bits o del dígito 15 después de una operación de 16 bits.

Bandera de Paridad

PF (Parity Flag)

Después de ciertas operaciones, la paridad del resultado del byte más bajo se verifica. Si el byte tiene un número par de 1's, la bandera de paridad se inicializa con 1; de otra manera se limpia.

Bandera Auxiliar

AF (Auxiliary Carry Flag)

Si existe un acarreo del dígito 3 al dígito 4 en una operación, este bit se enciende de otra manera se limpia. Esta bandera es utilizada por las instrucciones que ejecutan aritmética BCD (Binary Code Decimal).

Bandera de Cero

ZF (Zero Flag)

Esta bandera tiene valor de 1 si el resultado de una operación aritmética o lógica es cero; de otra manera su valor es 0.

Bandera de Signo

SF (Sign Flag)

La representación binaria de números con signo utilizan el bit más significativo como bit de signo. Después de operaciones lógicas y aritméticas se copia el valor de este bit a SF, indicando el signo del resultado.

Bandera de Sobreflujo

OF (Overflow Flag)

Esta bandera tiene valor de 1 si el resultado de una operación de números con signo es demasiado largo, causando que el bit alto tenga un sobreflujo (overflow) en el bit de signo.

BANDERAS DE CONTROL

Bandera de Trampa

TF (Trap Flag)

Cuando esta bandera tiene valor de 1 permite que el programa se ejecute instrucción por instrucción. Se utiliza para propósitos de depuración.

Bandera de Interrupción

IF (Interrupt Flag)

Este bit se enciende o se apaga para permitir o no interrupciones de peticiones externas.

Bandera de Dirección

DF (Direction Flag)

Este bit se utiliza para controlar las operaciones de cadenas.

SEGMENTO

Un segmento es un área de memoria que incluye hasta 64 K bytes y empieza en una dirección divisible por 16. El tamaño del segmento de 64k bytes es debido a que los Microprocesadores 8085 podían direccionar un máximo de 64K bytes de memoria física ya que sólo tenía 16 pins para las líneas de dirección (2^{16} = 64K). Por compatibilidad esta característica esta dentro del diseño de 8088/86. En 8085 solo existía 64K bytes de memoria para todo el código, los datos y el stack, en un 8088/86 existen 64K bytes de memoria asignada a cada categoría, y tiene un rango de 1 megabyte de memoria por sus 20 pins de direccionamiento (2^{20} = 1 megabyte).

Segmento de Código.

Contiene las instrucciones en Lenguaje Ensamblador que se ejecutan para realizar las tareas.

Segmento de Datos.

Es utilizado para almacenar información necesaria para ser procesada por las instrucciones del segmento de código.

Segmento Extra.

Es utilizado para algunas operaciones de cadenas.

Segmento de Stack

Es utilizado para almacenar información temporalmente.

El stack es un área de memoria de escritura y lectura utilizada por el CPU para almacenar información. El CPU necesita esta área ya que existe un número limitado de registros. La principal desventaja es el tiempo de acceso debido a que es memoria RAM toma más tiempo de acceso que los registros.

Acceso al Stack

Como el stack es una sección de la RAM existen registros en el CPU para apuntarla. Los dos principales registros utilizados para accesar el stack son el registro SS (Stack Segment) y el registro SP (Stack Pointer). Todos los registros del CPU excepto los registros de los segmentos y el SP pueden ser almacenados en el Stack
El grabar registros del CPU en el stack se llama *push* y el cargar los contenidos del stack en los registros del CPU se le llama *pop*.
En los Microprocesadores 80x86 el SP apunta la localidad de memoria actual utilizada en el tope del stack, cuando un dato es metido al *stack* se decrementa y cuando es sacado se incrementa.

1.2.5 DIRECCIÓN FÍSICA Y DIRECCIÓN LÓGICA

En la literatura de Intel de 8086, existen tres tipos de direcciones:

DIRECCIÓN LÓGICA.

Consiste en el valor de la dirección del segmento y la dirección de desplazamiento.

DIRECCIÓN DE DESPLAZAMIENTO.

Es la localización dentro de un rango de segmento de 64K byte, de 0000H a FFFFh.

DIRECCIÓN FÍSICA.

Es la dirección 20 bits que se presenta en los Microprocesadores 8086. Tiene un rango de 000000H a FFFFFH para 8086, 286, 386 y 486 CPU's en modo real

Para ejecutar un programa, el 8086 busca las instrucciones del segmento código. Las direcciones lógicas de una instrucción consisten en CS (Code Segment) y de IP (Instruction Pointer), con un formato CS:IP. La dirección física para la localización de una instrucción es generada moviendo el CS un dígito hexadecimal a la izquierda y sumando el IP. IP contiene el valor de la dirección de desplazamiento.

Para la dirección lógica CS:IP 2500:95F3H la dirección física será 25000 + 95F3 = 2E5F3:

CS : IP

| 2 | 5 | 0 | 0 |

| 9 | 5 | F | 3 |

Moviendo CS a la izquierda

| 2 | 5 | 0 | 0 | 0 |

Suma de IP

| 9 | 5 | F | 3 |

Resultado

| 2 | E | 5 | F | 3 |

1.2.6 MODOS DE DIRECCIONAMIENTO 80x88

El CPU puede accesar operandos de diversas maneras, llamadas modos de direccionamiento. El número de modos de direccionamiento es determinado cuando se diseña el microprocesador y no puede ser modificado.

Los procesadores 80x86 proveen siete distintos modos:

Registro.
Inmediato.
Directo.
Registro indirecto.
Con base.
Con índice.
Con base e índice.

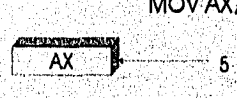
MODO DE DIRECCIONAMIENTO POR REGISTRO.

Implica el uso de registros para manipular el dato. Cuando se ejecuta este modo de direccionamiento no se accesa a la memoria.

DIRECCIONAMIENTO POR REGISTRO.

MODO DE DIRECCIONAMIENTO INMEDIATO.

El operando fuente es constante. Puede ser utilizada para cargar información en cualquier registro excepto en los registros de segmentos y en los registros de banderas.

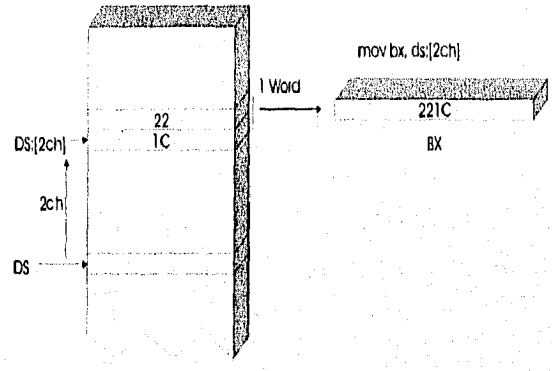


DIRECCIONAMIENTO INMEDIATO

MODO DE DIRECCIONAMIENTO DIRECTO.

En este modo el dato se encuentra de alguna localidad de memoria.

MOVBX, DS:(2Ch)

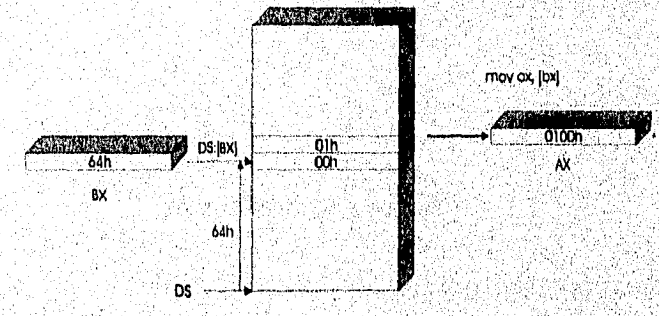


DIRECCIONAMIENTO DIRECTO.

MODO DE DIRECCIONAMIENTO INDIRECTO POR REGISTRO

La dirección de memoria de la localidad donde el dato reside está en un registro. Los registros utilizados para este propósito son SI, DI, y BX. Se debe utilizar DS para generar la dirección física, si estos tres registros son utilizados como apuntadores, es decir, que contienen el desplazamiento de la localidad de memoria.

MOV AX, (BX)

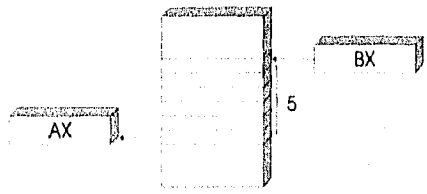


DIRECCIONAMIENTO INDIRECTO POR REGISTRO

MODO DE DIRECCIONAMIENTO CON REGISTRO BASE

Los registros base BX y BP así como un valor de desplazamiento, son usados para calcular cual es la dirección efectiva. Los segmentos por default utilizados para calcular la dirección física son DS para BX y SS para BP.

MOV AX, (BX + 5)

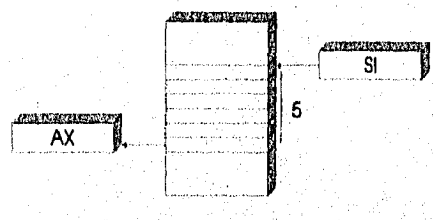


DIRECCIONAMIENTO CON REGISTRO BASE

MODO DE DIRECCIONAMIENTO CON INDICE

Trabaja igual que el modo anterior excepto que se utilizan los registros SI y DI para las direcciones de desplazamientos.

MOV AX, (SI + 5)

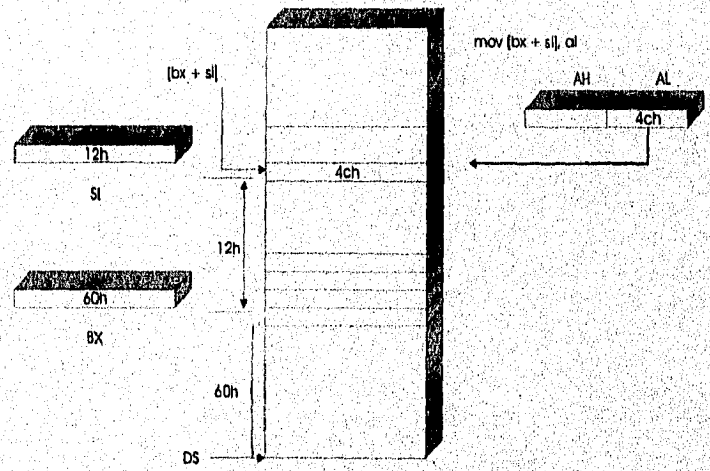


DIRECCIONAMIENTO CON ÍNDICE

MODO DE DIRECCIONAMIENTO CON INDICE Y BASE

Se utiliza un registro base y un registro índice.

MOV (BX+SI), AL



DIRECCIONAMIENTO CON ÍNDICE Y BASE

1.2.7 SEGMENTOS POR DEFAULT Y OVERRIDE

Los registros de desplazamiento por default que se pueden utilizar con los cuatro registros de segmento de 80x86 son

Registros de segmentos	Desplazamiento (offset)
CS	IP
DS	SI, DI, BX
ES	SI, DI, BX
SS	SP, BP

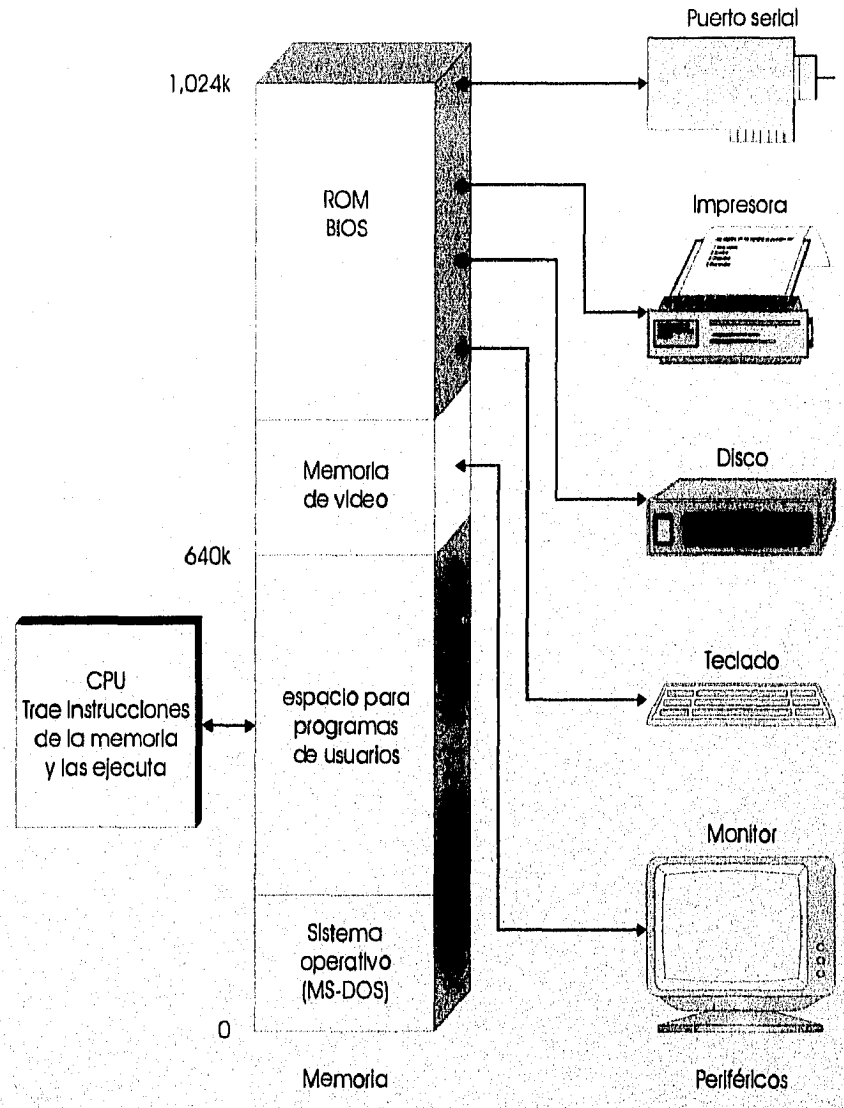
Los CPU 8086 Permiten al programa cambiar los segmentos por default y utilizar cualquier otro registro de segmento, especificándolo en el código.

Resumen de modos de direccionamiento

Modo de direccionamiento	Operando	Segmento
Registro	reg	ninguno
Inmediato	dato	ninguno
Directo	offset	DS
Registro indirecto	BX	DS
	SI	DS
	DI	DS
Con base	BX + desplazamiento	DS
	BP + desplazamiento	SS
Con índice	DI + desplazamiento	DS
	SI + desplazamiento	DS
Con base e índice	BX SI +desplazamiento	DS
	BX DI +desplazamiento	DS
	BP SI +desplazamiento	SS
	BP SI +desplazamiento	SS

1.2.8 EJECUCIÓN DE UN PROGRAMA BAJO DOS

Se puede visualizar el sistema operativo, el CPU, la memoria ROM, la memoria RAM, los periféricos de la siguiente manera:



SISTEMA OPERATIVO

Sin embargo un aspecto importante que el programador de Lenguaje Ensamblador necesita conocer es qué pasa en la memoria cuando un programa se esta ejecutando. Para entender que espera un programa del sistema operativo es necesario conocer los pasos que se generan cuando se enciende la PC hasta que aparece el prompt del DOS.

Cuando se enciende un sistema basado en Intel 8086, se empieza ejecutar el código que se encuentra en la dirección CS:IP = FFFF:0000 en la memoria ROM, éste código tiene rutinas que examinan la integridad del Hardware del sistema y es conocido como POST (Power-On Self Test).

Una vez que el POST se termina con éxito, se ejecuta una rutina que copia el sector del boot de la unidad de disco A o del disco duro en la memoria RAM, este sector contiene información del sistema operativo y un programa que busca los archivos IO.SYS y MSDOS.SYS para cargarlos en la memoria.

El código de IO.SYS consiste en dos partes:

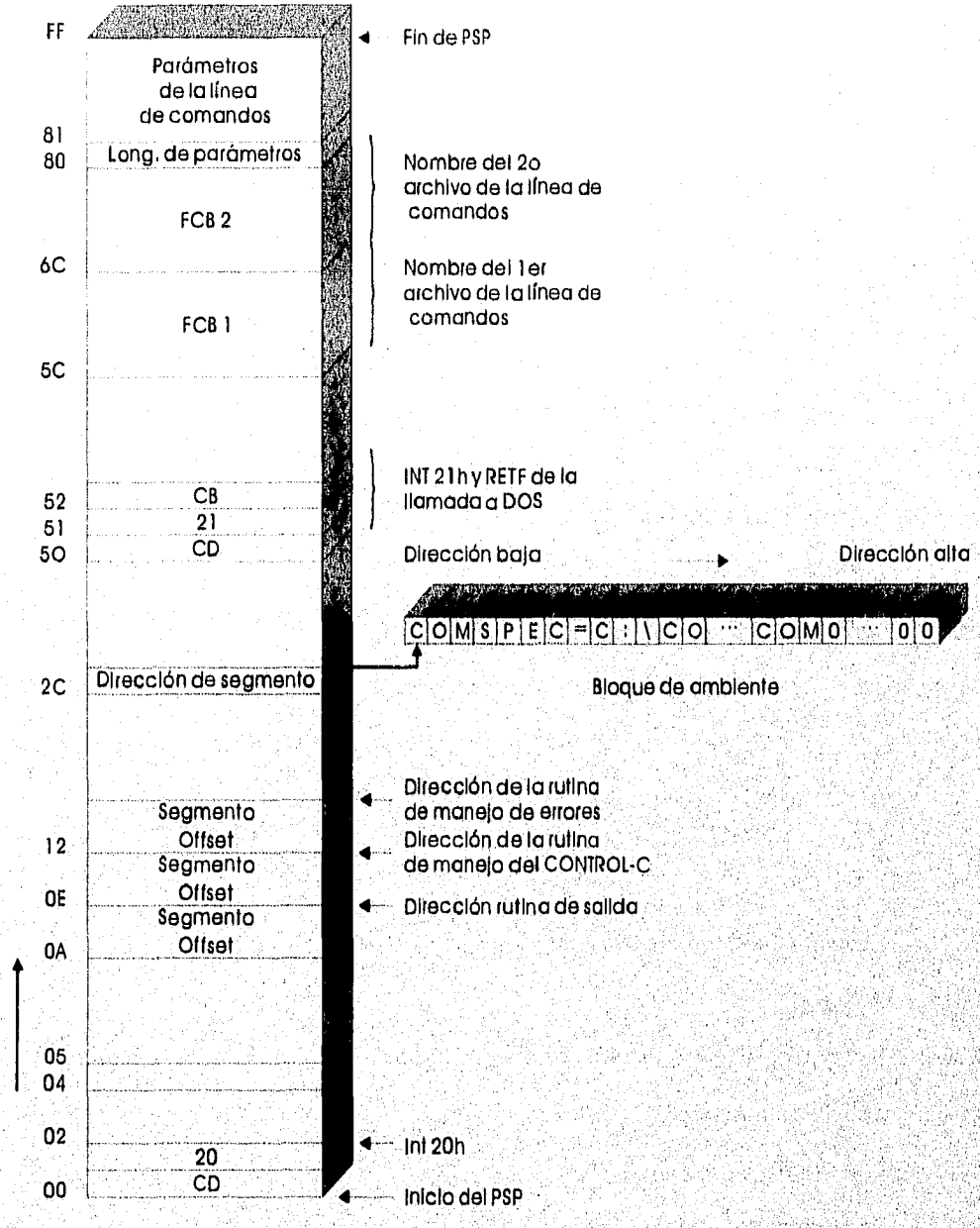
EL BIOS DOS incluye los manejadores, que son códigos que proveen mecanismos para comunicar dispositivos como el teclado, la pantalla, los puertos seriales y las unidades de disco.

EL SYSINIT ejecuta la inicialización del kernel del Sistema Operativo y pocesa el archivo CONFIG.SYS, asignando memoria para las líneas especificadas en BUFFER= y FILES= , cargando los manejadores especificados en la declaración DEVICE=.
Después SYSINIT carga y ejecuta el COMMAND.COM que tiene una parte de inicialización que ejecuta los comandos del archivo AUTOEXEC.BAT.

Ejecución de un programa bajo DOS

El COMMAND.COM carga y ejecuta los archivos con extensión .COM y .EXE: Primero asigna la memoria necesaria para el programa, construye un bloque de 256 bytes conocido como el PSP (Program Segment Prefix) al principio de la memoria asignada, y carga el programa inmediatamente después del PSP. Inicializa los registros del CPU y transfiere el control al inicio del programa cargando CS:IP con la dirección de la primera instrucción del programa.

ÁREA DEL PSP (PROGRAM SEGMENT PREFIX)



ÁREA DE P S P

El PSP es una estructura utilizada para almacenar información relacionada con el programa. Es un bloque de 256 bytes que DOS inserta antes de un programa EXE o un COM cuando lo carga en memoria. El PSP se crea en el offset 0 y el programa en el offset 100H del segmento. El PSP contiene los siguientes campos de acuerdo a su posición relativa:

00-01 Instrucción INT 20H

La primera palabra del PSP es el código máquina de la Interrupción 20H. Este comando es usualmente el último que se ejecuta en un programa.

02-03 Dirección del segmento del último párrafo de memoria asignada al programa.

04-09 Reservado

0A-0D Dirección INT 22H

En CS:000A está la dirección de término. La cual es la dirección de control que se regresa cuando se termina un programa. Utilizando esta dirección es posible hacer llamados a otros procesos en lugar de regresar a DOS.

0E-11 Dirección INT 23H

Dirección de la rutina de manejo del CONTROL-C

12-15 Dirección INT 24H

Dirección de la rutina de manejo de errores

16-17 Reservado

18-2B Tabla predeterminada para el manejo de archivos

Tabla	Dispositivo	Manejador	Dispositivo
01	Consola	0	Teclado
01	Consola	1	Pantalla
01	Consola	2	Pantalla
00	COM1 (puerto serial)	3	Auxiliar
02	Impresora	4	Impresora estándar
FF	Sin asignar	5	Sin asignar

2C-2D Dirección del segmento del ambiente del programa

2E-31 Reservado

32-33 Longitud de la tabla para el manejo de archivos.

34-37 Apuntador a la tabla

38-4F Reservado

50-51 Llamada a las funciones DOS

52-5B Reservado

5C-6B Primer archivo de la línea de comandos

6C-7F Segundo archivo de la línea de comandos (Si existe)

80-81 Longitud de parámetros

FF Parámetros

Un archivo .COM en memoria tiene un PSP de CS:0000 a CS:00ff, seguido del programa en CS:0100. Cuando un archivo .COM se carga, todos los registros CS,DS,ES y SS están en el mismo segmento. DOS coloca el stack al final del segmento, en general los últimos 256 bytes de la dirección CS:FF00 a CS:FFFF.

CAPÍTULO II

ENSAMBLADORES

2.1 *Lenguaje Ensamblador*

- 2.1.1 Pasos para Ensamblar, Ligar y Ejecutar un Programa
- 2.1.2 Reglas para Escribir en Lenguaje Ensamblador
- 2.1.3 Directivas
- 2.1.4 Programas .COM y .EXE
- 2.1.5 Ejercicios

CAPÍTULO II

ENSAMBLADORES

Este capítulo es una introducción al Lenguaje Ensamblador utilizando un producto llamado MASM de Microsoft. Se describe en que consiste las herramientas provistas por este producto y los pasos necesarios para desarrollar en Lenguaje Ensamblador.

2.1 LENGUAJE ENSAMBLADOR

Un Ensamblador es un programa que traduce instrucciones de Lenguaje Ensamblador a Lenguaje Máquina.

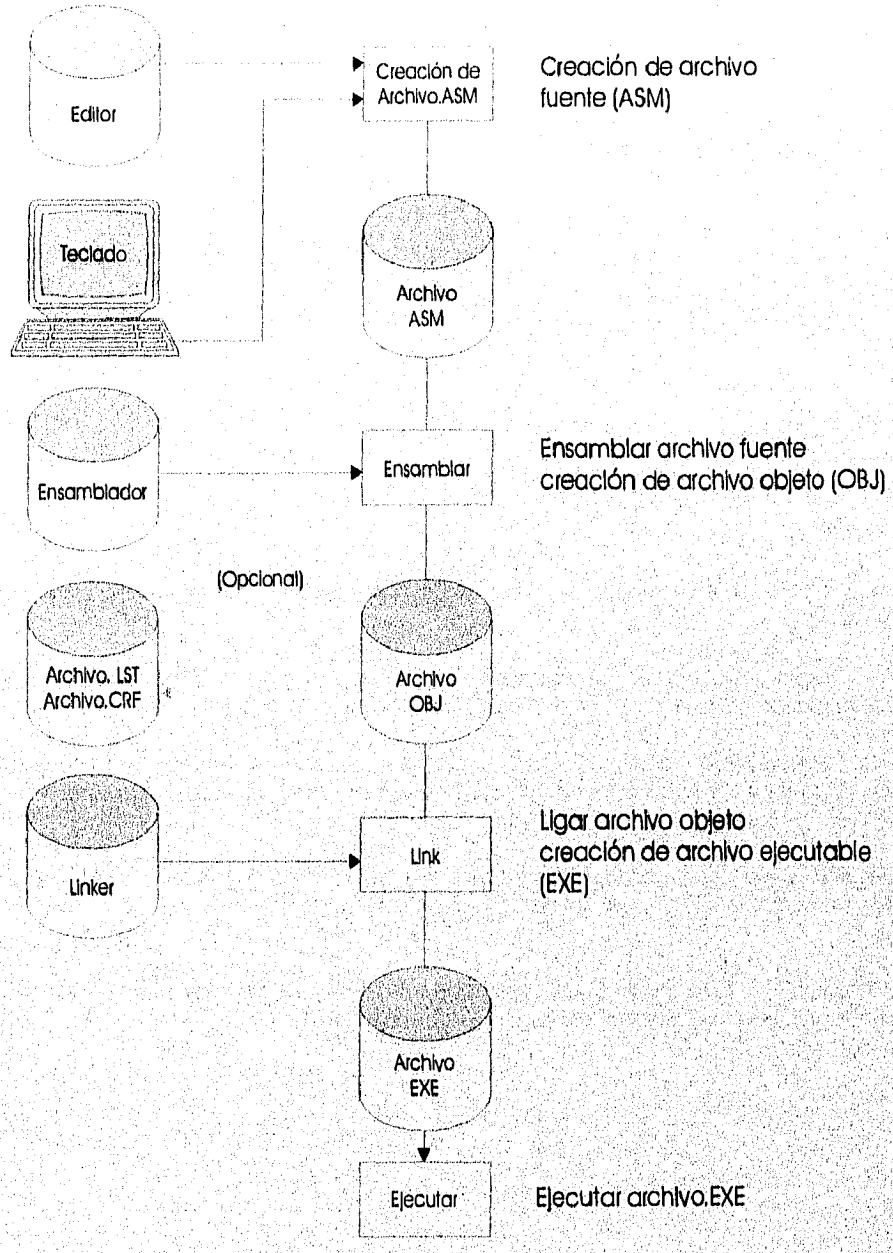
Para traducir código, el ensamblador lee el archivo fuente por lo menos una vez. La mayoría de los ensambladores requieren de dos lecturas para crear el archivo objeto.

Durante la primera pasada, el ensamblador lee el programa fuente y construye una tabla de símbolos de los nombre y etiquetas utilizadas en el programa y se determina el total de código que va a ser generado.

Durante la segunda pasada, el ensamblador utiliza la tabla de símbolos construida en el primera lectura y completa el código objeto para cada instrucción.

El ensamblador que se utiliza en este trabajo es el MASM de Microsoft que es uno de los más conocidos, este ensamblador es de dos pasadas (lecturas).

2.1.1 PASOS PARA ENSAMBLAR, LIGAR Y EJECUTAR UN PROGRAMA



CREACIÓN DE PROGRAMAS

Se necesitan por lo menos tres herramientas para escribir programas en Lenguaje Ensamblador:

Un editor de texto
El ensamblador MASM
El ligador LINK

MASM provee una serie de herramientas para ensamblar, ligar con librerías y crear el archivo ejecutable.

CREACIÓN DE ARCHIVO FUENTE

Escribir el archivo fuente con las instrucciones en Lenguaje Ensamblador, con cualquier editor que produzca archivos ASCII.

ENSAMBLAR EL ARCHIVO FUENTE

Para ensamblar el archivo fuente utilizando el Ensamblador de Microsoft (versiones anteriores a 6.0), desde el prompt del DOS se escribe:

```
C:\>MASM archivo.asm

Microsoft ® Macro Assembler Versión 5.10
Copyright © Microsoft Corp 1981, 1988. All rights reserved.

Object filename (ARCHIVO.OBJ):
Source listing (NUL.LST):archivo.lst
Cross-reference (NUL.CRF):archivo.crf
```

Los archivos utilizados en este proceso son:

Archivo Fuente

Es el archivo creado con un editor, éste contiene instrucciones en Lenguaje Ensamblador y usualmente tiene extensión .ASM.

Archivo Objeto

Después de ensamblar el archivo fuente resulta un archivo binario con código máquina e instrucciones para el linker y su extensión por default es .OBJ

El archivo .LST contiene todos los opcodes, direcciones, y errores que el MASM detecta.

El archivo .CRF provee una lista alfabética de todos los símbolos y etiquetas utilizadas en el programa así como el número de línea del programa donde son utilizadas.

La creación del archivo .LST y del archivo .CRF es opcional.

LIGAR ARCHIVO OBJETO

Para ligar un archivo utilizando las herramientas de Microsoft (versiones anteriores a la 6.0) se escribe desde el prompt de DOS:

```
C:\>LINK archivo.obj

Microsoft ® Overlay Linker Versión 3.64
Copyright © Microsoft Corp 1981, 1988. All rights reserved.

Run File (ARCHIVO.EXE):
List File (NUL.MAP):
Libraries (.LIB):
```

Los archivos utilizados en este proceso son:

Archivo Ejecutable.

Run File se refiere al archivo ejecutable con la extensión por default .EXE. LINK utiliza el primer archivo objeto como el nombre del archivo ejecutable.

List file (.MAP) es el archivo que muestra el nombre de cada segmento del programa, donde empieza, donde termina y su longitud en bytes. El nombre por default NUL.MAP significa que LINK no generará algún archivo map

ENSAMBLAR Y LIGAR CON EL ENSAMBLADOR DE MICROSOFT VERSIÓN 6.0 O POSTERIORES:

En la versión 6.0 Microsoft combina el ensamblador y el ligador en un simple comando: ML

Para ensamblar y ligar un archivo solo se tiene que escribir

ML archivo.asm

Lo que generará los archivos:

archivo.obj
archivo.exe

Para crear los archivos LST, MAP y CRF se utilizan las opciones

Opción	Servicio
/Fl	Crear archivo LST
/Fm	Crear archivo MAP
/Fr	Crear archivo CRF

Si se desea sólo ensamblar sin ligar se utiliza la opción /c

A continuación se analizan los archivos creados en cada paso al escribir un programa en Lenguaje Ensamblador. El programa utilizado despliega en la pantalla el mensaje:

Escuela Nacional de Estudios Profesionales, Acatlán.

El programa escribe.ASM creado con el editor sería:

```
; Escribe
;
; Propósito
; Este programa despliega un mensaje en la pantalla
;
SSEG SEGMENT STACK
      DB    32 DUP("STACK---")
SSEG ENDS

DSEG SEGMENT
      MSG  DB "Escuela Nacional de Estudios Profesionales, Acatlán",0DH,0AH,"$"
DSEG ENDS

CSEG SEGMENT 'CODE'
ASSUME CS:CSEG, SS:SSEG, DS:DSEG
PPAL PROC FAR
      PUSH DS
      MOV AX,00H
      PUSH AX
      MOV AX,DSEG
      MOV DS,AX
      LEA DX,MSG
      MOV AH,09H
      INT 21H
      RET
PPAL ENDP
CSEG ENDS
      END PPAL
```

Descripción.

Todos los programas deben terminar con la directiva END, la cual tiene dos propósitos: indicar el fin del programa y decirle al ensamblador donde se empezará a ejecutar el programa.

En el programa escribe.asm el operando de la directiva END es PPAL, lo cual le indica al ensamblador que la ejecución debe empezar con la instrucción con el nombre PPAL.

La directiva SEGMENT marca el emplezo de un segmento; la directiva ENDS señala el final de cada segmento. Cada segmento debe tener un nombre, en el programa el segmento del stack se inicializa con las líneas

```
SSEG SEGMENT STACK
SSEG ENDS
```

El segmento de datos:

```
DSEG SEGMENT
DSEG ENDS
```

El segmento de código:

```
CSEG SEGMENT 'CODE'
CSEG ENDS
```

En la definición del segmento de stack en la línea:

```
DB      32 DUP("STACK___")
```

La directiva **DB** (define byte) reserva 32 bytes de memoria, **DUP** (duplicar) inicializa el área con la cadena STACK---

En la definición del segmento de datos en la línea:

MSG DB "Escuela Nacional de Estudios Profesionales, Acatlán", 0DH, 0AH, "$"

La directiva **DB** (define byte) le dice al ensamblador el número de bytes para reservar en la memoria y los inicializa la cadena de caracteres.

En el segmento de código la línea:

ASSUME CS:CSEG, SS:SSEG, DS:DSEG

Le indica al ensamblador que cuando el programa se ejecuta, el registro CS contiene la dirección del nombre CSEG; SS contiene la dirección de SSEG; y DS contiene la dirección de DSEG

Así como todos los segmentos tiene directivas que indican donde empiezan y donde terminan, en las líneas:

PPAL PROC FAR
PPAL ENDP

Las directivas **PROC** y **ENDP** indican donde empieza y donde termina el procedimiento **PPAL**

La directiva **FAR** le indican al ensamblador que el procedimiento va a ser llamado desde otro segmento.

La última instrucción del procedimiento PPAL es RET la cual regresa el control al procedimiento que lo llamo, en este caso se regresa el control a DOS. La dirección de retorno se encuentra en el STACK

PUSH DS
MOV AX,00H

Después de ensamblar el archivo fuente se obtiene el archivo escribe.LST que despliega dos partes:

```
Microsoft (R) Macro Assembler Versión 5.00              7/16/95 20:01:40
                                                        Page    1-1
    1                                   ; Escribe
    2                                   ;
    3                                   ; Propósito
    4                                   ; Este programa despliega un mensaje en la pantalla
    5                                   ;
    6 0000                              SSEG SEGMENT STACK
    7 0000  0020[                           DB      32 DUP("STACK---")
    8           53 54 41 43 4B
    9           2D 2D 2D
   10                     ]
   11
   12 0100                              SSEG ENDS
   13
   14 0000                              DSEG SEGMENT
   15 0000  45 73 63 75 65 6C 61            MSG DB "Escuela Nacional de Estudios Profesionales, Acatlán",0DH,0AH,"$"
   16       20 4E 61 63 69 6F 6E
   17       61 6C 20 64 65 20 45
   18       73 74 75 64 69 6F 73
   19       20 50 72 6F 66 65 73
   20       69 6F 6E 61 6C 65 73
   21       2C 20 41 63 61 74 6C
   22       A0 6E 0D 0A 24
   23 0036                              DSEG ENDS
   24
   25 0000                              CSEG SEGMENT 'CODE'
   26                                   ASSUME CS:CSEG, SS:SSEG, DS:DSEG
   27 0000                              PPAL PROC FAR
   28 0000  1E                              PUSH DS
   29 0001  B8 0000                         MOV AX,00H
   30 0004  50                              PUSH AX
   31 0005  B8 ---- R                       MOV AX,DSEG
   32 0008  8E D8                           MOV DS,AX
   33 000A  8D 16 0000 R                    LEA DX,MSG
   34 000E  B4 09                           MOV AH,09H
   35 0010  CD 21                           INT 21H
   36 0012  CB                              RET
   37 0013                              PPAL ENDP
   38 0013                              CSEG ENDS
   39                                       END PPAL
```

Descripción

En ésta parte del archivo .LST el programa aparece a la derecha y la traducción en Código Máquina a la izquierda, todos los números en la traducción están en hexadecimal. Los primeros cuatro dígitos hexadecimales son el offset dentro del segmento, a la derecha de este offset están los números hexadecimales que representan la instrucción en lenguaje hexadecimal de esa línea.

El ensamblador empieza cada nuevo segmento en el offset 0000H (cero) y puede llegar hasta FFFFH lo que da un tamaño máximo de segmento de 64K bytes.

La segunda parte:

```
Microsoft (R) Macro Assembler Versión 5.00              7/16/95 20:01:40
                                                        Symbols-1
Segments and Groups:
                N a m e         Length   Align   Combine Class
CSEG  . . . . . . . . . . . .    0013    PARA     NONE   'CODE'
DSEG  . . . . . . . . . . . .    0036    PARA     NONE
SSEG  . . . . . . . . . . . .    0100    PARA     STACK
Symbols:
                N a m e         Type     Value    Attr
MSG   . . . . . . . . . . . .    L BYTE   0000     DSEG
PPAL  . . . . . . . . . . . .    F PROC   0000     CSEG   Length = 0013
@FILENAME . . . . . . . . . .    TEXT  uno
        28 Source  Lines
        28 Total  Lines
         8 Symbols
50634 + 332086 Bytes symbol space free
         0 Warning Errors
         0 Severe  Errors
```

Descripción

Esta parte contiene información acerca de los nombres que se definieron en el programa y un resumen de total de líneas y errores.

Es posible crear un reporte de referencias cruzadas a partir de la información del archivo .CRF utilizando el programa CREF.EXE que viene con el ensamblador

```
C:\>cref escribe.crf
Microsoft (R) Cross-Reference Utility  Version 5.00
Copyright (C) Microsoft Corp 1981-1985, 1987.  All rights reserved.

Listing [escribe.REF]:

6 Symbols

C:\>_
```

El archivo que se genera (escribe.ref) es:

```
Microsoft Cross-Reference  Version 5.00        Sun Jul 16 20:45:20 1995

Symbol Cross-Reference      (# definition, + modification)  Cref-1

CODE . . . . . . . . . . . .     25
CSEG . . . . . . . . . . . .     25#  26   38

DSEG . . . . . . . . . . . .     14#  23   26   31

MSG. . . . . . . . . . . . .     15#  33

PPAL . . . . . . . . . . . .     27#  37   39

SSEG . . . . . . . . . . . .     6#   12   26

 6 Symbols
```

Descripción

Este archivo contiene todos los nombres utilizados en el programa, en orden alfabético y el número de línea donde se hace referencia a éstos.

El archivo escribe.map generado por el linker es:

```
Start     Stop     Length     Name     Class
 00000H    000FFH   00100H    SSEG
 00100H    00135H   00036H    DSEG
 00140H    00152H   00013H    CSEG     CODE

Program entry point at 0014:0000
```

Descripción

Este archivo despliega cada segmento en el orden que son cargados en la memoria. La longitud de los segmentos en bytes y la dirección relativa de inicio y fin de cada segmento.

USO DE ARCHIVOS BATCH

El uso de Archivos BATCH es muy útil al desarrollar programas. Un Archivo Batch es un archivo texto con extensión .BAT que contiene una serie de comandos MSDOS. Puede utilizar argumentos que se encuentran en %1 para el primer argumento, %2 para el segundo argumento, etc.

Un ejemplo sería

```
echo off
rem
if not "%1" == " " goto ensamblar
:uso
echo !        uso:            ENSAMBLA nom_arch
echo !
echo ! donde        nom_arch es el nombre del archivo
echo !
goto fin
:ensambla
if exist %1.asm goto trabaja
echo !        El archivo %1.asm no existe.
goto fin
:trabaja
echo !        Ensamblando . . .
ML %1.asm
:fin
```

Descripción

Este archivo batch revisa si existe algún argumento, verifica que exista el nombre de archivo que está en el argumento y ejecuta el programa MASM.

ML traduce los archivos fuente en archivos objeto. El sistema operativo no puede ejecutar estos archivos. ML invoca el ligador LINK para combinar uno o más archivos objeto dentro de un archivo ejecutable.

El siguiente comando ensambla y liga el archivo prueba.asm

ML PRUEBA.ASM

y produce los archivos PRUEBA.OBJ y PRUEBA.EXE.

2.1.2 Reglas para Escribir en Lenguaje Ensamblador

Los programas en Lenguaje Ensamblador consisten en una secuencia de declaraciones. Existen tres tipos de declaraciones: comentarios, instrucciones y directivas.

Los comentarios son ignorados por ensamblador, son útiles para documentar y explicar la lógica de los programas.

Las instrucciones son aquellas que le dicen a la computadora que hacer. Una instrucción es una declaración que va a ser traducida en Lenguaje Máquina.

Una directiva es una declaración que provee direcciones al ensamblador. Algunas veces son llamadas pseudo operaciones. Las directivas no son traducidas a Lenguaje Máquina.

Las reglas para escribir en Lenguaje Ensamblador dependen del ensamblador que se este utilizando pero en general:

Cada línea puede contener solo una declaración
Una declaración puede empezar en cualquier posición de la línea
Es posible utilizar mayúsculas o minúsculas. El ensamblador solo hace distinciones en cadenas de caracteres que están entre comillas.

El formato de los comentarios es simple: solo deben empezar con un punto y coma.
Las instrucciones y las directivas tienen tres partes: una etiqueta o nombre, un opcode o mnemónico y un operando separados por lo menos por un espacio o una tabulación.

(etiqueta:) opcode (operandos)

Todas las instrucciones o directivas deben tener un opcode, pero no todas un operando o un nombre. Los paréntesis indican que el campo es opcional.

La etiqueta permite al programa referirse a una línea de código por un nombre. El opcode es la parte que identifica una operación especifica. El operando es la parte que representa el valor en el cual la instrucción o directiva actúa.

Reglas para Nombres

Es posible utilizar letras, dígitos y los caracteres ? $ _ @.
El primer caracter de un nombre no debe ser algún dígito.
Pueden ser de la longitud que se quiera, el ensamblador solo utiliza los primeros 31 caracteres.
No escribir nombres que sean iguales a algún opcode.

Reglas para Números

Se pueden utilizar números en binario, decimal o hexadecimal, para especificar número decimales se utilizan los dígitos 0-9. Para especificar un número hexadecimal se debe añadir al final del número la letra "H", si comienza el número con una letra A-F se debe añadir un cero antes. Para un número binario se utilizan los dígitos 0 y 1, y se añade al final del número una "B".

2.1.3 DIRECTIVAS

CONSTANTES Y VARIABLES

Al definir una constante se especifica el valor con el cual va a ser inicializada:

Ejemplo:

```
CONST   DB   100
```

Nota: Se define una constante llamada CONST que va a ser inicializada con el valor de 100 (decimal)

Al definir una variable se especifica que no va a ser inicializada.

Ejemplo:

```
TOTAL   DB   ?
```

Nota: En esta línea se define una variable llamada TOTAL, con el símbolo ? el ensamblador no inicializa el valor de total.

TIPOS DE DATOS

Existen varios tipos de datos:

TIPO	DESCRIPCIÓN	TAMAÑO (en Bytes)
BYTE	byte	1
WORD	word	2
DWORD	doubleword	4
QWORD	quadword	8
TBYTE	10 bytes	10

DIRECTIVAS PARA DEFINICIÓN DE DATOS

Para definir los datos es necesario utilizar la directiva adecuada para cada tipo de dato. Se escribe el nombre del dato en la parte del nombre de la declaración, Si se requiere definir una variable, se escribe el símbolo ? (pregunta) en la parte del operando de la declaración; si se define una constante se especifica su valor como operando.

DIRECTIVA	TIPO
DB	BYTE
DW	WORD
DD	DWORD
DQ	QWORD
DT	TBYTE

Ejemplo:

```
        CONST   DB   ?
```

Nota: Se define una variable de un byte

```
        VARS    DW   0
```

Nota: Se define una constante de un word

Para definir una variable de más de un byte, word, dword,...se repite el signo **?**, el número da veces que se requiera.

Ejemplo:

Lista **DD** ?,?,?,?,?

Nota: Se define una variable de cinco doublewords

Para definir una constante de más de un byte, word, dword,...se especifica la lista de valores.

Ejemplo:

Tabla **DB** 1,2,3,4,5,6,7,8,9,10

Nota: Se define una constante llamada tabla de una lista de 10 números, cada uno almacenado en un byte.

También se pueden definir constantes de caracteres, escribiendo los caracteres entre comillas dobles o simples.

DIRECTIVA PARA REPETIR VALORES

Si se requiere definir datos de un valor repetido, se especifica el número de valores idénticos, seguido de la palabra DUP y el valor entre paréntesis.

Ejemplo:

GUIONES DB **DUP** ("-")

Nota: Se define una constante llamada guiones de 30 bytes, cada una inicializada con "-".

LISTA DW 200 **DUP**(?)

Nota: Se define una variable de 200 words

REFERENCIA A DATOS

Cuando se utiliza el nombre del dato dentro de una instrucción, el nombre se refiere al offset del primer dato, para referirse al siguiente bloque de datos, es necesario sumar el valor del tamaño en bytes al nombre de la variable

Ejemplo:

BLISTA DB ?,?,?

NOTA: BLISTA es el primer dato,
BLISTA + 1 es el segundo dato
BLISTA + 2 es el último

WLISTA DW 500 DUP(?)

NOTA: WLISTA es el primer dato,
WLISTA + 2 es el segundo dato
WLISTA + 4 es el tercer dato

USO DE LA DIRECTIVA DB

Caracteres

Cada byte puede contener un caracter o un número.

Si se requiere definir una constante de un caracter, se especifica el caracter entre comillas dobles o sencillas, para definir una serie de caracteres se escribe toda la cadena entre comillas o cada caracter entre comillas separados por comas, si se utiliza un caracter que no tenga algún símbolo asociado a él, se especifica el código ASCII del caracter sin comillas.

Ejemplos:

```
ASTERIS   DB "*"
LETRAS    DB "A","B","C","D"
LETRAS    DB "ABCD"
MSG       DB "H O L A",0DH,0AH,"$"
MSG       DB "H O L A",0DH,0AH, 24H        ; ASCII de $ en hex
MSG       DB "H O L A",0DH,0AH, 36         ; ASCII de $ en dec
```

Números

El rango de números que pueden estar almacenados en un byte es:

Tipo de número	Rango
Sin signo	0 - 255
Con signo	-127 a 128

Ejemplos:

```
SSIGNO    DB  0, 35, 0EFH, 43, 1001100B
CSIGNO    DB  0, -35, 126, -128, -70H, -100010B
```

USO DE LA DIRECTIVA DW

Caracteres

Cada word puede contener uno o dos caracteres, un número o un offset.

Normalmente se utilizan bytes en lugar de words para definir caracteres, pero en ciertas circunstancias, se necesita almacenar caracteres en un word.

Ejemplo:

```
DOSCAR DW "13"
```

Nota:El byte izquierdo se inicializa con el caracter "1" (31H) y el caracter derecho con el caracter "3" (33H), por lo que la constante DOSCAR contiene 3133H

```
UNCAR  DW "1"
```

Nota:El byte izquierdo se inicializa con 00H y el caracter derecho con el caracter "1" (31H), por lo que la constante UNCAR contiene 0031H

No se puede utilizar la directiva DW para definir constantes que consistan en una serie de caracteres.

Números

El rango de números que pueden estar almacenados en un byte es:

Tipo de número	Rango
Sin signo	0 - 65535
Con signo	-32768 - 32767

Ejemplos:

```
SSIGNO    DW  0, 34998, 0FFADH, 47, 1011010011101B
CSIGNO    DW  0, -31999, -800H, 32767, -128, -1000100001000B
```

DIRECCIONANDO ELEMENTOS SIN NOMBRE

No es necesario definir un nombre para cada dato que se defina. Se puede hacer referencia a cualquier localidad dando la dirección relativa al nombre previo.

Ejemplo:

```
          TABLA     DB   1,2,3,4,5
                    DB   ?,?
                    DB   8,9,10
```

Nota:Esta declaración define una tabla de 10 bytes, TABLA+5 y TABLA+6 hacen referencia al byte 6 y al byte 7.

ATRIBUTOS

Un atributo describe una característica particular del dato. Cada dato tiene cinco diferentes atributos

TYPE

Es el número de bytes reservados para cada definición

LENGTH

Es el número total de datos

SIZE

Es el número total de bytes reservados para el dato (TYPE multiplicado por LENGTH)

SEG

Es la dirección de inicio del segmento en donde esta el dato

OFFSET

Es la dirección del primer byte del dato.

Ejemplo:

```
LISTA   DB  100 DUP(?)
MOV     AX, TYPE LISTA          ;2
MOV     AX, LENGTH LISTA        ;100
MOV     AX, SIZE LISTA          ;200
```

OPERADORES DENTRO DE LOS OPERANDOS

Existen varios operadores que se pueden utilizar para ejecutar aritmética simple dentro de los operandos:

+	Suma
-	Resta
*	Multiplicar
/	Dividir
MOD	Modulo

Estos operadores sólo especifican cálculos para el ensamblador, no realizan aritmética cuando el programa se esta ejecutando.

Ejemplos:

```
        MOV   AX,TABLA+10
        MOV   AX,TABLA+2*(100+66)
TABLA   DB    2*1024 DUP(?)
LISTA2  DB    2*(SIZE LISTA1) DUP(?)
```

EL OPERADOR PTR

El operador PTR le dice al ensamblador que invalide el tipo por default por solo una instrucción. Se especifica el tipo de datos que se quiere utilizar seguido de PTR y el nombre del dato.

Ejemplos:

```
PAR     DB    ?,?
        MOV   AX, WORD PTR PAR

PALABRA DW    ?
        MOV   AH, BYTE PTR PALABRA
        MOV   BH, BYTE PTR PALABRA+1
```

Tipos de datos que pueden ser utilizados por el operador PTR:

BYTE
WPRD
DWPRD
QWPRD
TBYTE

LA DIRECTIVA LABEL

Permite definir un nombre con un atributo específico. EL formato es:

nombre **LABEL** type

El nombre no ocupa algún espacio en el Lenguaje Máquina. Su uso es una alternativa para referenciarse a un lugar particular en el programa.

LA DIRECTIVA EQU

Permite definir símbolos que tienen un valor específico mientras el programa es ensamblado. El formato es:

nombre **EQU** expresión

donde expresión puede ser un número sin signo (0..65535) en decimal, hexadecimal o binario y puede tener operadores aritméticos y atributos

No genera algún código máquina en el programa. Solo son instrucciones que se utilizan durante el proceso de ensamblar.

Ejemplo:

```
K       EQU  1024

DATO1   DB   K    DUP(?)     ; 1024
DATO2   DB   2*K  DUP(?)     ; 2*1024
DATO3   DB   3*K  DUP(?)     ; 3*1024
```

LA DIRECTIVA ORG

El ensamblador mantiene un valor llamado contador de localidades el cual contiene el offset de la siguiente localidad disponible. Cuando el ensamblador empieza un programa, el contador de localidades es 0, cada vez que se asigna memoria para datos o instrucciones el valor del contador aumenta.
En un programa se puede hacer referencia a este valor utilizando el símbolo $. Este caracter, es útil para definir la longitud de una secuencia de caracteres.

Se puede inicializar el contador a un valor utilizando la directiva ORG. El formato es:

ORG valor

La cual le indica al ensamblador empezar a ensamblar en un offset mayor que 0.

2.1.4 PROGRAMAS .COM Y .EXE

En DOS existen dos tipos de programas ejecutables:

.COM
.EXE

Diferencias entre un programa EXE y un programa COM

Tamaño de programa

Un programa .EXE puede ser de cualquier tamaño, un programa .COM está restringido a un segmento (64k), incluyendo el PSP. Un programa .COM es siempre más pequeño que su programa .EXE original; una de las razones es que el bloque de 512 bytes que precede a un programa .EXE no existe en los programas .COM.

Segmentos

Segmento del stack
Un programa .EXE se define con el segmento de stack mientras que un programa .COM genera automáticamente un stack. Cuando un programa en Lenguaje Ensamblador va a ser convertido en .COM se debe omitir el stack.

Segmento de datos.
Un programa .EXE usualmente define el segmento de datos e inicializa el registro DS con su dirección. En un programa .COM se define los datos dentro del segmento de código.

Segmento de código
El programa .COM completo consiste en un segmento de código de un máximo de 64k, incluyendo el PSP, el stack y los datos.

Conversión

Si se escribe un programa para que se ejecute como un programa .COM, el linker produce el mensaje:

Warning: No STACK Segment

Se debe ignorar este mensaje, ya que en un programa .COM no se define el stack. Existe un programa llamado EXE2BIN que convierte programas .EXE a programas .COM.

Uso:

C:\>EXE2BIN PROG PROG.COM

El primer operando se refiere al archivo .EXE por lo que no es necesario escribir la extensión. El segundo operando puede ser otro nombre que PROG.COM. Si se omite la extensión, EXE2BIN asume la extensión BIN, pero es necesario renombrar el archivo como COM.

PROTOTIPO DE UN PROGRAMA COM

```
1
2     CSEG SEGMENT
3          ASSUME CS:CSEG, SS:CSEG, DS:CSEG, ES:CSEG
4
5          ORG 100H
6     INICIO:
7          JMP PPAL
8     MSG  DB    "Definición de datos",00H
9     L_MSG      EQU  $-MSG
10
11    PPAL  PROC
12
13
14
15
16
17         RET
18    PPAL  ENDP
19
20    CSEG ENDS
21
22         END INICIO
```

DISEÑO DE UN PROGRAMA COM

Estructurar el programa de manera que sólo ocupe un segmento.

Utilizar la directiva ASSUME para indicarle al ensamblador que todos los registros de los segmentos van apuntar al mismo segmento.

Utilizar la directiva ORG para asignar 100H bytes al principio del programa para el Program Segment Prefix (PSP).

El programa debe empezar inmediatamente después del PSP, por lo que se pone una etiqueta después de la directiva ORG y la directiva END refiriéndose a esta etiqueta.

Todos los datos deben estar en algún lugar dentro del segmento, es recomendable ponerlos adelante del procedimiento principal y utilizar la instrucción JMP para ir al procedimiento principal.

Todos los procedimientos del programa, incluyendo el principal, deben ser de tipo NEAR.

No es necesario definir el área del stack. Cuando se carga el programa, el DOS inicializa el stack al final de los 64k. Por lo que, el stack crece hacia el área del programa.

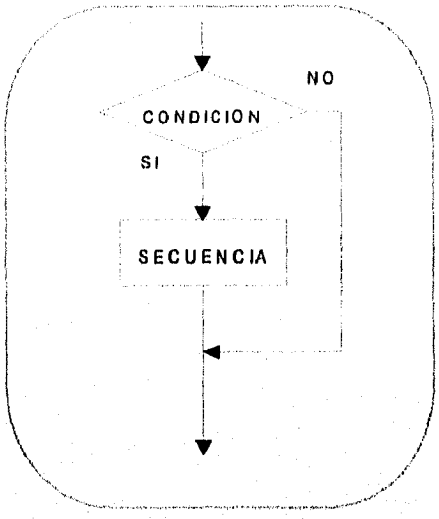
RET al final de un programa .COM transfiere el control al principio del PSP donde es ejecutada la instrucción:

```
INT   20H
```

2.1.5 EJERCICIOS

Un lenguaje de alto nivel tiene funciones construidas lo que hace fácil utilizarlas, en Lenguaje Ensamblador se tiene que construir.

IF . . . THEN



if CONDICION **then** SECUENCIA

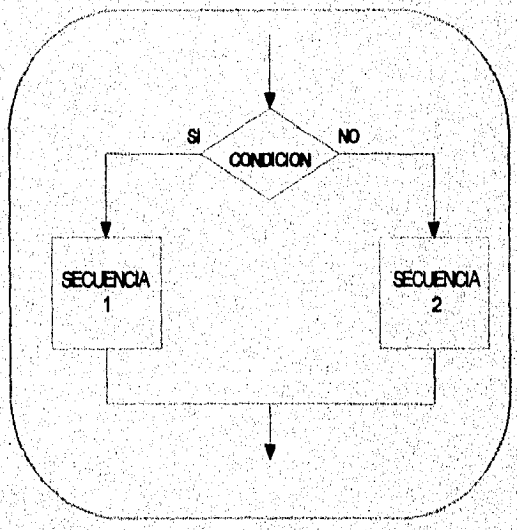
if OPCION = 1 **then**
SWAP(VAR1, VAR2)

```
        CMP   OPCION, 1
        JNE   L1
        MOV   AX, VAR1
        XCHG  AX, VAR2
        MOV   VAR1, AX
L1:
```

if OPCION < 0 **then**
VAR1 = VAR1 + 1
OPCION = 0

```
        CMP   OPCION, 0
        JNL   L1
        ADD   VAR1, 1
        MOV   OPCION, 0
L1:
```

IF . . . THEN . . . ELSE

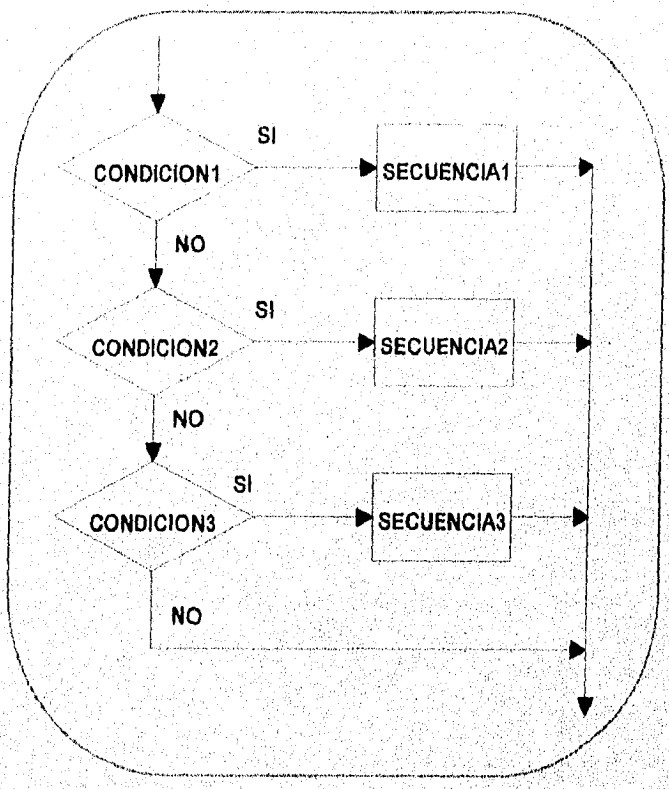


if CONDICION **then**
SECUENCIA1
else
SECUENCIA2

```
If OPCION > 0 then
        VAR1 = VAR1 +1
else
        VAR2 = VAR2 + 1
```

```
                CMP   OPCION,0
                JG    L1
                JMP   L2

        L1:
                ADD   VAR1,1
                JMP   L3
        L2:
                ADD   VAR2,1
        L3:
```

CASE

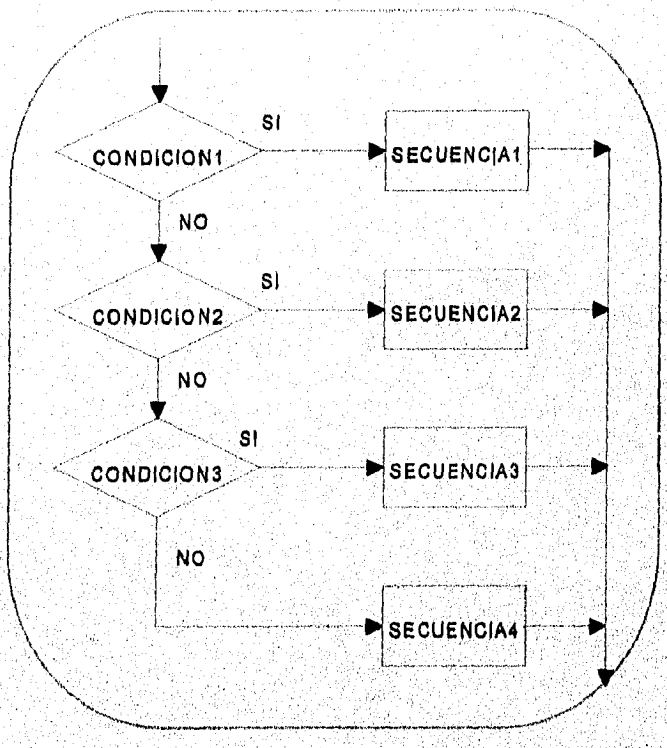



```
case
CONDICION1: SECUENCIA1,
CONDICION2: SECUENCIA2,
CONDICION3: SECUENCIA3, . . .
```

```
case OPCION
        1:    call procedimiento PROC1
        2:    call procedimiento PROC2
        < 0:  AX + VAR1
```

```
        CMP  OPCION,1
        JE   L1
        CMP  OPCION,2
        JE   L2
        CMP  OPCION,0
        JL   L3
        JMP  L4

L1:
        CALL PROC1
        JMP  L4
L2:
        CALL PROC2
        JMP  L4
L3:
        ADD  AX,VAR1
L4:
```

CASE . . . ELSE . . .

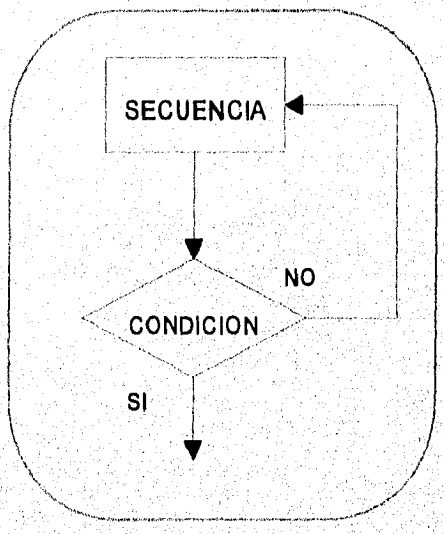


case
CONDICION1: SECUENCIA1,
CONDICION2: SECUENCIA2,
CONDICION3: SECUENCIA3,
else
SECUENCIA4

case OPCION
1: call procedimiento PROC1
2: call procedimiento PROC2
< 0: sumar VAR al registro AX
else: sumar 1 a VAR

```
        CMP   OPCION,1
        JE    L1
        CMP   OPCION,2
        JE    L2
        CMP   OPCION,0
        JL    L3
        JMP   L4
L1:
        CALL  PROC1
        JMP   L5
L2:
        CALL  PROC2
        JMP   L5
L3:
        ADD   AX,VAR1
        JMP   L5
L4:
        ADD   VAR,1
L5:
```

REPEAT . . . UNTIL . . .

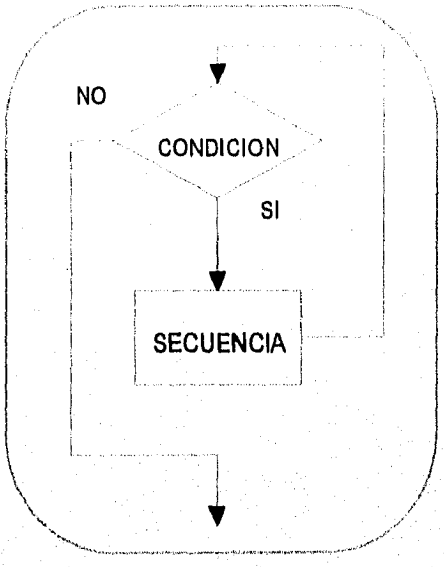



```
repeat
    SECUENCIA
until CONDICION
```

```
repeat
        PROC1
        PROC2
until   CF != 1

L1:
        CALL  PROC1
        CALL  PROC2
        JC    L1
```

```
repeat
        PROC1
        PROC2
until   AX = 0

L1:
        CALL  PROC1
        CALL  PROC2
        CMP   AX,0
        JNE   L1
```

WHILE . . . DO



while CONDICION **do**
SECUENCIA

while CONT > 0 **do**
CALL PROC1

```
L1:
        CMP   CONT,0
        JNG   L2
        CALL  PROC1
        SUB   CONT,1
        JMP   L1
L2:
```

CAPÍTULO III

PROGRAMACIÓN EN LENGUAJE ENSAMBLADOR

CAPÍTULO III

PROGRAMACIÓN EN LENGUAJE ENSAMBLADOR

Este capítulo describe como utilizar los comandos de DEBUG, y explica brevemente las instrucciones más comunes del Lenguaje Ensamblador.

3.1 USO DEL DEBUG

Los programadores de computadoras se refieren a los errores de los programas como bugs, existen dos clases de depuradores para quitar los bugs de los programas:

Software
Hardware

Un depurador de software (como el DEBUG.COM que se distribuye con el Sistema Operativo) es un programa que se queda residente en memoria y el depurador de Hardware combina Hardware y software para realizar sus funciones.

El DEBUG reconoce los archivos EXE y COM y sirve para:

Ver el contenido de la memoria (RAM y ROM)
Ejecutar el programa instrucción por instrucción
Traducir y ejecutar programas en Lenguaje Ensamblador.

DEBUG

Inicia un programa que se utiliza para probar y depurar archivos ejecutables.

Sintaxis

DEBUG (*nomarchivo* (*parámetros*))

Parámetros

nomarchivo
Especifica la posición y el nombre del archivo ejecutable del que se desee hacer la prueba.

parámetros
Especifica cualquier información en la línea de comandos, requerida por el archivo ejecutable del que se desee hacer la prueba.

Si usa el comando DEBUG sin indicar una posición ni nombre de archivo, deberá escribir todos los comandos en respuesta a la línea de comandos de DEBUG (un guión).

3.1.1 COMANDOS DE DEBUG

Los comandos del programa DEBUG son:

A	Ensamblar	Ensambla códigos mnemónicos
C	Comparar	Compara dos porciones de la memoria.
D	Vaciar	Presenta el contenido de una porción de la memoria.
E	Introducir	Introduce datos en la memoria a partir de una dirección especificada.
F	Llenar	Llena un rango de la memoria con los valores especificados
G	Ir	Ejecuta el archivo ejecutable que está en la memoria.
H	Hex	Realiza cálculos aritméticos hexadecimales.
I	Entrada	Presenta el valor de un byte de un puerto especificado.
L	Cargar	Carga el contenido de un archivo o sectores de un disco en la memoria
M	Desplazar	Copia el contenido de un bloque de memoria.
N	Nombre	Especifica un archivo para un comando L o W o especifica los parámetros para el archivo del que se esté haciendo la prueba
O	Salida	Envía el valor de un byte a un puerto de salida
P	Continuar	Ejecuta una operación de bucle, una instrucción de cadena repetida, una interrupción de software o una subrutina.
Q	Salir	Finaliza la sesión con DEBUG.
R	Registro	Presenta o altera el contenido de uno o más registros
S	Buscar	Busca una configuración específica de uno o más valores de bytes en una porción de la memoria.
T	Seguir	Ejecuta una instrucción y presenta el contenido de todas los registros, el estado de los indicadores y la forma descodificada de la siguiente instrucción que será ejecutada por DEBUG.
U	Desensamblar	Desensambla bytes y presenta las instrucciones de origen correspondientes.
W	Escribir	Escribe en un disco el archivo cuya que se esté probando

? Presenta una lista de los comandos de DEBUG.

Todas las comandos DEBUG, excepto el comando Q, aceptan parámetros. Estos comandos se pueden separar con comas o espacios, pero las separadores sólo son requeridos entre dos valores hexadecimales. Por la tanto los siguientes comandos son equivalentes:

```
dcs:100 110
d cs:100 110
d,cs:100,110
```

En un comando de DEBUG, un parámetra dirección especifica una pasición en la memoria. Dirección es una designación que consta de dos partes y que contiene una dirección de segmento de cuatro dígitos, más un valor de desplazamiento. Se puede omitir el registro del segmento o la dirección del segmento. El segmento predeterminado para los comandos A, G, L, T, U y W es CS. El segmento predeterminado para todos las demás comandos es DS. Todos los valores numéricos tienen formato hexadecimal.

Las direcciones siguientes son válidas:

```
CS:0100
04BA:0100
```

Los dos puntos (:) entre el nombre del segmento y el valor de desplazamiento son requeridos.

Un parámetro de rango en un comando de DEBUG especifica un rango de memoria. Se podrá elegir entre dos formatos para rango: una dirección de inicio y de finalización o una dirección de inicio y la longitud (definida por L) del rango.
Por ejemplo, las dos sintaxis que se indican a continuación especifican un rango de 16 bytes que comienza en CS:100:

```
cs:100 l 10
cs:100 10f
```

A
Assemble

La instrucción assemble es utilizada para introducir Lenguaje Ensamblador y traducirlo en instrucciones de Lenguaje Máquina en memoria.

Sintaxis

A (*dirección*)

Parámetros

dirección

Es una dirección de inicio opcional donde se van a introducir las instrucciones de Lenguaje Ensamblador. Si no se especifica la dirección, DEBUG empieza en CS:0100 o después de la última instrucción de Lenguaje Máquina introducida.

Cada línea es ensamblada después de oprimir el enter.

```
C:\>DEBUG
-A
15CE:0100 JMP FFFF:0000
15CE:0105
-
```

El ensamblador ensamblará automáticamente en la dirección de destino las llamadas e instrucciones de salto cortas, cercanas o lejanas, según el desplazamiento de los bytes. Usted podrá anular estas instrucciones de salto o llamadas usando un prefijo NEAR o FAR, como se indica en el ejemplo siguiente:

```
-a0100:0500
0100:0500 jmp  502          ; un salto corto de 2 bytes
0100:0502 jmp  near 505     ; un salto cercano de 3 bytes
0100:0505 jmp  far  50a     ; un salto lejano de 5-bytes
```

El prefijo NEAR se puede abreviar NE.

Cuando un operando puede hacer referencia a la posición de memoria de una palabra o a la posición de memoria de un byte, será necesario especificar el tipo de datos con el prefijo WORD PTR o BYTE PTR. Las abreviaturas admitidas son WO y BY, respectivamente. El siguiente ejemplo muestra los dos formatos:

```
dec     wo (si)
neg     byte ptr (128)
```

El programa DEBUG utiliza la convención común que pone entre corchetes [ ] los operandos que hacen referencia a una posición de memoria. De otra manera, DEBUG no podrá distinguir entre un operando inmediato y un operando que represente una posición de memoria. El ejemplo siguiente muestra los dos formatos:

```
mov     ax,21       ; cargar AX con 21h
mov     ax,[21]     ; cargar AX con el contenido de la posición de memoria 21h
```

Dos seudoinstrucciones comúnmente empleadas están disponibles con el comando A: el código de operación DB, que ensambla directamente en la memoria los valores de byte y el código de operación DW, que ensambla directamente en la memoria los valores de las palabras. A continuación damos ejemplos de ambas seudoinstrucciones:

```
db      1,2,3,4,"ESTE ES UN EJEMPLO"
dw      1000,2000,3000,"BACH"
```

C
Compare

Compara y reporta cualquier diferencia entre el contenido de dos bloques de memoria.

Sintaxis

C *rango dirección*

Parámetros

rango

Es la dirección inicio y la dirección de término del bloque o la dirección de inicio y la longitud del primer bloque de memoria.

dirección

Es el inicio del segundo bloque de memoria. Se asume que la longitud del segundo bloque es igual que la primera.

Compara los dos bloques byte por byte, y reporta cualquier diferencia en el siguiente formato:

dirección1 valor1 valor2 dirección2

```
C:\>DEBUG
-D 100 L 20
15CE:0100 41 42 43 44 45 46 47 48-49 4A 4B 4C 4D 4E 4F 50
ABCDEFGHIJKLMNOP
15CE:0110 61 62 63 64 65 66 67 68-69 6A 6B 6C 6D 6E 6F 70
abcdefghijklmnop
-C 100 L 10 110
15CE:0100  41   61  15CE:0110
15CE:0101  42   62  15CE:0111
15CE:0102  43   63  15CE:0112
15CE:0103  44   64  15CE:0113
15CE:0104  45   65  15CE:0114
15CE:0105  46   66  15CE:0115
15CE:0106  47   67  15CE:0116
15CE:0107  48   68  15CE:0117
15CE:0108  49   69  15CE:0118
15CE:0109  4A   6A  15CE:0119
15CE:010A  4B   6B  15CE:011A
15CE:010B  4C   6C  15CE:011B
15CE:010C  4D   6D  15CE:011C
15CE:010D  4E   6E  15CE:011D
15CE:010E  4F   6F  15CE:011E
15CE:010F  50   70  15CE:011F
-
```

D

Dump

Despliega el contenido de una serie de localidades de memoria.

Sintaxis

D (*rango*)

Parámetros

rango

Es la dirección de inicio y finalización o la dirección de inicio y la longitud del área de memoria que se desea presentar.

Si no se especifica alguna dirección, DEBUG comienza a desplegar las localidades de memoria a partir de DS:0100 o si Dump ya ha sido utilizado, a partir del último byte desplegado en el mas reciente dump.

Si no se especifica la dirección final, DEBUG despliega 128 bytes de memoria. Cada byte se muestra en su representación hexadecimal y en su representación ASCII.

```
C:\>DEBUG
-D
15CE:0100  0F 00 B9 30 28 8B FF F3-AE 47 1F 6C 40 8B C3 48  ...0(....G.l@..H
15CE:0110  12 B1 04 8B C6 F7 D0 D3-8102 48 DA 34 00 BD 15  ..........H.4...
15CE:0120  2A 2B D2 D3 E0 60 DB 03-F0 8E DA 8B C7 16 C2 36  *+...`.........6
15CE:0130  00 16 C0 16 F8 8E C2 AC-8A  D0 4E AD 8B 00 00  C8  ..........N.....
15CE:0140  46 8A C2 24 FE 3C B0 75-05 AC F3 AA EB 06 3C 54  F..$..u......T
15CE:0150  01 B2 75 6D 6D 13 A8 01-74 B1 BE 8A 42 32 01 8D  ..umm...t...B2..
15CE:0160  8B 1E 8E FC 33 D2 02 55-29 E3  13 8B C2 03 C3 69  ....3..U).....I
15CE:0170  00 00 0B F8 83 FF FF 74-11 26  01 1D E2 F3 81 FA  .......t.&......
-
```

Cuando usa el comando D, DEBUG presenta el contenido de la memoria en dos porciones: una porción hexadecimal (cada valor de byte es mostrado con formato hexadecimal) y una porción ASCII (cada valor de byte es mostrado como caracter ASCII). Los caracteres no imprimibles están indicados por un punto (.) en la porción ASCII de la presentación. Cada línea presentada muestra el contenido de 16 bytes, con un guión entre el octavo y noveno byte. Cada línea presentada comienza en un límite de 16 bytes.

```
E
Enter
```

Introduce datos en la memoria en la dirección especificada.

Sintaxis

E *dirección* (*lista*)

Parámetros

dirección

Es la dirección de inicio para introducir los datos.

lista

Es una lista opcional de datos que se van a introducir.

La lista se puede especificar en cualquier combinación de números hexadecimales o en caracteres ASCII.

Si no se especifica la lista DEBUG permite cambiar los valores un byte a la vez.

```
C:\>DEBUG
-E 100 'minuscula      '
-D 100 L 10
15CE:0100  6D 69 6E 75 73 63 75 6C-6120 20 20 20 20 20 20 20  minuscula
-E 100
15CE:0100  6D.4D  69.41  6E.59  75.55  73.53  63.43  75.55  6C.4C
15CE:0108  61.41
-D 100 L 10
15CE:0100  4D 41 59 55 53 43 55 4C-4120 20 20 20 20 20 20 20  MAYUSCULA
-
```

Si se especifica un valor para dirección sin especificar el valor del parámetro optativo lista, DEBUG presentará la dirección y su contenido, repetirá la dirección en la línea siguiente y esperará a que se introduzcan datos. Llegado este momento, permite realizar una de las acciones siguientes:

Reemplazar el valor del byte, introduciendo un nuevo valor después del valor actual.
Avanzar al siguiente byte presionando la BARRA ESPACIADORA.
Regresar al byte anterior presionando la tecla GUION.
Finalizar el comando E, presionando la tecla ENTRAR.

Si se especifican valores para el parámetro lista, el comando E reemplazará en forma secuencial los valores de bytes existentes con los valores de la lista.

Los valores de lista pueden ser hexadecimales o cadenas. Cada valor debe separarse del anterior por un espacio, coma o tabulación. Las cadenas deben estar entre comillas simples o dobles.

F

FIII

Se utiliza para llenar un bloque de memoria con un valor específico o una serie de valores.

Sintaxis

F *rango valor*

Parámetros

rango

Es la dirección inicio y la dirección de término del bloque o la dirección de inicio y la longitud del bloque de memoria.

valor

Puede ser una combinación de números hexadecimales o caracteres ASCII.

```
C:\>DEBUG
-F 100 L 10 'ABCDEFGHIJKLMNOP'
-D 100 I 10
15CE:0100  41 42 43 44 45 46 47 48 - 49 4A 4B 4C 4D 4E 4F 50
ABCDEFGHIJKLMNOP
-
```

Si rango contiene más bytes que el número de valores de lista, DEBUG asignará repetidamente los valores de lista hasta llenar todos los bytes de rango.
Si alguna parte de la memoria de rango es defectuosa o no existe, DEBUG presentará un mensaje de error y detendrá el comando F.

Si lista contiene más valores que el número de bytes de rango, DEBUG hace caso omiso de los valores adicionales de lista.

G

Go

Permite que instrucciones de Lenguaje Máquina sean ejecutadas. Permite especificar puntos de ruptura opcionales, los cuales son direcciones donde la ejecución del programa termina.

Sintaxis

G (=*inicio*) (*ptoruptura*)

Parámetros

=inicio
Es una dirección de inicio opcional.

ptoruptura
Son direcciones de puntos de ruptura opcionales.

Si no se especifica la dirección de inicio Go empezará a ejecutar a partir de la dirección de CS:IP actual. Si no se llega a los puntos de ruptura, la ejecución continúa hasta que el programa se termina.

```
C:\>DEBUG
-G = FFFF:0000
```

Se requiere un signo igual (=) antes de dirección a fin de distinguir la dirección de inicio (dirección) de las direcciones de los puntos de ruptura (puntos de ruptura).

El programa se detendrá en el primer punto de ruptura que encuentre, sin importar la posición en la que se haya introducido en la lista de puntos de ruptura. DEBUG reemplazará la instrucción original en cada punto de ruptura con un código de interrupción.
Cuando el programa llegue a un punto de ruptura, DEBUG restablecerá las instrucciones originales de todas las direcciones de los puntos de ruptura y presentará el contenido de todos los registros, el estado de todos los indicadores y la forma decodificada de la última instrucción ejecutada. DEBUG presentará la misma información que si se hubiera utilizado el comando de DEBUG R (registro) y especificado la dirección del punto de ruptura.
Si no se detiene el programa en uno de los puntos de ruptura, DEBUG no reemplazará los códigos de ruptura con las instrucciones originales.

Se podrán fijar puntos de ruptura solamente en las direcciones que contengan el primer byte de un código de operación 8086 (código de operación). Si se especifican más de 10 puntos de ruptura, DEBUG presentará el siguiente mensaje:

bp Error

DEBUG colocará un código de interrupción (0CCh) en la dirección (o direcciones) del punto de ruptura especificado.

Para reiniciar el programa correctamente, se deberá volver a cargarlo usando los comandos de DEBUG N (Nombre) y L (Cargar).

H
Hexadecimal Arithmetic

Realiza adiciones y sustracciones hexadecimales.

Sintaxis

H *valor1 valor2*

Parámetros

valor1 y valor2 Son números hexadecimales.

DEBUG suma primero los dos parámetros especificados y luego resta el segundo parámetro del primero. El resultado de estos cálculos se presenta en una línea, primero la suma, luego la diferencia.

No altera los registros del CPU.

```
C:\>DEBUG
-H 10 10
0020  0000
```

I

Input

Busca un byte del puerto.

Sintaxis

I *puerto*

Parámetros

puerto

Es la dirección de un puerto específico para leer.

```
C:\>DEBUG
-I 2F8
FF
-
```

L

Load

Es utilizado para cargar un archivo o sectores del disco a la memoria.

Sintaxis

L *dirección numdrive lnisector numsector*

Parámetros

dirección

Es la dirección de memoria destino donde va a ser cargada la información.

numdrive

Es un número opcional para designar el drive.
0 = A, 10= B, 2 = C.

inisector

Es el sector donde va a empezar la lectura

numsector

Es el número total de sectores de disco que se van a leer.

```
C:\>DEBUG
-L CS:100 0 0 1

No está lista leyendo unidad A
Anular, Repetir, Descartar? R
-_
```

Si no se proveen los parámetros DEBUG carga el archivo especificado en la instrucción **N** Name en la dirección CS:0100.
DEBUG también asignará los registros BX y CX al número de bytes cargados. Si no se especificó un archivo en la línea de comandos de DEBUG, se cargará el último archivo que se especificó usando el comando N.

Si se usa el comando L con el parámetro dirección, DEBUG comenzará a cargar el archivo o el contenido de los sectores indicados en la dirección de memoria especificada.

Si se usa el comando L con todos los parámetros, DEBUG cargará el contenido de sectores de disco específicos en lugar de cargar un archivo.

Cada sector del rango especificado se leerá desde unidad. DEBUG comienza a cargar a partir de inicio y continuará hasta cargar el contenido del número de sectores especificado por el parámetro número.

DEBUG hace caso omiso del parámetro dirección para archivos .EXE. Si se especifica un archivo .EXE, DEBUG cambiará la posición del archivo, colocándolo en la dirección de carga especificada en el encabezado del archivo .EXE. El encabezado mismo será eliminado del archivo .EXE antes de que sea cargado en la memoria, de manera que el tamaño de un archivo .EXE en el disco diferirá de su tamaño en la memoria. Si se desea examinar un archivo .EXE completo, se debe asignar otra extensión.

M
Move

Mueve un bloque de memoria de una localidad a otra.

Sintaxis

M *rango dirección*

Parámetros

rango

Es la dirección inicio y la dirección de término del bloque o la dirección de inicio y la longitud del bloque de memoria.

dirección

Es la dirección destino para el movimiento.

```
ENEP ACATLAN
C:\>DEBUG
-F 100 L 18 "E N E P  A C A T L A N"
-M CS:100 L 18 B800:0
-
```

El comando M realiza operaciones de copias en las que una parte del bloque de destino está parcialmente superpuesta en el bloque de origen, sin ocasionar pérdida de datos en las direcciones de destino. Se copia primero el contenido de las direcciones en las que se sobreescribirá. De esta manera, si los datos se copian desde direcciones altas a direcciones más bajas, la operación de copia comienza en la dirección más baja del bloque de origen y termina en la dirección más alta. Inversamente, si los datos se copian desde direcciones bajas a direcciones más altas, la operación de copia comienza en la dirección más alta del bloque de origen y termina en la más baja.

N

Name

Se utiliza para especificar un nombre de archivo que va a ser utilizado por las instrucciones Load o Write o por el programa que se esta depurando.

Sintaxis

N *nomarch*

Parámetros

nomarch

Posición y nombre del archivo que se desea probar.

```
C:\>DEBUG
-N EDIT.COM
-L
Archivo no se encuentra

-
```

El comando N tiene dos funciones:
Para especificar un archivo que será utilizado por un comando L o W posterior. Si se inicia DEBUG sin especificar el archivo que será depurado, se deberá usar el comando N antes de usar el comando L para cargar el archivo. El nombre del archivo tendrá el formato correcto para un bloque de control de archivos en CS:5C.
Para especificar parámetros y modificadores en la línea de comandos para el archivo que se esté depurando.

Las siguientes cuatro áreas de la memoria pueden verse afectadas por el comando N:

Posición de memoria	Contenidos
CS:5C	Bloque de control de archivos (FCB) para el archivo 1
CS:6C	Bloque de control de archivos (FCB) para el archivo 2
CS:80	Longitud de la línea del comando N (en caracteres)
CS:81	Principio de los caracteres de la línea del comando N

El primer nombre de archivo especificado para el comando N se coloca en un bloque de control de archivos (FCB) situado en CS:5C. Si se especifica un segundo nombre de archivo, éste será colocado en un bloque de control de archivos situado en CS:6C. El número de caracteres introducidos en la línea del comando N (excluyendo la letra N) será almacenado en la posición CS:80. Los caracteres que se encuentren en la línea del comando N (excluyendo de nuevo la letra N) serán almacenados a partir de CS:81. Observe que estos caracteres podrán ser cualquiera de los modificadores y delimitadores que se consideren válidos en un comando que se introduzca a continuación del símbolo del sistema de MS-DOS.

O
Output

Envía un byte por un puerto específico.

Sintaxis

O *puerto valor*

Parámetros

puerto
Es la dirección del puerto específico.

valor
Es el byte hexadecimal que se va a escribir.

```
C:\>DEBUG
-I 3f8
00
-
```

P
Proceed

Permite ejecutar instrucciones de Lenguaje Máquina paso a paso, considerando los CALL e INT como una simple instrucción.

Sintaxis

P (=*inicio*) (*cont*)

Parámetros

Inicio

Es la posición de la primera instrucción que será ejecutada. Si no se especifica una dirección, la predeterminada será la dirección actual de CS:IP.

cont

Es el número de instrucciones ejecutadas antes de devolver el control a DEBUG

```
-U 110 L 5
15EE:0110  B44A      MOV  AH,4ª
15EE:0112  CD21      INT  21
15EE:0114  BA5102    MOV  DX,0251
-T=0110

AX=4A00  BX=0049  CX=01BB  DX=0000  SP=0482  BP=0000  SI=0000  DI=0000
DS=15EE  ES=15EE  SS=15EE  CS=15EE  IP=0112   NV UP DI PL NZ AC PO NC
15EE:0112 CD21       INT  21
-P

AX=15EE  BX=0049  CX=01BB  DX=0000  SP=0482  BP=0000  SI=0000  DI=0000
DS=15EE  ES=15EE  SS=15EE  CS=15EE  IP=0114   NV UP DI PL NZ AC PO NC
15EE:0114 BA5102     MOV  DX,0251
-
```

Cuando el comando P transfiere el control desde DEBUG al programa que se esté probando, el programa en prueba se ejecutará sin interrupción hasta que finalice el bucle, la instrucción de cadena repetida, la interrupción del software o la subrutina en la dirección que se haya especificado, o hasta que se haya ejecutado el número de instrucciones de máquina especificado. El control volverá después a DEBUG.

Si el parámetro dirección no especifica un segmento, DEBUG utilizará el registro CS del programa en prueba. Si dirección se omite por completo, el programa se ejecutará comenzando con la dirección especificada por los registros CS:IP de ese programa. El parámetro dirección debe estar precedido por un signo igual (=) para poder distinguirlo del parámetro número. Si la instrucción en la dirección especificada no es un bucle, una instrucción de cadena repetida, una interrupción de software o una subrutina, el comando P funcionará como el comando T.

Después de que P ejecute una instrucción, DEBUG presentará el contenido de los registros del programa, el estado de los indicadores y la forma descodificada de la siguiente instrucción a ejecutarse.

No se puede utilizar el comando P para hacer seguimiento a través de memoria de sólo-lectura (ROM).

Q

Quit

Se utiliza para salir de DEBUG y regresar el control al DOS.

Sintaxis

Q

```
C:\>DEBUG
-Q

C:\>_
```

R

Register

Despliega los valores de los registros del microprocesador y de las banderas., y permite cambiar el valor de los registros.

Sintaxis

R (*registro*)

Parámetros

registro

Es el nombre del registro que se va a cambiar AX, BX, CX, DX, SP, BP, SI, DI, DS, ES, SS, CS, IP Y F.

Sin parámetros DEBUG despliega el contenido de todos los registros.

```
C:\>DEBUG
-R
AX=0000  BX=0000  CX=0000  DX=0000  SP=FFEE  BP=0000  SI=0000  DI=0000
DS=15CE  ES=15CE  SS=15CE  CS=15CE  IP=0100   NV UP EI PL NZ NA PO NC
15CE:0100 0F        DB   0F

-R CX
CX 0000
:5

-R F
NV UP EI PL NZ NA PO NC -ZR
-R F
NV UP EI PL ZR NA PO NC -
```

Si se especifica un nombre de registro, MS-DOS presentará el valor de 16 bits de ese registro en anotación hexadecimal y mostrará dos puntos (:) como símbolo. Si desea cambiar el valor contenido en el registro, deberá introducir un nuevo valor y presionar ENTRAR. Si no; presione ENTRAR para regresar al símbolo del sistema de DEBUG.

Los siguientes son valores válidos para nombre-registro: AX, BX, CX, DX, SP, BP, SI, DI, DS, ES, SS, CS, IP, PC y F. Tanto IP como PC se refieren al apuntador de instrucciones.
Si especifica un nombre de registro diferente a los de la lista precedente, MS-DOS presentará el siguiente mensaje:

bf error

Si se introduce el caracter F en lugar de un nombre de registro, DEBUG presentará el estado actual de cada indicador en formato de código de dos letras y después presentará la línea de comandos

de DEBUG. Para cambiar cualquier indicador, introduzca el código de dos letras que corresponda, como se muestra en la siguiente tabla:

Indicador	Activado	Desactivado
Desbordamiento	OV	NV
Dirección	DN (decremento)	UP (incremento)
Interrupción	EI (habilitado)	DI (inhabilitado)
Signo	NG (negativo)	PL (positivo)
Zero	ZR	NZ
Acarreo auxiliar	AC	NA
Paridad	PE (par)	POR (impar)
Acarreo	CY	NC

Podrá introducir nuevos valores para los indicadores en cualquier orden. No necesitará dejar espacios entre esos valores. Para salir del comando R, presione ENTRAR. Cualquier indicador para el que no se haya especificado un nuevo valor quedará inalterado.

Si se especifica más de un valor para un indicador, DEBUG presentará lo siguiente:

dferror

Si se especifica un código de indicador distinto a los mostrados en la tabla anterior, DEBUG presentará el siguiente mensaje:

bf error

En ambos casos, DEBUG hará caso omiso de todos los valores especificados después del valor no válido.

Al iniciarse DEBUG, los registros de segmento están establecidos en la parte inferior de la memoria libre, el apuntador de instrucciones está establecido como 0100h, todos los indicadores están borrados y los demás registros están establecidos como 0, con la excepción de SP, que está establecido como FFEEh.

S
Search

Permite buscar un bloque de memoria con un secuencia de valores específicos.

Sintaxis

S *rango valor*

Parámetros

rango

Es la dirección inicio y la dirección de término del bloque o la dirección de inicio y la longitud del bloque de memoria

valor

Son los valores que se quieren buscar en el bloque de memoria.

Los valores que DEBUG busca pueden ser números hexadecimales o caracteres ASCII. Si encuentra el patrón se despliega su dirección de inicio.

```
C:\>DEBUG
-S FFFF:1C80 L 20 "Copyright"
FFFF:1C91
-D FFFF:1C91
FFFF:1C90       43 6F 70 79 72 69 67 - 68 74 20 31 39 38 31 2D   Copyright 1981-
FFFF:1CA0  31 39 39 33 20 4D 69 63 - 72 6F 73 6F 66 74 20 43   1993 Microsoft C
FFFF:1CB0  6F 72 70 4D 61 74 65 72 - 69 61 6C 20 42 61 6A 6F   orpMaterial Bajo
FFFF:1CC0  20 4C 69 63 65 6E 63 69 - 61 20 2D 20 50 72 6F 70   Licencia - Prop
FFFF:1CD0  69 65 64 61 64 20 64 65 - 20 4D 69 63 72 6F 73 6F   iedad de Microso
FFFF:1CE0  66 74 20 44 65 72 65 63 - 68 6F 73 20 72 65 73 65   ft Derechos rese
FFFF:1CF0  72 76 61 64 6F 73 20 3C - 06 76 03 B0 FF CF 1E 2E   rvados .v......
FFFF:1D00  8E 1E F7 3D 50 56 BE 37 - 03 32 E4 0B C0 75 04 8A   ...=PV.7.2...u..
FFFF:1D10  14
-
```

Si el parámetro lista contiene más de un valor de byte, DEBUG presentará solamente la primera dirección en la que ocurra dicho valor de byte. Si la lista contiene sólo un valor de byte, DEBUG presentará todas las direcciones dentro del rango especificado en el que ocurra el valor.

T

Trace

Permite ejecutar instrucciones de Lenguaje Máquina paso a paso.

Sintaxis

T (*= inicio*) (*cont*)

Parámetros

inicio

Es la posición de la primera instrucción que será ejecutada. Si no se especifica una dirección, la predeterminada será la dirección actual de CS:IP.

cont

Es el número de instrucciones ejecutadas antes de devolver el control a DEBUG

```
C:\>DEBUG
-N C:\DOS\HELP.COM
-L
-U 100 L A
15EE:0100 B88204        MOV  BX,0482
15EE:0103 8BE3          MOV  SP,BX
15EE:0105 83C30F        ADD  BX,+0F
15EE:0108 D1EB          SHR  8X,1
-T

AX=0000   BX=0482  CX=01BB  DX=0000  SP=FFFE   BP=0000   SI=0000   DI=0000
DS=15EE   ES=15EE  SS=15EE  CS=15EE  IP=0103   NV UP EI PL NZ NA PO
NC
15EE:0103 8BE3       MOV   SP,BX
-T

AX=0000   BX=0482  CX=01BB  DX=0000  SP=0482   BP=0000   SI=0000   DI=0000
DS=15EE   ES=15EE  SS=15EE  CS=15EE  IP=0105   NV UP EI PL NZ NA PO
NC
15EE:0105 83C30F     ADD   BX,+0F
-
```

El comando T utiliza el modo de seguimiento de Hardware de los Microprocesadores 8086 u 8088, de manera que se pueda hacer el seguimiento de las instrucciones almacenadas en la memoria de sólo-lectura (ROM).

El parámetro dirección deberá estar precedido por el signo (=) a fin de distinguirlo del parámetro número.

U

Unassemble

Decodifica los valores de un grupo de localidades de memoria en mnemónicos 8088.

Sintaxis

U (*rango*)

rango Es la dirección de inicio opcional del área de memoria que va a ser desensamblada.

```
C:\>DEBUG
-N BOOT.COM
-L
-U 100 L 5
15ED:0100 EA0000FFFF   JMP   FFFF:0000
-
```

W

Write

Se utiliza para almacenar un archivo o sectores individuales en el disco.

Sintaxis

W buffer numdrive lnlsector numsector

buffer	Es la dirección de memoria de la información que va a ser almacenada.
numdrive	Es un número opcional para designar el drive. 0 = A, 10= B, 2 = C.
lnlsector	Es el sector donde va a empezar la escritura.
numsector	Es el número total de sectores de disco que se van a escribir.

Si no se proveen los parámetros DEBUG escribe el archivo especificado en la instrucción **N** Name en la dirección CS:0100.

Antes de que se escriba el archivo, CX debe contener el número de bytes que se van a escribir.

```
C:\>DEBUG
-A
15CE:0100 JMP FFFF:0000
15CE:0105
-N BOOT.COM
-RCX
CX 0000
:5
-W
Escribiendo 00005 bytes
-
```

Si ha utilizado el comando G (Ir), T (Seguir), P (Continuar) o R (Registro) de DEBUG, deberá reasignar los registros BX:CX antes de usar el comando W sin parámetros.

Si modifica el archivo pero no cambia el nombre, la longitud ni la dirección de inicio, DEBUG podrá aún escribir el archivo correctamente en la posición original del disco.
No es posible escribir un archivo .EXE o .HEX con este comando.

Creación de comandos utilizando el DEBUG

Una de las formas para crear un archivo COM es:

1. Invocar el DEBUG con el nombre del programa.
2. Introducir las instrucciones utilizando el comando A 100
3. Utilizar el comando R CX para indicar el tamaño del programa
4. Utilizar el comando W para escribir el programa en el disco
5. Utilizar el comando Q para salir del DEBUG y regresar al prompt del DOS

Invocar el DEBUG con el nombre del archivo como se muestra:

```
C:\>DEBUG HOLA.COM
No se encontró el archivo
```

DEBUG despliega el mensaje "Archivo no encontrado" ya que el archivo HOLA.COM no existe todavía, DEBUG utiliza el caracter guión (-) como su prompt. Para decirle a DEBUG que se quiere introducir instrucciones, se teclea A 100:

```
C:\>DEBUG HOLA.COM
No se encontró el archivo

-A 100
16C9:0100
```

DEBUG responde desplegando la localidad de memoria donde se va a ensamblar el archivo. Teclear las instrucciones que constituyen el programa

```
16C9:0100  MOV AH, 9
16C9:0102  MOV DX, 10B
16C9:0105  INT 21
16C9:0107  MOV AH, 4C
16C9:0109  INT 21
16C9:010B  DB 'Hola, mundo!$'
16C9:0119
-
```

Teclear el comando R CX para indicar cuántos bytes de memoria abarcan las instrucciones que se acaban de introducir.

```
-R CX
```

Para determinar el tamaño se debe restar la dirección de inicio de la última dirección. En este caso la última dirección es 0119 y la de inicio es 0100, Realizando la resta el tamaño es de 19 bytes,

```
CX 0000
:19
```

Utilizar el comando W para escribir el programa en el disco,

```
-W
Escribiendo 00019 bytes
```

Utilizar el comando Q, para salir del DEBUG y regresar al prompt del DOS

```
-Q

C:\>_
```

Una vez realizado estos pasos se puede ejecutar el programa:

```
C:\> HOLA
Hola, mundo!
```

Si el programa no se ejecuta con éxito, es necesario utilizar la combinación de teclas Control-Alt-Supr.

Otros programas ejemplo:

```
C:\>DEBUG BOOT.COM
No se encontró el archivo

-A
15CE:0100 JMP FFFF:00
15CE:0105
-RCX
CX 0000
:5
-W
Escribiendo 00005 bytes
-Q

C:\>_
```

Descripción.
Este programa transfiere el control a la rutina de inicialización que se encuentra en FFFF:0000.

```
C:\ >debug
-A
15CE:0100 MOV AH,9
15CE:0102 MOV DX,10B
15CE:0105 INT 21
15CE:0107 MOV AH,4C
15CE:0109 INT 21
15CE:010B DB "Universidad Nacional Autónoma de México$"
15CE:0133
-RCX
CX 0000
:33
-N PRIMER.COM
-W
Escribiendo 00033 bytes
-Q

C:\>PRIMER
Universidad Nacional Autónoma de México
C:\>_
```

Descripción
Este programa despliega el mensaje -> Universidad Nacional Autónoma de México

---

```
C:\>DEBUG
-A
15CE:0100 MOV AH,2
15CE:0102 MOV DL,7
15CE:0104 INT 21
15CE:0106 INT 20
15CE:0108
-RCX
CX 0000
:8
-N BEEP.COM
-W
Escribiendo 00008 bytes
-Q

C:\>_
```

Descripción.
Este programa produce un BEEP.

```
C:\>DEBUG S_N.COM
No se encontró el archivo

-A 100
16C9:0100    MOV AH, 0
16C9:0102    INT 16
16C9:0104    MOV AL, 0
16C9:0106    CMP AH, 1F
16C9:0109    JE 112
16C9:010B    CMP AH, 31
16C9:010E    JE 114
16C9:0110    JMP 100
16C9:0112    MOV AL, 1
16C9:0114    MOV AH, 4c
16C9:0116    INT 21
16C9:0118
-R CX
CX 0000
: 18
-W
Escribiendo 00018 bytes
-Q
```

Descripción
S_N espera que el usuario teclee una S o una N. Si teclea S se regresa un valor = 1.
El scancode de

	Decimal	Hexadecimal
"S"	31	1F
"N"	49	31

3.2 USO DE LAS INSTRUCCIONES

El conjunto de instrucciones puede ser dividido de acuerdo a su propósito en:

Manipulación de Bits

Estas instrucciones manipulan los bits dentro de cada registro, reflejando la operación en el registro de banderas. la ejecución de un programa puede cambiar dependiendo del resultado.

Control de Banderas y Procesador

Este grupo contiene instrucciones que afectan el registro de banderas.

Transferencia de Datos

Este grupo de instrucciones es utilizado para mover datos. El movimiento puede ser entre registros, entre registros y memoria o entre localidades de memoria.

Transferencia de Control

Estas instrucciones son utilizadas para cambiar el orden en el cual un programa es ejecutado. Afectan directamente al registro IP, cargando una dirección diferente, para que la ejecución continúe en otra localidad diferente.

Aritméticas

Esta clasificación de instrucciones incluye las operaciones matemáticas básicas.

Manipulación de Cadenas

Este grupo de operaciones opera con cadenas dentro de la memoria. Son instrucciones que manipulan bloques contiguos de memoria.

A continuación se describen algunas instrucciones, dando la siguiente información:

Categoría de la Instrucción,
Nombre de la Instrucción
Sintaxis
Operación

3.2.1 MANIPULACIÓN DE BITS

AND

Logical AND

AND *destino, fuente*

Operación:

Ejecuta un **AND** lógico en los operandos destino y fuente, colocando el resultado en el operando destino.

Fuente	*Destino*	*Destino*
1	1	1
1	0	0
0	1	0
0	0	0

Ejemplo:

```
MOV AX,FFFFH
AND AX,1111H
```

Solución

```
FFFFH   1111 1111 1111 1111

1111H   0001 0001 0001 0001
1111H   0001 0001 0001 0001
```

NOT

Logical NOT

NOT *destino*

Operación

Cambia los bits del operando.

destino	destino
1	0
0	1

Ejemplo:

```
MOV AX,1010H
NOT AX
```

Solución

```
1010H   0001 0000 0001 0000
EFEFH   1110 1111 1110 1110
```

OR

Logical Inclusive OR

OR *destino, fuente*

Operación

Fuente	Destino	Destino
1	1	1
1	0	1
0	1	1
0	0	0

Ejemplo:

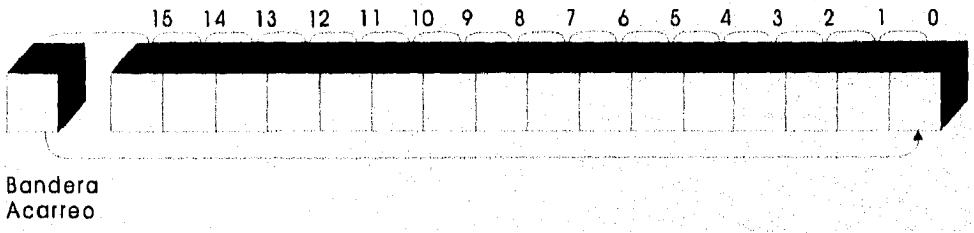
```
MOV AX,1234H
OR  AX,1111H
```

Solución

```
AX=1234H   0001 0010 0011 0100
     1111H   0001 0001 0001 0001
AX=1335H   0001 0011 0011 0101
```

RCL
Rotate Left Through Carry

RCL *destino, contador*



RCL ROTAMIENTO DE BITS A LA IZQ A TRAVÉS DE LA BANDERA DE ACARREO

Operación:

Se utiliza para rotar los bits del operando destino a la izquierda el número de veces especificado en el operando contador, el cual debe ser 1 o el registro CL. La instrucción RCL utiliza la bandera de acarreo como una extensión del operando destino, trasladando su valor al bit menos significante del operando destino y remplazándolo con del bit más significante del operando destino.

Ejemplo:

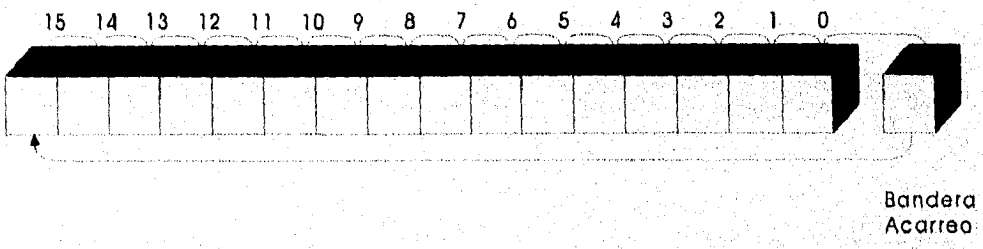
```
CLC
MOV  AX, AAAAH

RCL   AX, 1
```

Solución

```
CF = 0
AX = AAAAH    1010  1010  1010  1010

AX = 5554H    0101  0101  0101  0100
CF = 1
```

RCR
Rotate Through Carry Right

RCR *destino , contador*



RCR ROTAMIENTO DE BITS A LA IDER A TRAVÉS DE LA BANDERA DE ACARREO

Operación:

Se utiliza para rotar los bits del operando destino a la derecha el número de veces especificado en el operando contador, el cual debe ser 1 o el registro CL. La instrucción RCL utiliza la bandera de acarreo como una extensión del operando destino, trasladando su valor al bit más significante del operando destino y remplazándolo con del bit menos significante del operando destino.

Ejemplo:

```
CLC
MOV AX, AAAAH

RCR  AX, 1
```

Solución

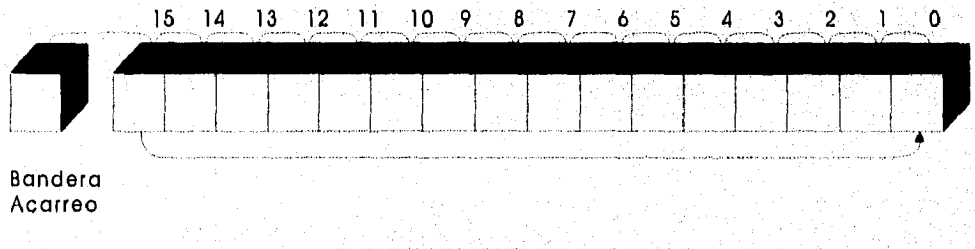
CF = 0
AX = AAAAH 1010 1010 1010 1010

AX = 5555H 0101 0101 0101 0101
CF = 0

ROL

Rotate Left

ROL *destino, contador*



ROL ROTAMIENTO DE BITS A LA IZQUIERDA

Operación:

Se utiliza para rotar los bits del operando destino a la izquierda tantas veces como se especifica en el operando contador. El bit más significativo del operando destino es copiado al bit menos significativo, así como en la bandera de acarreo.

Ejemplo:

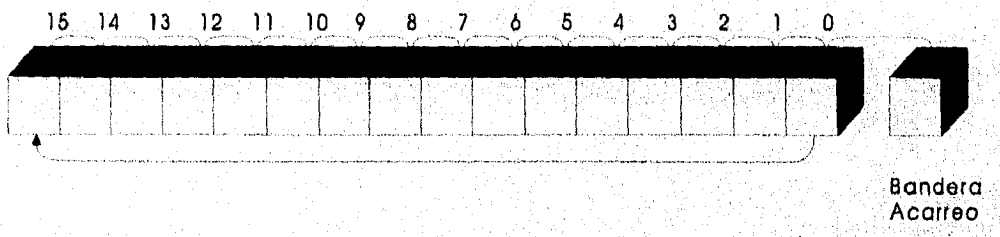
```
CLC
MOV AX, 7601H

ROL  AX, 1
```

Solución

```
CF = 0
AX = 7601H   0111  0110  0000  0001
AX = ECO2H   1110  1100  0000  0010
CF = 0
```

ROR
Rotate Right

ROR *destino, contador*



ROR ROTAMIENTO DE BITS A LA DERECHA

Operación:

Se utiliza para rotar los bits del operando destino a la izquierda tantas veces como se especifica en el operando contador. El bit menos significativo del operando destino es copiado al bit más significativo, así como en la bandera de acarreo.

Ejemplo:

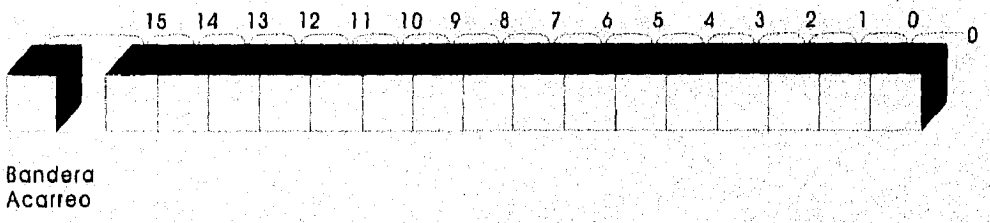
```
CLC
MOV AX, 806EH

ROR  AX, 1
```

Solución

```
CF = 0
AX = 806EH    1000 0000 0110 1110

AX = 4037H    0100 0000 0011 0111
CF = 0
```

SAL	SHL
Shift Arithmetic	Left Shift Logical Left

SAL *destino, contador*
SHL *destino, contador*



SAL SHL TRASLADO DE BITS HACIA LA IZQUIERDA

Operación

Se utiliza para trasladar los bits del operando destino a la izquierda, tantas veces como se especifica en el operando contador. Cada vez que se mueve a la izquierda una posición, hay un 0 en el bit menos significativo del operando destino.

Ejemplo:

```
CLC
MOV  AX, 0002H

SHL   AX, 1
```

Solución

```
CF = 0
AX = 0002H    0000  0000  0000   0010

AX = 0004H    0000  0000  0000   0100
CF = 0
```

SAR

Shift Arithmetic

SAR *destino, contador*

SAR TRASLADO DE BITS HACIA LA DERECHA

Operación

Se utiliza para trasladar los bits del operando destino a la derecha, tantas veces como se especifica en el operando contador. Cada vez que se mueve a la derecha una posición, el bit más significativo retiene su valor original, por lo que se conserva su signo.

Ejemplo:

```
CLC
MOV  AX, 9003H

SAR   AX, 1
```

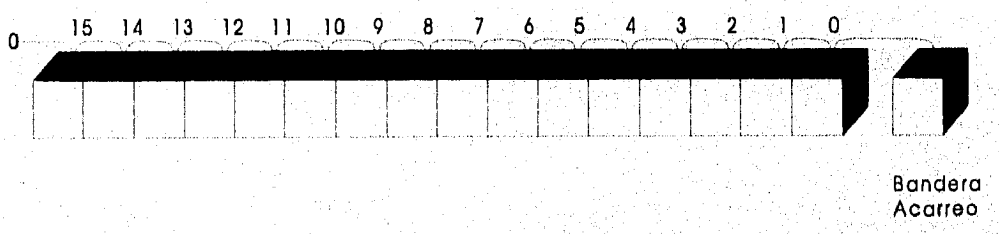
Solución

```
CF = 0
AX = 9003H    1001  0000  0000   0011

AX = C801H    1100  1000  0000   0001
CF = 1
```

SHR

Right Shift Right Logical

SHR *destino, contador*



SHR TRASLADO DE BITS HACIA LA DERECHA

Operación:

Se utiliza para trasladar los bits del operando destino a la derecha, tantas veces como se especifica en el operando contador, Cada vez que se mueve a la derecha una posición, hay un 0 en el bit más significativo del operando destino.

Ejemplo:

```
CLC
MOV  AX, 0010H

SHR   AX, 1
```

Solución

```
CF = 0
AX = 0010H    0000  0000  0001   0000

AX = 0008H    0000  0000  0000   1000
CF = 0
```

TEST

TEST by logical AND

TEST *destino, fuente*

Operación:

Ejecuta un **AND** lógico bit por bit del operando destino con el operando fuente e inicializa las banderas de acuerdo al resultado.

XOR

Logical Exclusive OR

XOR *destino, fuente*

Operación:

Ejecuta un **OR** exclusivo con el operando fuente y el operando destino almacenando el resultado en el operando destino.

Fuente	*Destino*	*Destino*
1	1	0
1	0	1
0	1	1
0	0	0

Ejemplo:

```
MOV AX, 1234H
XOR AX, 1111H
```

Solución

```
AX = 1234H   0001  0010  0011  0100
     1111H   0001  0001  0001  0001
AX = 0325H   0000  0011  0010  0101
```

3.2.2 CONTROL DE BANDERAS Y PROCESADOR

CLC

Clear Carry Flag

CLC

Operación:

Limpia bandera de acarreo.

CLD
Clear Direction Flag

CLD

Operación:

Limpia bandera de dirección.

CLI
Clear Interrupt Flag

CLI

Operación:

Limpia bandera de interrupción.

CMC
Complement Carry Flag

CMC

Operación:

Complementa bandera de acarreo.

HLT
Halt

HLT

Operación:

El microprocesador detiene la ejecución y CS:IP apunta a la instrucción que sigue.

LOCK
Lock Bus

LOCK

Operación:

Prohibe la interferencia de otros procesadores durante la ejecución de la siguiente instrucción.

NOP
No operation

NOP

Operación:

Toma espacio y tiempo, hace que el CPU no haga nada.

STC
Set Carry Flag

STC

Operación:

Activa la bandera de acarreo sin importar su estado

STD
Set Direction Flag

STD

Operación:

Activa la bandera de dirección sin importar su estado

STI
Set Interrupt Flag

STI

Operación:

Activa la bandera de interrupción sin importar su estado

WAIT

WAIT

Operación:

Ocasiona que el CPU espere una interrupción externa en la línea de TEST antes de continuar

3.2.3 TRANSFERENCIA DE DATOS

IN
Input from port

IN acumulador, puerto

Operación:

Carga un byte/word de la dirección específica del puerto de E/S a AL o AX.

Si el número de puerto es menor o igual a 256 el operando puerto puede estar como constante o en DX

Ejemplo:

```
IN    AX, 2FH
```

Si el número es mayor que 256 debe estar en DX

Ejemplo:

```
MOV  DX, 3FCH
IN    AX, DX
```

LAHF
Load AH from Flags

LAHF

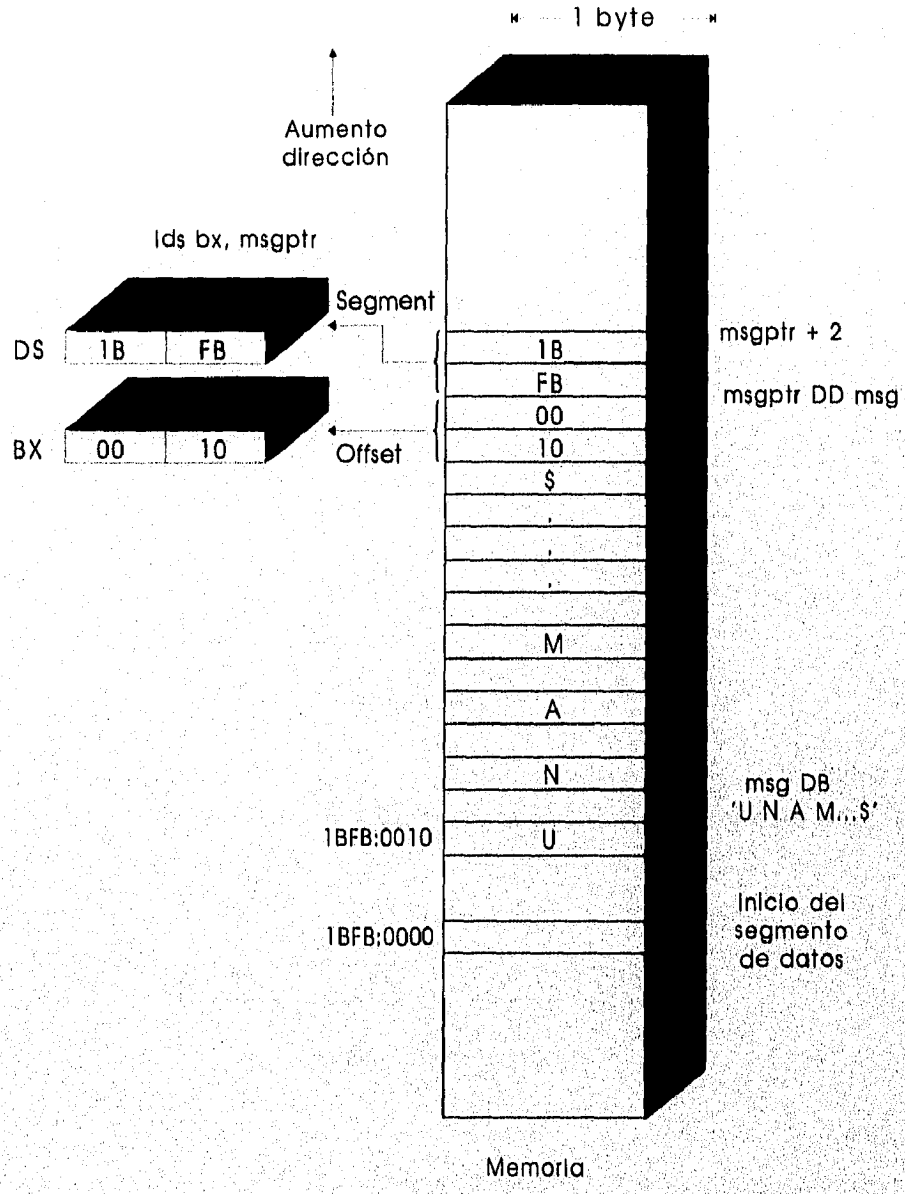
Operación:

Carga el byte bajo del registro de las banderas al registro AH (ZF, AF, PF, y CF).

LDS

Load Data Segment Register

LDS *destino, fuente*



LDS CARGAR EL REGISTRO DS

Operación:

Carga DS con la dirección de segmento del operando fuente y carga operando destino con la dirección desplazamiento del operando fuente

LEA
Load Effective Address

LEA *destino, fuente*

Operación:

Transfiere la dirección de desplazamiento del operando fuente al operando destino

LES
Load Extra Segment Register

LES *destino, fuente*

Operación

Carga ES; con la dirección de segmento del operando fuente y carga el operando destino con la dirección desplazamiento del operando fuente

MOV
Move

MOV *destino, fuente*

Operación

Copia el contenido del operando fuente en el operando destino

OUT
Output to port

OUT *puerto, acumulador*

Operación:

Envía un byte (AL) o un word (AX) a la dirección del puerto de E/S

Si el número de puerto es menor o igual a 256 el operando puerto puede estar como constante o en DX

Ejemplo:

```
OUT    2FH,AX
```

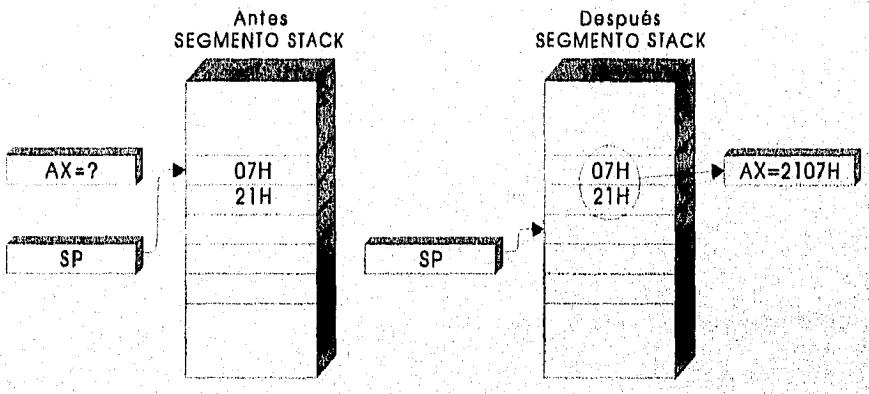
Si el número es mayor que 256 debe estar en DX

Ejemplo:

```
MOV  DX, 3FCH
OUT  DX, AL
```

POP

POP *destino*



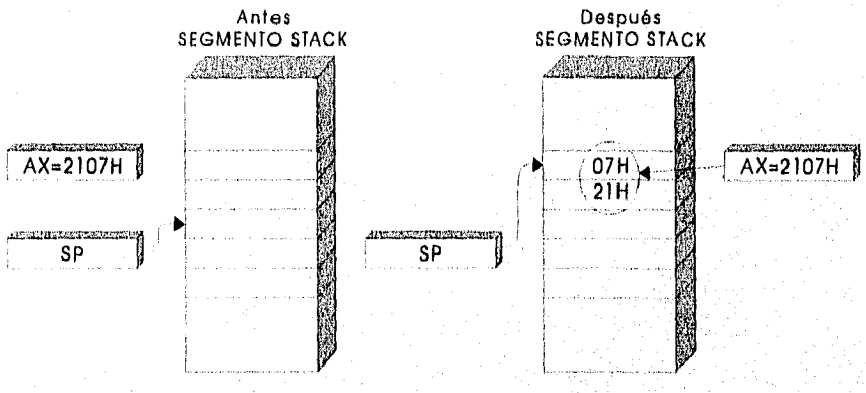
POP USO DE LA PILA

Operación

Traslada una palabra de la pila y la coloca en el destino.

PUSH

PUSH *fuente*



PUSH USO DE LA PILA

Operación:

Coloca en la pila una copia del valor del operando.

POPF

POPF

Operación:

Quita un word de la pila y la coloca en el registro de banderas, después se incrementa el registro SP en 2.

PUSHF

PUSHF

Operación:

Se decrementa en dos el registro SP y coloca en la pila una copia del registro de banderas.

SAHF
Store AH In Flags

SAHF

Operación:

Copia AH en el byte menos significativo del registro de banderas.

XCHG
Exchange

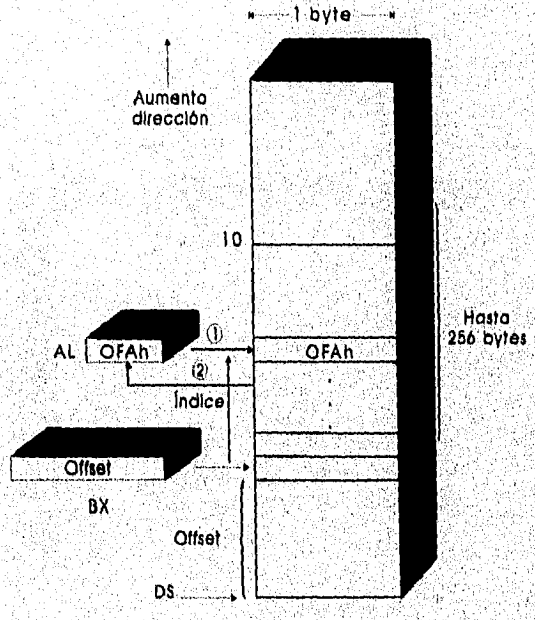
XCHG *destino, fuente*

Operación:

Intercambia el contenido de los operandos fuente, destino.

XLAT
Translate

XLAT
XLAT *segmento:desplazamiento*

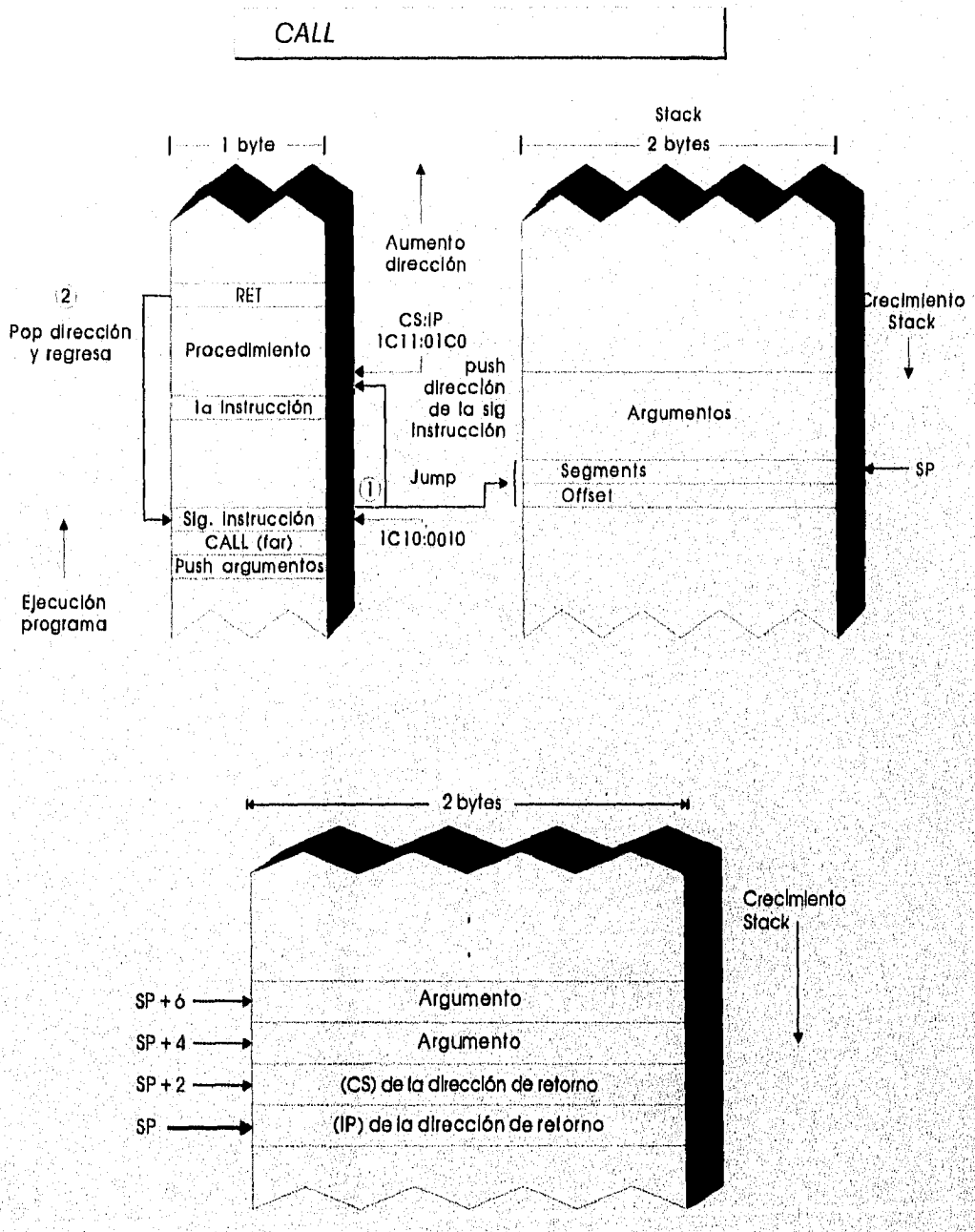


XLAT TRADUCCIÓN POR MEDIO DE UNA TABLA

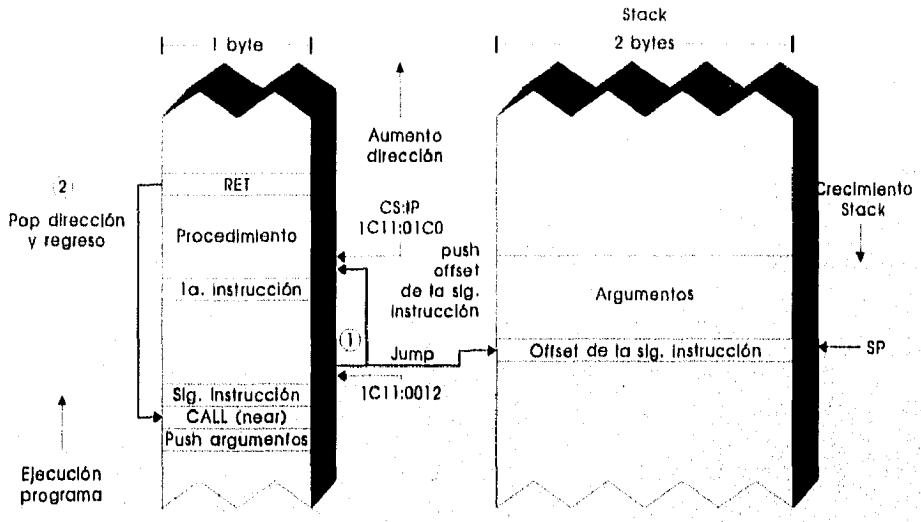
Operación:

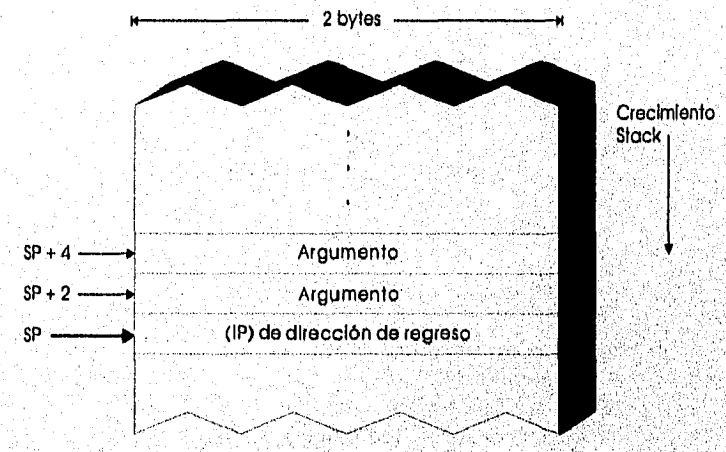
Traduce AL en otro valor por una búsqueda de tabla. Utiliza el valor de AL como un índice dentro de una tabla que empleza en la dirección DS:BX y copia su contenido del byte de esa entrada en AL.

3.2.4 TRANSFERENCIA DE CONTROL



CALL (FAR) LLAMADA DE PROCEDIMIENTOS





CALL (NEAR) LLAMADA DE PROCEDIMIENTOS

Operación:

Ejecuta una subrutina.

1. Mete la dirección de desplazamiento de la siguiente instrucción a la pila.

2. Si el procedimiento llamado se declara como FAR mete la dirección de segmento de la siguiente instrucción en la pila.

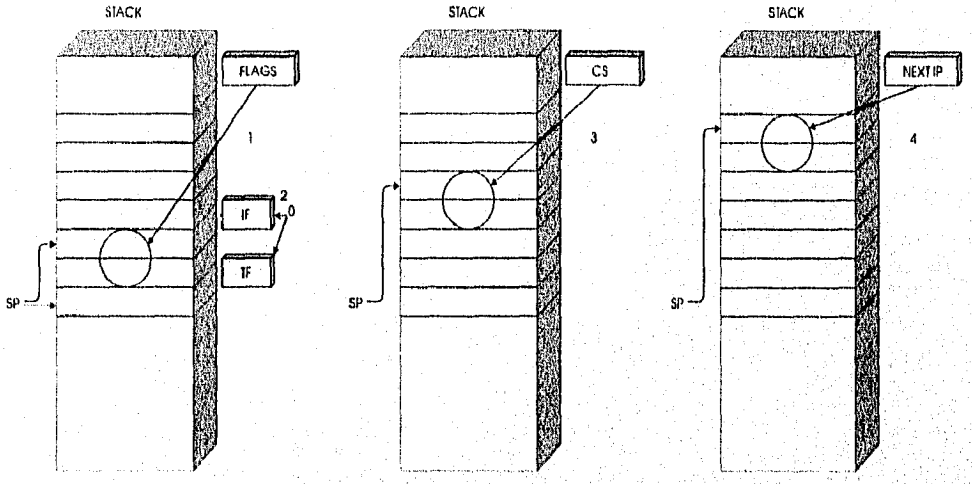
3. Carga IP con la dirección de desplazamiento del procedimiento llamado.

4. Si el procedimiento llamado se declara como FAR carga CS con la dirección de segmento del procedimiento llamado.

5. La ejecución continua en CS:IP hasta RET.

INT
Software Interrupt

INT *número de interrupción*



INT INICIO DE UNA INTERRUPCIÓN DE SOFTWARE

Operación:

Inicia una interrupción de software.

1. Mete las banderas en la pila.
2. Limpia las banderas de TF e IF.
3. Mete el valor de CS a la pila.
4. Carga CS con la dirección segmento interrupción.
5. Mete el valor de IP en la pila.
6. Carga IP con la dirección desplazamiento de la interrupción.
7. La ejecución continua en CS:IP hasta IRET.

INTO
Interrupt on OverFlow

INTO

Operación:

Se utiliza para generar la interrupción de Hardware número 4 si la bandera de overflow es 1.

IRET
Return from Interrupt

IRET

Operación:

Termina un procedimiento de interrupción y regresa el control al punto que ocurrió la interrupción (saca IP y CS de la pila).

RET
Return from Subrutine

RET

Operación:

Al sacar IP transfiere el control al punto donde se hizo el CALL, si CALL llama a un procedimiento FAR se sacan de la pila IP y CS.

LOOP

LOOP *etiqueta*

Operación:

Decrementa el registro CX y continuar la ejecución donde se encuentra la etiqueta, hasta que CX tenga un valor de 0.

LOOPE	*LOOPZ*
Loop while equal	*Loop while zero*

LOOPE *etiqueta*

Operación:

Decrementa el registro CX y continuar la ejecución donde se encuentra la etiqueta, hasta que CX tenga un valor de 0 y la bandera ZF de 0.

LOOPNE	*LOOPNZ*
Loop while not equal	*Loop while not zero*

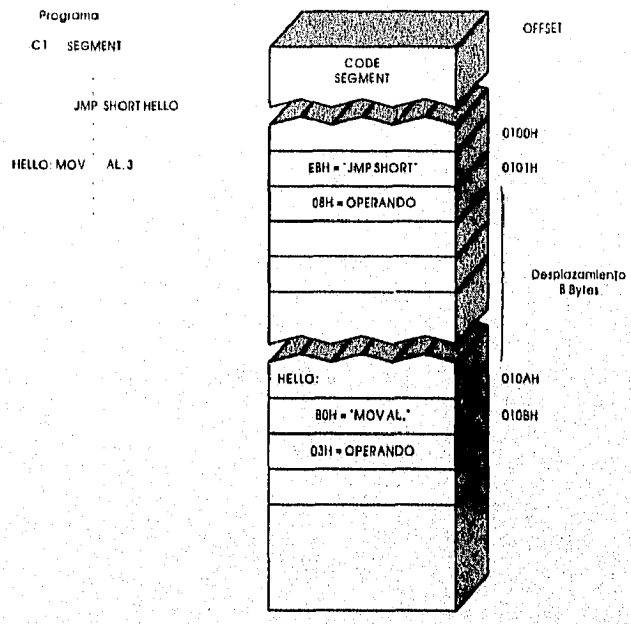
LOOPNE *etiqueta*

Operación:

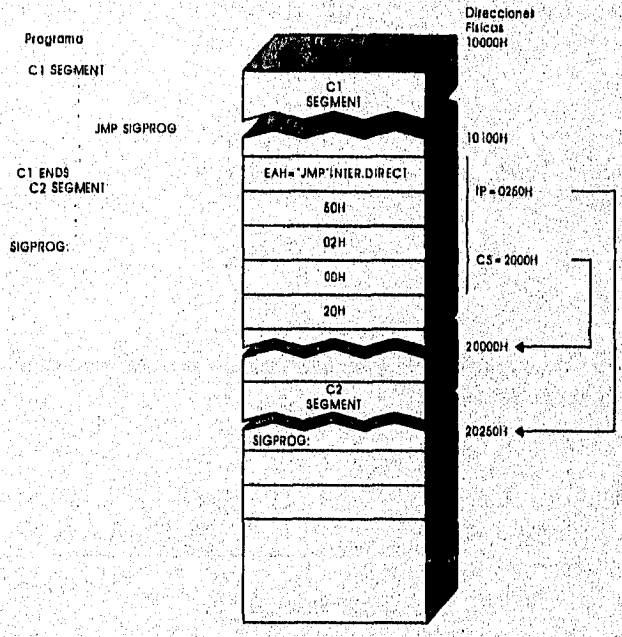
Decrementa el registro CX y continuar la ejecución donde se encuentra la etiqueta, hasta que CX tenga un valor de 0 y la bandera ZF de 1.

JMP

Jump



JMP EJECUCIÓN DE SALTO A OTRA DIRECCIÓN DENTRO DEL MISMO SEGMENTO



JMP EJECUCIÓN DE SALTO A OTRA DIRECCIÓN EN UN SEGMENTO DIFERENTE

Operación:

Se utiliza para ejecutar un salto incondicional a la dirección especificada.

Instrucción			Condición	Jump If...
JA	JNBE		(CF or ZF) = 0	Above Not below or Equal
JAE	JNB	JNC	CF = 0	Above or Equal Not Below Not Carry
JB	JNAE	JC	CF = 1	Below Not Above nor Equal Carry
JBE	JNA		(CF or ZF) = 1	Below or Equal Not Above
JCXZ			CX = 0	CX Igual a Cero
JE	JZ		ZF = 1	Equal Zero
JG	JNLE		((SF XOR OF) OR ZF) = 0	Greater Not Less or Eaual
JGE	JNL		(SF XOR OF) = 0	Greater or Equal Not Less
JL	JNGE		(SF XOR OF) = 1	Less Not Greater or Equal
JLE	JNG		((SF XOR OF) OR ZF) = 1	Less or Equal Not Greater
JNE	JNZ		ZF = 0	Not Equal Not Zero
JNO			OF = 1	Not OverFlow
JNP	JPO		PF = 0	Not Parity
JNS			SF = 0	Not Sign
JO			OF = 1	Overflow
JP	JPE		PF = 1	Parity Parity Even
JS			SF = 1	Sign

3.2.5 ARITMÉTICA

AAA
ASCII Adjust for Adition

AAA

Operación:

Se utiliza después de sumar dos dígitos BCD para convertir el resultado (AL) en un dígito BCD.

AAD
ASCII Adjust before division

AAD

Operación:

Convierte dos dígitos BCD en AH y AL, en un número binario en AL. Después de **AAD**, AL va a contener 10 veces su valor más el valor de AH y AH va a ser 0.

AAM

ASCII Adjust AX after multiplication

AAM

Operación:

Convierte un valor binario (menor que 100) en AL a dos dígitos BCD en AH y AL, con el bit menos significativo en AL, y el dígito más significativo en AH.

AAS

ASCII Adjust AL after Substraction

AAS

Operación:

Se utiliza para corregir el resultado de restar un dígito BCD de otro.

ADC

Add with carry

ADC *destino, fuente*

Operación:

Suma los operandos fuente y destino con la bandera de acarreo y almacena el resultado en el operando destino.

ADD

Addition

ADD *destino, fuente*

Operación:

Suma los operandos fuente y destino y almacena el resultado en el operando destino.

CBW
Convert byte to word

CBW

Operación:

Convierte el valor del byte en AL a un valor de word en AX, extendiendo el valor del bit de alto orden de AL a través de AH.

CMP
Compare

CMP *destino, fuente*

Operación:

Resta el operando fuente del operando destino e inicializa las banderas de acuerdo al resultado. Los operandos permanecen sin cambios.

CWD
Convert word to doubleword

CWD

Operación:

Convierte el valor de word en AX a un valor double word en DX:AX, extendiendo el valor del bit de alto orden de AX a través de todos los bits de DX.

DAA
Decimal adjust for addition

DAA

Operación:

Se utiliza después de utilizar la instrucción ADD o la instrucción ADC.
Si el nibble más bajo es mayor que 9 o si la bandera AF = 1, suma 6 a ese nibble.
Si el nibble más alto es mayor que 9 o si la bandera CF = 1, suma 6 a ese nibble.

Ejemplo

Hex	BCD
29	0010 1001
+ 18	+ 0001 1000
41	0100 0001 AF = 1
+ 6	+ 0110
47	0100 0111

DAS

Decimal adjust for substraction

DAS

Operación:

Se utiliza después de utilizar la instrucción SUB o la instrucción SBB.
Si el nibble más bajo es mayor que 9 o si la bandera AF = 1, resta 6 a ese nibble.
Si el nibble más alto es mayor que 9 o si la bandera CF = 1, resta 6 a ese nibble.

Ejemplo

Hex	BCD
2B	0010 1000
- 19	- 0001 1001
0F	0000 1111 AF = 1
- 6	- 0110
9	0000 1001

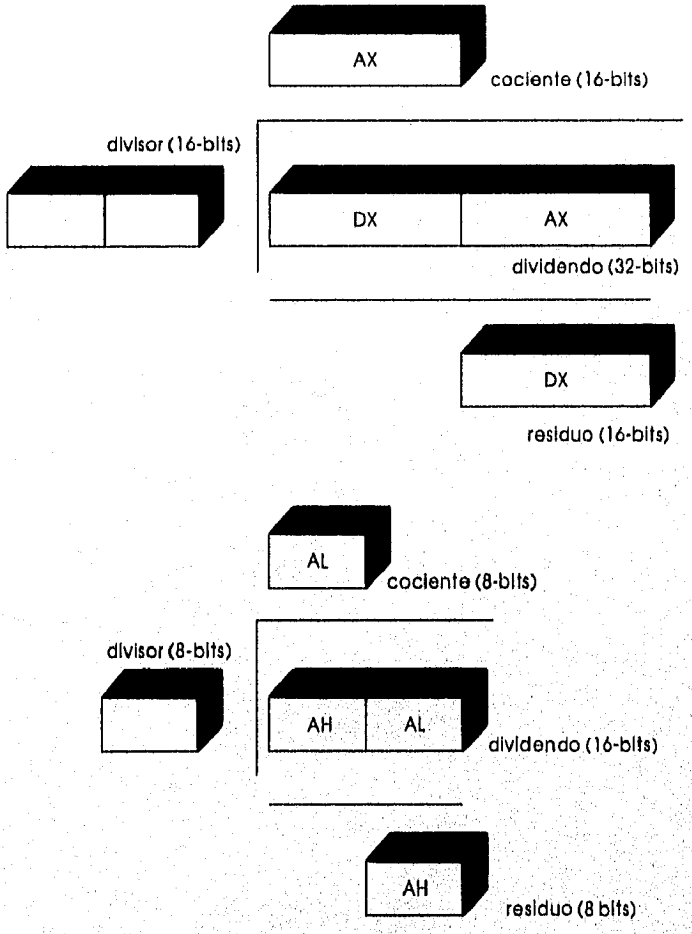
DEC

Decrement destination by one

DEC *operando*

Operación:

Resta 1 del operando.



DIV DIVISIÓN CON DIVIDENDO DE 16 Y 8 BITS

DIV
Division

DIV *divisor*

Operación:

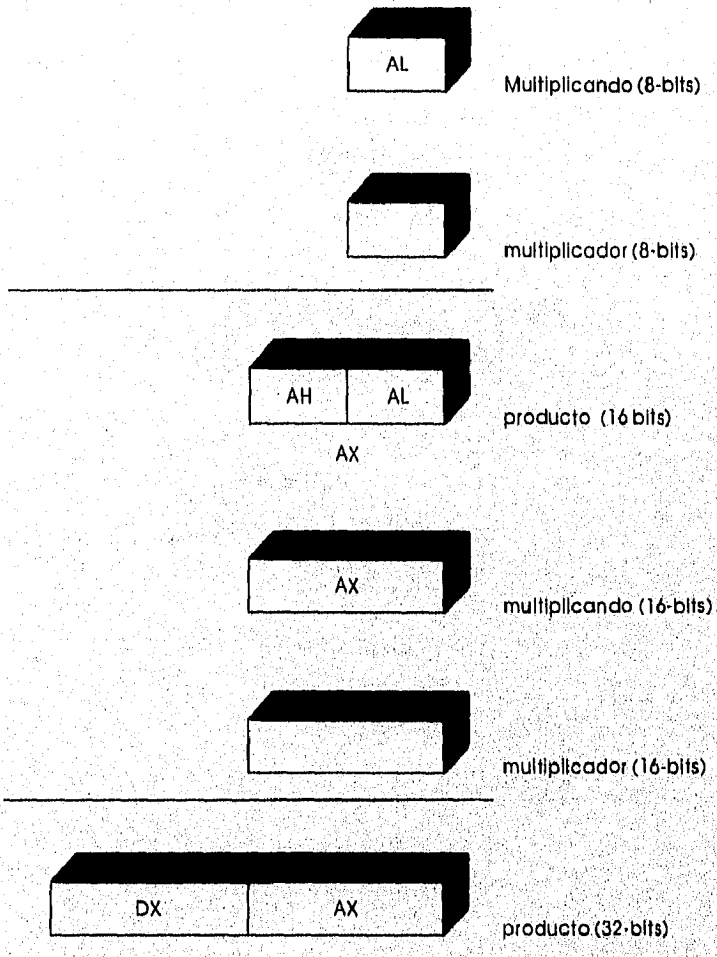
Divide un número sin signo en AX o DX:AX por el divisor especificado. Cuando el divisor es un word, el cociente se queda en AX y el residuo en DX. Si el divisor es un byte, el cociente en AL y el residuo en AH.

IDIV
Integer divide

IDIV *divisor*

Operación:

Divide un número con signo en AX o DX:AX por el divisor especificado. Cuando el divisor es un word, el cociente se queda en AX y el residuo en DX. Si el divisor es un byte, el cociente en AL y el residuo en AH.



MUL MULTIPLICACIÓN COM OPERANDOS DE 16 Y 8 BITS

MUL
Multiply

MUL *operando*

Operación:

Multiplica un número sin signo en AL o AX por el operando especificado. Si el operando es un byte, el resultado se queda en AX. Cuando el operando es un word, el producto esta en los registros DX:AX, con los 16 bits de alto orden en DX.

IMUL
Integer multiply

IMUL *operando*

Operación

Multiplica un número con signo en AL o AX por el operando especificado. Si el operando es un byte, el resultado se queda en AX. Cuando el operando es un word, el producto esta en los registros DX:AX, con los 16 bits de alto orden en DX.

INC
Increment

INC *operando*

Operación

Suma 1 al operando.

NEG
Negate

NEG *operando*

Operación:

Se utiliza para negar el operando. Calcula el complemento a dos.

SBB
Substract with borrow

SBB *destino, fuente*

Operación:

Suma la bandera de acarreo (0 ó 1) al operando fuente y resta el resultado del operando destino.

SUB
Substract

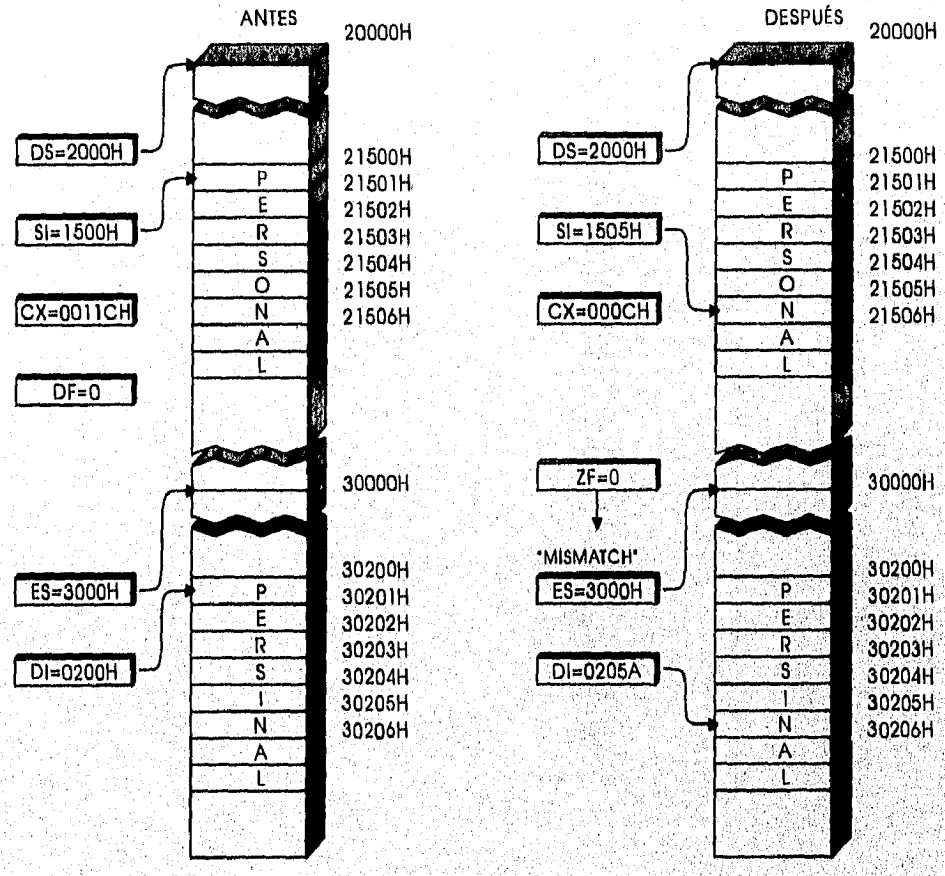
SUB *destino, fuente*

Operación:

Resta el operando fuente del operando destino y almacena el resultado en el operando destino.

3.2.6 MANIPULACIÓN DE CADENAS

CMPSB	*CMPSW*
Compare strings byte for byte	*Compare strings word for word*

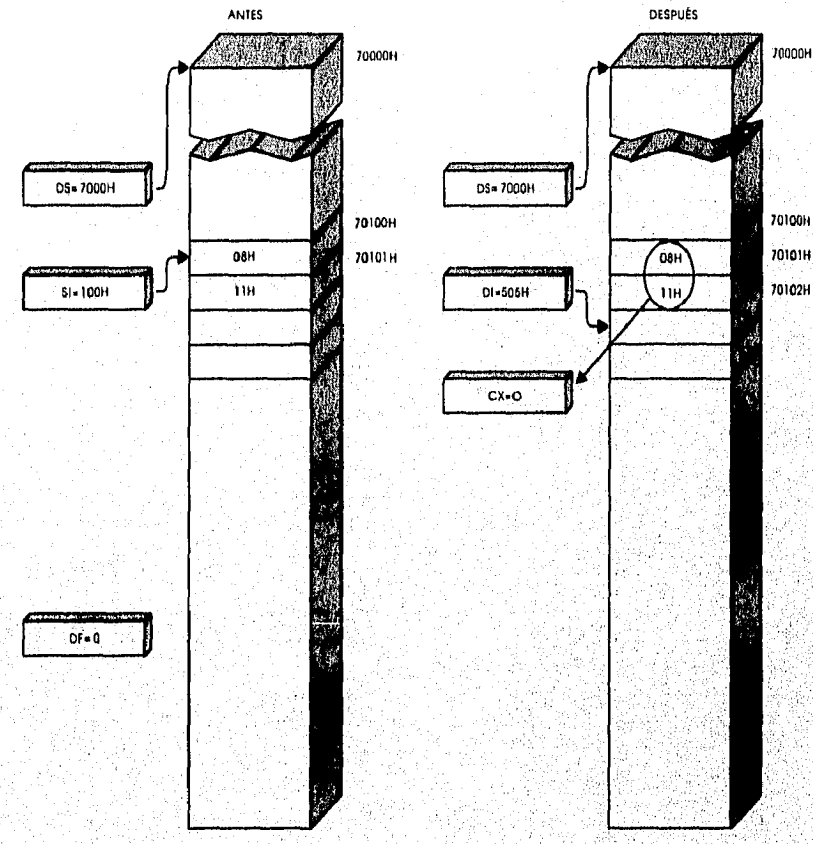


COMPARACIÓN DE CADENAS

Operación:

Compara un byte / word de la cadena fuente DS:SI con la cadena destino ES:DI restando el elemento de ES:DI del DS:SI. EL resultado de la resta no se almacena en algún lugar; sólo las banderas se alteran dependiendo del resultado. Después de la comparación, SI y DI son ajustados por el tamaño del elemento, incrementándose si DF = 0 ó decrementándose si DF = 1.

LODSB	*LODSW*
Load byte from string into AL	*Load word from string into AX*

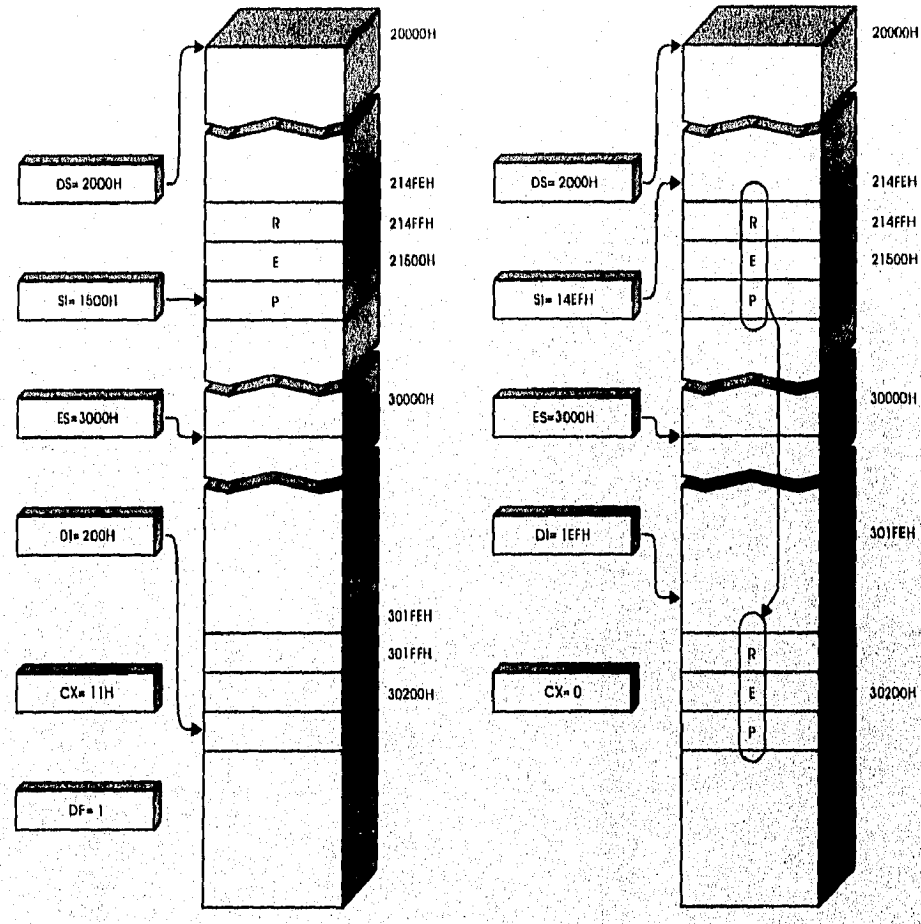


OBTENCIÓN DE UN VALOR DE LA CADENA

Operación:

Carga un byte / word de la cadena fuente DS:SI en AL o AX. Después de la transferencia, si DF = 0 se incrementa SI el tamaño del operando. De otra manera, se decrementa SI.

MOVSB	*MOVSW*
Move strings byte for byte	*Move strings word for word*

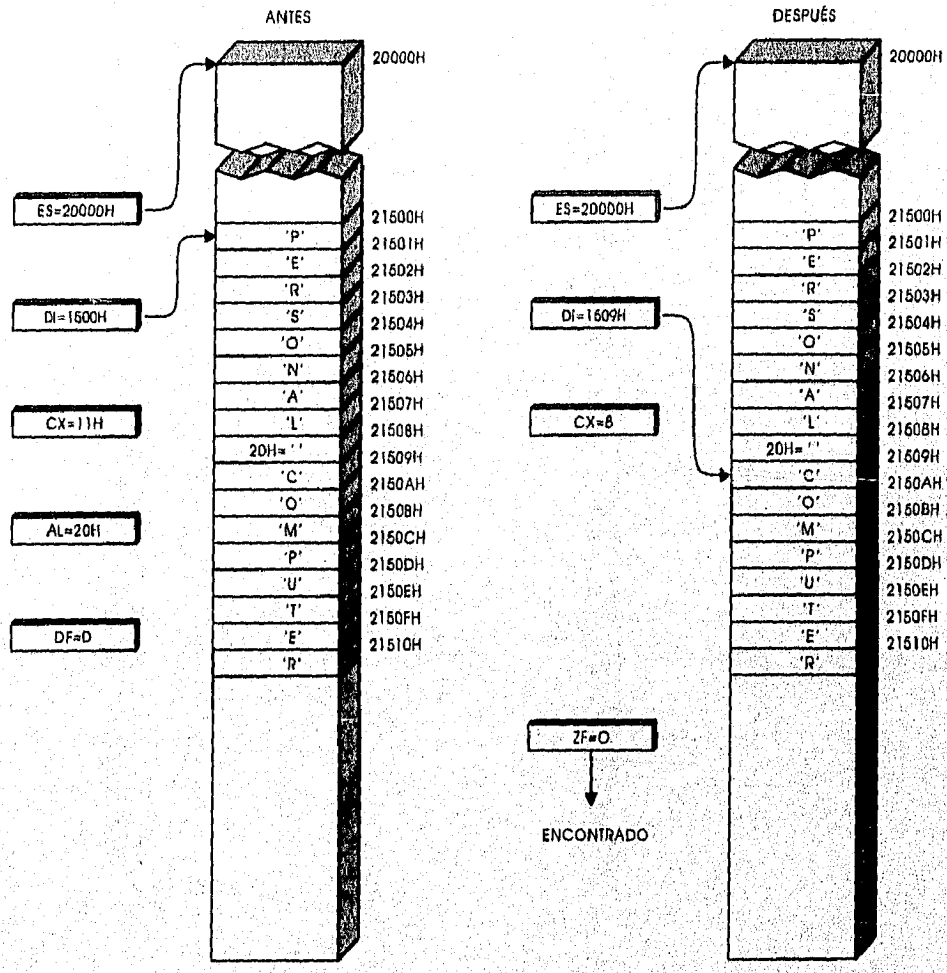


MOVIMIENTO DE CADENAS

Operación:

Copia un byte / word de la cadena fuente DS:SI al destino ES:DI. Después de que el elemento es copiado, si DF = 0 se incrementa SI y DI el tamaño del elemento,. De otra manera, se decrementa SI y DI.

SCASB	*SCASW*
Scan string for byte	*Scan string for word*

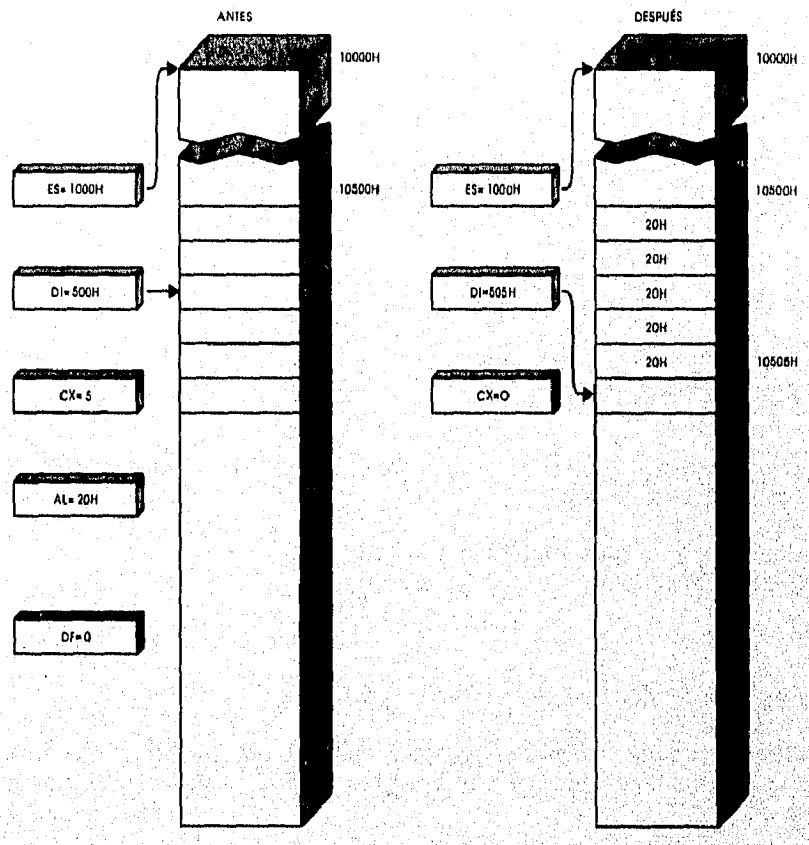


BÚSQUEDA DE VALORES EN CADENAS

Operación:

Resta un byte / word de la cadena destino ES:DI del valor de AL ó AX. si DF = 0 se incrementa DI el tamaño del operando. De otra manera, se decrementa DI.

STOSB	*STOSW*
Store strings byte for byte	*Store strings word for word*



ALMACENAMIENTO DE CADENAS

Operación:

Carga AL o AX en un byte / word de ES:DI. Después de la transferencia, si DF = 0 se incrementa DI el tamaño del operando. De otra manera, se decrementa DI.

REP

REP *instruccion*

Operación:

Repite la siguiente instrucción el número de veces especificado en CX.

REPZ	*REPE*

Operación:

Repite la siguiente instrucción mientras ZF = 1 y CX sea mayor que 0.

REPNZ	*REPNE*

Operación:

Repite la siguiente instrucción mientras ZF =0 y CX sea mayor que 0.

3.3 USO DE INTERRUPCIONES

El procesador debe estar disponible para los eventos que pueden ocurrir en tiempos no predecibles, que pueden ser originados por Hardware:

Las señales del teclado (que una tecla fue oprimida)
Las señales de la impresora (que no tiene papel)

y por software

Un programa este tratando de realizar un división por cero

El procesador esta diseñado para manejar estos eventos como vayan ocurriendo. La computadora puede reconocer una señal que le indique que un evento requiere de su atención, esta señal es llamada interrupción.

Es importante mencionar que las interrupciones pueden ocurrir en cualquier momento y no hay forma de predecir que va a estar ejecutando el procesador cuando ocurra una. Por lo que el mecanismo de interrupción provee dos funciones:

Deshabilitar el procesador para que no reconozca interrupciones.
Atender todas las interrupciones recibidas.

Después de que se detecta una interrupción, el procesador ejecuta una rutina. Cada tipo de interrupción requiere su propia rutina.

Desde el punto de vista del procesador, empezar una rutina de una interrupción difiere de empezar un procedimiento llamado. La principal diferencia en que antes de ir al principio de la rutina de la interrupción, el procesador debe guardar las banderas y la dirección de la siguiente instrucción del programa actual. Cuando termina, el procesador recupera las banderas y sigue con el programa que fue interrumpido.

Cuando una interrupción ocurre el procesador debe identificar el tipo de interrupción y la dirección de la rutina correspondiente.

El DOS mantiene una tabla con direcciones apuntando a cada rutina. Está almacenada en las localidades de memoria más bajas, de 0000H a 03FFH (1024 bytes), como una dirección requiere de cuatro bytes, la tabla tiene 256 interrupciones diferentes, aunque muchas no se utilicen.

Los 256 vectores de interrupción posibles están numerados desde 0H a FFH (0 a 256), la dirección 0000H almacena la dirección de la rutina de la interrupción 0, la 0004H la dirección de la 1, y así sucesivamente.

Cuando una interrupción ocurre el procesador encuentra la dirección de la rutina correspondiente, multiplicando el vector por 4.

Esta tabla de direcciones se inicializa cuando el DOS se carga, es posible cambiar estas direcciones para realizar interrupciones propias, las interrupciones tienen dos propósitos:

permitir al procesador responder a los necesidades del Hardware
permitir a un programa requerir servicios del DOS y del BIOS.

Una interrupción que se origina desde un dispositivo de Hardware es llamada interrupción de Hardware, y una interrupción que se origina desde un programa que se está ejecutando es llamada interrupción de software.

BIOS

El BIOS es una serie de programas complejos que se almacenan en la memoria ROM. Provee entre otras funciones, un conjunto de servicios los cuales son requeridos por las interrupciones de software.

DOS

Además de ser el programa de control principal que se ejecuta en la computadora, provee servicios que son requeridos por las interrupciones de software.

3.3.1 SERVICIOS DEL DOS

DOS provee cerca de 130 servicios agrupados en la interrupción número 21H

ENTRADA Y SALIDA DE CARACTERES

Servicio	
1H	Entrada por el teclado con eco
2H	Salida de caracter en monitor
3H	Entrada asincrona por un dispositivo auxiliar
4H	Salida asincrona de caracter
5H	Salida de caracter por impresora
6H	Consola E/S
7H	Entrada por el teclado sin eco
8H	Entrada por el teclado sin eco
9H	Imprimir cadena
AH	Entrada de cadena
BH	Obtener estado de teclado
CH	Reinicialización buffer de entrada y llamada función de entrada de teclado

OPERACIONES DE ARCHIVO

Servicio	
FH	Abrir archivo
10H	Cerrar archivo
11H	Buscar el nombre de un archivo
12H	Buscar el siguiente
13H	Borrar archivo(s)
16H	Crear / abrir archivo
17H	Renombrar archivo
23H	Obtener tamaño de archivo
29H	Análisis de nombre de archivo
3CH	Crear un archivo
3DH	Abrir un archivo
3EH	Cerrar un archivo
41H	Borrar un archivo
43H	Cambiar atributos de archivo
45H	Duplica file handle
46H	Forza duplicación de File handle
4EH	Buscar el primer nombre de archivo que concuerda con el patrón
4FH	Buscar el siguiente nombre de archivo
56H	Renombrar un archivo
57H	Obtener o configurar la fecha de los archivos
5AH	Crear archivos temporales
5BH	Crear un nuevo archivo

OPERACIONES DE REGISTRO

Servicio	
14H	Lectura secuencial
15H	Escritura secuencial
1AH	Especificar DTA (disk transfer addres)
21H	Lectura aleatoria
22H	Escritura aleatoria
24H	Inicializa un campo registro aleatorio
27H	Lectura por bloque aleatoria
28H	Escritura por bloque aleatoria
2FH	Obtener DTA (disk transfer area)
3FH	Lectura de un archivo o un dispositivo
40H	Escritura de un archivo o un dispositivo
42H	Mover apuntador de escritura/lectura

OPERACIONES DE DIRECTORIO

Servicio	
39H	Crear un subdirectorio
3AH	Remover un subdirectorio
3BH	Cambiar el subdirectorio actual
47H	Obtener directorio actual en un drive específico

MANEJO DE DISCO

Servicio	
DH	Reinicialización de disco
EH	Inicializar un drive predeterminado
19H	Obtener drive actual
1BH	Obtener FAT (file allocation table)
36H	Obtener espacio libre del disco
54H	Revisar si el disco esta disponible o no para escritura

MANEJO DE PROCESOS

Servicio	
0H	Fin de programa
26H	Crea un nuevo PSP (Program Segment Prefix)
31H	Termina el proceso y se mantiene residente
4BH	Cargar y ejecutar un programa
4CH	Termina un proceso
4DH	Obtiene el código de retorno de un subproceso
62H	Obtener la dirección del PSP

MANEJO DE MEMORIA

Servicio	
48H	Asignación de memoria
49H	Liberar memoria asignada
4AH	Modifica el tamaño de bloques de memoria asignada

FUNCIONES DE REDES

Servicio	
5E00H	Obtener nombre de la máquina
5E02H	Configurar impresora
5E03H	Obtener configuración de impresora
5F02H	Obtener lista de de redirecciones
5F03H	Redireccionar salida de impresora
5F04H	Cancela redirección

HORA Y FECHA

Servicio	
2AH	Obtener la fecha de sistema
28H	Inicializa la fecha de sistema
2CH	Obtener la hora del sistema
2DH	Inicializa la hora del sistema

SISTEMA

Servicio	
25H	Inicializa vector de interrupción
30H	Obtener versión de DOS
33H	Control Ctrl-break
35H	Obtener dirección de vector de interrupciones
38H	Información del país
44H	Control de E/S

CAPÍTULO IV

TÓPICOS AVANZADOS

CAPÍTULO IV

TÓPICOS AVANZADOS

4.1 PUERTO PARALELO

Las señales del puerto paralelo para impresión son:

Nombre	Dirección	Número de pin
Bit de dato 0	Impresora	2
Bit de dato 1	Impresora	3
Bit de dato 2	Impresora	4
Bit de dato 3	Impresora	5
Bit de dato 4	Impresora	6
Bit de dato 5	Impresora	7
Bit de dato 6	Impresora	8
Bit de dato 7	Impresora	9
Strobe	Impresora	1
Impresora en línea	Impresora	13
Avance de línea	Impresora	14
Inicializar impresora	Impresora	16
Reconocimiento	Computadora	10
Ocupado	Computadora	11
Falta de papel	Computadora	12
Impresora en línea	Computadora	13
Error	Computadora	15

4.1.1 OPERACIÓN DE PUERTO PARALELO

Cuando se va a imprimir se colocan los datos en las 8 líneas de bits de datos y la línea de strobe recibe la señal. La impresora procesa el dato y envía de regreso la señal de acknowledge. Cuando la computadora recibe esta señal, envía el siguiente caracter a la impresora. La computadora puede verificar si esta ocupada la impresora, en lugar de esperar la señal de acknowledge, y enviar caracteres a la impresora mientras no esté ocupada. La línea de error le dice a la computadora que la impresora está fuera de línea, que le falta papel u otro estado de error.

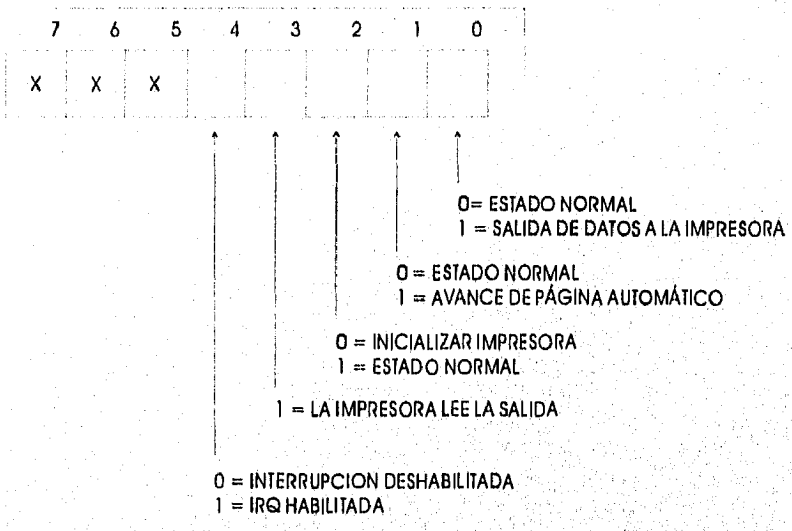
Ciertas impresoras reconocen la señal de avance de línea. Esta señal puede ser utilizada para forzar a la impresora generar un avance de línea cada vez que se imprima un control de retorno de carro.

Direcciones de puertos para el control de la impresora.

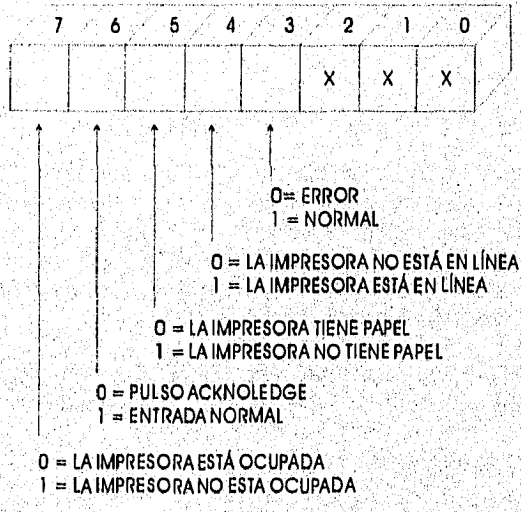
Función	Direcciones
Datos	378H
Control	37AH
Estado	379H

Cuando el dato es enviado por el puerto 378H, en el puerto de control 37AH el bit 3 debe estar siempre igual a 1, si se requiere que la impresora no genere avance de línea automático el bit 1 debe ser igual a 0, el bit 4 es para habilitar las interrupciones de impresora. Si este bit es igual a 1, cuando el adaptador reciba la señal de acknowledge, se generaré una petición al IRQ7, esta interrupción puede ser utilizada para decirle al programa que la impresora está lista para recibir otro caracter.

Cuando la impresora es inicializada el bit 2 es igual a 0, por lo menos 50 microsegundos, y después otra vez es igual a 1.Cuando se envía un dato, el bit de strobe debe ser igual a 1 e inmediatamente cambiarlo a 0, lo que causa que la impresora lea el caracter del puerto.



PUERTO DE CONTROL (37AH)



PUERTO DE CONTROL (379)

```
imprime  Segment
         assume cs:imprime, ds:imprime
         org    100h
         Datos  =  378h
         Status =  379h
         Control=  37ah

inicio   Proc     near
         mov      si,offset mensaje
         cld
         mov      cx,FinMen - Mensaje
ciclo:   lodsb
         call     print
         loop     ciclo
         int      20h
Mensaje  db       'Uso de puertos de I/O para puerto paralelo',13,10,13,10
         db       'Universidad Nacional Autónoma de México',13,10
         db       15,'ENEP  Acatlán',18,13,10
FinMen   label    byte
inicio   Endp

print    proc     near
         mov      dx,datos
         out      dx,al
repite:  mov      dx,status
         in       al,dx
         mov      dx,offset msjerr
         test     al,00001000b
         jz       Error
         mov      dx,offset msjonl
         test     al,00100000b
         jz       Error
         mov      dx,offset msjoop
         test     al,00100000b
         jz       Error
         test     al,10000000b
         jz       repite
         mov      dx,Control
         mov      al,1101b
         out      dx,al
         mov      al,1100b
         out      dx,al
         ret
Error:   mov      ah,9
         int      21h
         int      20h
msjerr   db       13,10,'Error en la Impresora',13,10,'$'
msjonl   db       13,10,'Impresora fuera de linea',13,10,'$'
msjoop   db       13,10,'Impresora sin papel',13,10,'$'
print    endp
imprime  ENDS
         END      inicio
```

4.2 PUERTO SERIAL

4.2.1 COMUNICACIONES SERIALES.

La principal característica de una comunicación serial es que todos los bits de datos y de control para recibir y transmitir se deben de mover un bit a la vez sobre la línea de datos.

La velocidad de transmisión determina el espacio entre cada bit. La velocidad se mide en bits por segundo, y se llama baud rate. Si la velocidad de transmisión es de 2400 baud, se están transmitiendo 2400 bits por segundo.

Cuando se desea transmitir un caracter, el primer bit enviado es el bit de inicio. El receptor sabe que tiene que esperar un caracter cuando recibe el bit de inicio.

Después del bit de inicio se envía el caracter de información, y opcionalmente les sigue un bit de paridad, que permite al receptor detectar cierto tipos de errores, por lo que debe ser consistente en toda la transmisión. Si la paridad es par, el número de bits igual a 1 que conforman el dato incluyendo el de paridad debe ser par, si es impar el número debe ser impar.

Después del bit de paridad se envía el bit de paro, si existe otro caracter debe seguir después del bit de paro con su correspondiente bit de inicio.

EL UART

El Receptor Transmisor Universal Asíncrono (UART Universal Asynchronous Receiver Transmitter) es un microprocesador, utilizado para realizar comunicaciones seriales.

Antes de que el UART pueda ser utilizado se tiene que dar los parámetros deseados de velocidad, del número de bits, del tipo de paridad y del número de bits de parada.

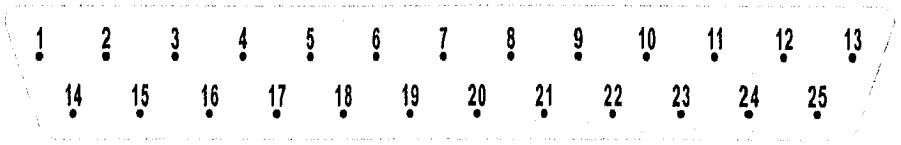
Cuando se quiere transmitir un caracter, se revisa el estado del UART para verificar que el registro de retención de transmisión (Transmitter Holding Register) esté vacío y colocar el byte del dato en ese registro. Cuando no se está transmitiendo algún dato, el contenido del registro de retención de transmisión se pasa al registro de transmisión (Transmitter Shift Register).

El UART añade los bits apropiados de inicio, paridad y el de paro, después envía una cadena entera de bits.

El UART coloca cualquier entrada serial que recibe en el registro de recepción (Receiver Shift Register) . Después de que los bits de parada son recibidos y de que se verifica si no existe algún error, el caracter es colocado en el registro de recepción de datos (Receiver Data Register). El UART indica que el dato recibido está listo para que sea leído por la computadora.

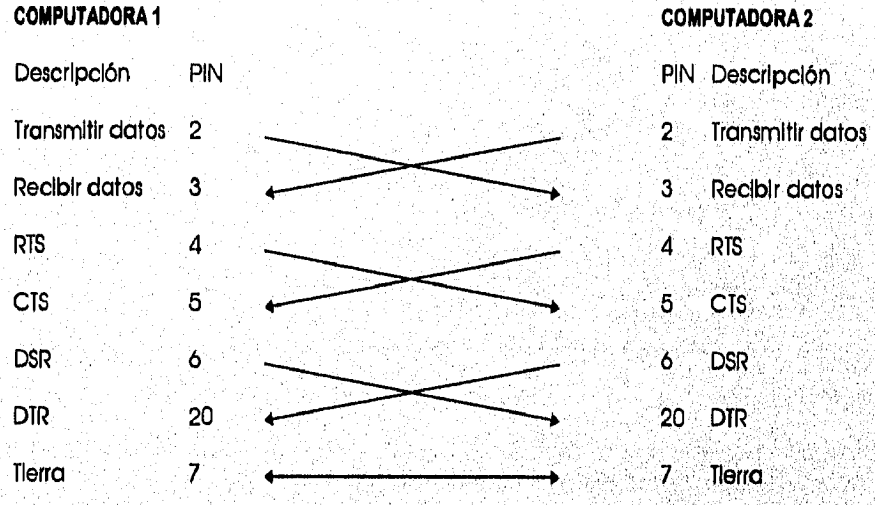
INTERFAZ RS232

La interfaz RS232 conecta el UART con el mundo exterior, es una interfaz de 25 cables entre una computadora y un dispositivo periférico. Es una norma para transmisiones en serie que utiliza conectores DB25 de 25 pins:



No. Pin	Dirección	Función	
2	Salida	Datos transmitidos	
3	Entrada	Datos recibidos	
4	Salida	Petición para enviar	RTS
5	Entrada	Limpia para enviar	CTS
6	Entrada	Datos colocados listos	DSR
7		Señal de tierra	
8	Entrada	Detector portador de datos	
20	Salida	Terminal de datos lista	DTR
22	Entrada	Indicador de alerta	

En distancias cortas es posible conectar directamente dos interfaces RS232, se utiliza un esquema de conexión llamado de módem nulo.



Para programar el UART es necesario conocer las direcciones asociadas con sus registros:

Dirección del puerto	E/S	Registro
3F8H	S	Registro de retención de transmisión
	E	Registro de recepción de datos
3F9H	S	Registro habilitar interrupción
3FAH		Registro de identificación de interrupción
3FBH	S	Registro de control de línea
3FCH	S	Registro de control de módem
3FDH	E	Registro de estado de línea
3FEH	E	Registro de estado del módem

El registro de control de estado de línea es utilizado para enviar o recibir datos. Si se desea transmitir un caracter se debe leer este registro y verificar el bit 5, ya que un caracter no puede ser guardado en el registro de retención de transmisión hasta que el bit 5 sea igual a 1. Cuando se envía el caracter el bit 5 vuelve a ser 0.

Cuando el bit 0 es igual a 1 significa que un caracter ha sido recibido y que ha sido colocado en el registro de recepción de datos, el cual debe ser leído antes de recibir otro caracter. Cuando se lee el caracter el bit 0 vuelve a ser 0.

Esqueleto del programa con variables y mensajes:

```
Dialogo   Segment
          assume  cs:Dialogo, ds:Dialogo
          org     100h
Inicio    proc    near
          jmp     ppal


pStat     = 03fdh
pDato     = 03f8h
fin       = 01
Return    = 13
SaltoL    = 10
llama     = 0
respo     = 1

modo      db      ?
intro1    db      'Dialogo PC<->PC',13,10,13,10,'$'
intro2    db      '(^A) para terminar',13,10,'$'
uso1      db      'Uso:',13,10,13,10
          db      '    Dialogo d',13,10,13,10
          db      '     donde d=(1,2), una PC con d=1 la otra d=2',13,10,'$'

;                 PROCEDIMIENTOS

Inicio    endp
Dialogo   ends
          end     Inicio
```

PROCEDIMIENTOS

Tx Envía un byte por serial dado en AL
Lógica: Espera hasta que se enciende el bit 5 de pStat
Transmite el byte

```
Tx        proc    near
          push    ax
          mov     dx,pStat
Espera:   in      al,dx
          test    al,20h
          jz      Espera
          mov     dx,pDato
          pop     ax
          out     dx,al
          ret
Tx        endp
```

Rx Recibe un byte por serial, presupone que está listo
Lógica: Lee el byte de pDato y lo coloca el AL

```
Rx      proc    near
        mov     dx,pDato
        In      al,dx
        ret
Rx      endp
```

RxRdy Verifica si ha llegado un byte por el serial.
Lógica: Verifica bit 0 de pStat y define la bandera de acarreo así:
CF Dato pendiente
NC Nada pendiente

```
RxRdy  proc     near
       clc
       mov      dx,pStat
       In       al,dx
       test     al,1
       Jz       NoRdy
       stc
NoRdy: ret
RxRdy  endp
```

Envía Envía el byte en AL por serial con protocolo llama responde.
Lógica: Envía byte por serial
Si está en modo responde genera eco local, si es return genera salto de línea

```
Envia   proc    near
        call    Tx
        cmp     modo,llama
        je      SalirE
        call    print
        cmp     al,return
        jne     SalirE
        push    ax
        mov     al,saltoL
        call    Tx
        call    print
        pop     ax
SalirE: ret
Envia   endp
```

Recv Recibe el byte en AL por serial con protocolo llama responde
Lógica: Recibe un caracter
Si es modo RESPONDE genera eco remoto

```
Recv  proc      near
      call      Rx
      cmp       modo,llama
      je        SaleR
      call      Tx
      cmp       al,return
      jne       SaleR
      push      ax
      mov       al,saltoL
      call      Tx
      call      print
      pop       ax
SaleR: ret
Recv  endp
```

ChkIn Verifica Status de entrada del serial.
Lógica: Si hay un caracter disponible la lee.
Genera un eca local.

```
ChkIn   proc    near
        call    RxRdy
        jnc     NoIn
        call    Recv
        call    print
NaIn:   ret
ChkIn   endp
```

ChkOut Verifica estado de salida por teclado para enviar por serial.
Lógica: Si hay caracter listo por teclado lo lee.
La envía al serial.

```
ChkOut proc  near
       mov   ah,1
       int   6h
       jz    NoOut
       mov   ah,0
       call  envia
NoOut: ret
ChkOut endp
```

Print Imprime en pantallla el caracter en AL.

```
print   proc    near
        mov     ah,0eh
        mov     bl,0
        int     19h
        ret
print   endp
```

Ppal. EL programa principal define modo llama o modo responde según los parámetros .

```
Ppal    proc    near
        mov     ax,3
        int     10h
        mov     ah,9
        mov     dx,offset intro1
        int     21h
        cmp     byte ptr ds:80h,2
        jne     Uso
        cmp     byte ptr ds:82h,'1'
        je      Ok
        cmp     byte ptr ds:82h,'2'
        je      Ok
Uso:    mov     ah,9
        mov     dx,offset uso1
        int     21h
        int     20h
Ok:
        mov     modo,llama
        cmp     byte ptr ds:82h,'2'
        jne     sigue
        mov     modo,respo
sigue:  mov     ah,9
        mov     dx,offset intro2
        int     21h
        mov     dx,pDato
        in      al,dx
        mov     al,return
Repite: call    Chkln
        Call    ChkOut
        cmp     al,Fin
        jne     Repite
        int     20h
Ppal    endp
```

4.3 MANEJO DE PANTALLA

En esta sección se examinará la manera en que se pueden escribir y leer caracteres en la pantalla así como la forma de manipular el cursor.

La pantalla es una cuadrícula de localidades direccionables. Un monitor típico tiene 25 renglones (numerados de 0 a 24) y 80 columnas (numerados de 0 a 79).

El sistema provee un espacio en la memoria para el buffer de la pantalla. Un monocromático empieza en la localidad B0000 y soporta 4k bytes de memoria, de los cuales 2k son para los caracteres y 2k para los atributos de cada caracter.

En un monitor de color soporta 16k bytes de memoria empezando en la dirección B8000. Puede estar en modo texto o en modo gráfico. Para modo texto, el buffer provee 4 páginas para una pantalla de 80 columnas (4k) y 8 páginas para una pantalla de 40 columnas (2k).

MODO TEXTO
El modo texto es similar en un monitor monocromático y en uno de color, excepto que el monitor de color no soporta el atributo de subrayado. El modo texto provee soporte al ASCII extendido (256 caracteres).

Atributo
En modo texto, un atributo determina las características de cada caracter que va a ser desplegado. Para cada caracter en la pantalla, existen dos bytes almacenados en la memoria. El primer byte es el código ASCII del caracter. El segundo byte es el atributo de dicho caracter.

Atributos de un Monitor Monocromático
En la mayoría de los monitores monocromáticos, el fondo es negro y los caracteres son de color verde o ambar

	fondo				caracter			
7	6	5	4	3	2	1	0	Atributo
	0	0	0		0	0	0	Nada
	1	1	1		1	1	1	Un cuadro blanco
	0	0	0		1	1	1	Caracter blanco, fondo negro (Video normal)
	0	0	0		0	0	1	Caracter blanco, fondo negro subrayado
	1	1	1		0	0	0	Caracter negro, fondo blanco (Video inverso)

Bit 7 = 1 Intermitente
= 0 normal

Bit 3 = 1 Alta Intensidad
= 0 Intensidad normal

Atributos de un Monitor de color

Valor	Color
1	Caracter Azul
2	Caracter Verde
4	Caracter Rojo
8	Intensidad alta
16	Fondo Azul
32	Fondo Verde
64	Fondo Rojo
128	Intermitente

El BIOS y DOS provee servicios para manipular la pantalla:

Servicios de **BIOS**

INT 10H
Servicio **00** Inicializar modo de video

Esta operación puede cambiar el modo de un monitor de acuerdo a la siguiente tabla:

Modo	**Texto/Gráfico**	**Mono/Color**
00	Texto (25 x 40)	Mono
01	Texto (25 x 40)	Color
02	Texto (25 x 80)	Mono
03	Texto (25 x 80)	Color
04	Gráfico	Color
05	Gráfico	Mono
06	Gráfico	Mono
07	Texto (25 x 80)	Mono

INT 10H
Servicio 1 Inicializar tipo de cursor

INT 10H
Servicio **2** Inicializar posición del cursor

Coloca el cursor en cualquier parte de la pantalla de acuerdo a sus coordenadas.

INT 10H
Servicio **3** Posición y tamaño del cursor

Este servicio para determinar el renglón, la columna, y el tamaño del cursor.

INT 10H
Servicio **6** Ventana hacia arriba

INT 10H
Servicio **7** Ventana hacia abajo

INT 10H
Servicio **8** Leer caracter y atributo en la posición del cursor

Permite leer el caracter y el atributo de la pantalla.

INT 10H
Servicio **9** Escribir caracter y atributo en la posición del cursor

Este servicio despliega caracteres en modo texto o gráfico con el atributo deseado.

INT 10H
Servicio **A** Escribir un caracter en la posición del cursor

Escribe caracteres en modo texto y en modo gráfico, la única diferencia entre el servicio 9H y el servicio AH, es que este servicio utiliza el atributo actual.

INT 10H
Servicio **E** Escribir caracter en modo TTY

Esta operación permite utilizar el monitor como terminal.

INT 10H
Servicio **F** Obtener modo de video

Esta operación regresa el modo, los caracteres por línea y el número de página.

Servicios de **DOS**

INT 21H
Servicio **2** Salida de caracter en monitor

Este servicio imprime un caracter en la pantalla y avanza el cursor

INT 21H
Servicio **6** Consola E/S

Este servicio imprime un caracter en la pantalla

INT 21H
Servicio **9** Imprimir cadena

Este servicio envía a la pantalla una serie de caracteres que se encuentran en una dirección específica.

INT 21H
Servicio **40** Escritura de un archivo o un dispositivo

Este servicio puede ser utilizado para escribir en la pantalla (File Handle = 1 -> pantalla).
El siguiente programa realiza un marco en la pantalla:

```
codigo  segment
        org 80h
nombre  db ?
        db ?
xo      dw ?
        db ?
yo      dw ?
        db ?
xn      dw ?
        db ?
yn      dw ?
        db ?
        org 100h
        assume cs:codigo,ds:codigo
        ren   dw ?
        col   dw ?

ppal    proc near
        mov     ax,0B800H
        mov     es,ax
        mov     ax,xo
        call    asb_1
        mov     xo,ax
        mov     ax,yo
        call    asb_1
        mov     yo,ax
        mov     ax,xn
        call    asb_1
        mov     xn,ax
        mov     ax,yn
        call    asb_1
        mov     yn,ax
        call    maroo
        mov     ah,4CH
        int     21h
ppal    endp
```

```
marco   proc near
        call    contador
        mov     ax,yo           ; posición esq sup izq
        mov     bl,80
        mul     bl
        add     ax,xo
        shl     ax,1
        mov     bx,ax
        mov     dx,ax
        mov     al,218          ; esq sup izq
        mov     es:(bx),al
        call    desren
        mov     al,192          ; esq inf izq
        mov     es:(bx+di),al
        add     bx,2
        mov     al,196
        mov     cx,col

h:      mov     es:(bx),al      ; línea horizontal
        mov     es:(bx+di),al
        add     bx,2
        loop    h
e:      mov     al,191          ; esq sup der
        mov     es:(bx),al
        mov     al,217          ; esq inf der
        mov     es:(bx+di),al
        mov     bx,dx           ; pos esq izq
        add     bx,160
        call    descol
        mov     al,179
        mov     cx,ren
v:      mov     es:(bx),al
        mov     es:(bx+di),al
        add     bx,160
        loop    v
        ret
marco   endp

contador proc near
        mov     ax,xn
        sub     ax,xo
        dec     ax
        mov     col,ax          ; xn-xo-1
        mov     ax,yn
        sub     ax,yo
        dec     ax
        mov     ren,ax          ; yn-yo-1
        ret
contador endp
```

```
                                        .
                                        .
                                        .
desren  proc near
        mov     ax,yn           ; desplazamiento de renglón
        sub     ax,yo
        mov     cl,160
        mul     cl
        mov     dl,ax
        ret
desren  endp

descol  proc near
        mov     ax,xn   ;       ; desplazamiento de columna
        sub     ax,xo
        shl     ax,1
        mov     dl,ax
        ret
descol  endp

asb_1   proc near
        sub     al,'0'
        sub     ah,'0'
        mov     cx,ax
        and     ah,0
        mov     bl,10
        mul     bl
        xchg    ch,cl
        and     ch,0
        add     ax,cx
        ret
asb_1   endp

codigo ends
        end ppal
```

4.4 MANEJO DEL TECLADO

En esta sección se analiza la manera de que un programa puede aceptar una entrada por el teclado.

El área de datos del BIOS en el segmento 40:00 contiene información útil que incluye:

Estado del teclado (dirección 40:17H) :

Bit	Acción
7	Cambio de estado de la tecla Insert
6	Cambio de estado de la tecla Bloq Mayús
5	Cambio de estado de la tecla Bloq Num
4	Cambio de estado de la tecla Bloq Despl
3	Tecla Alt oprimida
2	Tecla Control oprimida
1	Tecla Shift izquierda oprimida
0	Tecla Shift derecha oprimida

Estado del teclado (dirección 40:18H):

Bit	Acción
7	Tecla Insert oprimida
6	Tecla Bloq Mayús oprimida
5	Tecla Bloq Num oprimida
4	Tecla Bloq Despl oprimida
3	Tecla Control/Bloq Num oprimida
2	Tecla PetSis oprimida
1	Tecla Alt Izquierda oprimida
0	Tecla Alt derecha oprimida

4.4.1 BUFFER DEL TECLADO

En el área del datos del BIOS en la dirección 40:1EH se encuentra el buffer del teclado. Esta área permite teclear hasta 15 caracteres antes de que un programa requiera la entrada. Cuando se presiona una tecla, el procesador del teclado genera un código llamado Scan Code y realiza un llamado a la INT 09.

La interrupción del BIOS obtiene el scan code del teclado, lo convierte a caracter ASCII, y lo libera del buffer del teclado. Después la INT 16H lee el caracter del buffer y lo libera para el programa. EL programa no requiere de hacer un llamado a la INT 09, porqué el llamado ocurre inmediatamente cuando se presiona una tecla.

El buffer del teclado requiere dos direcciones: una que le dice a la INT 09 donde insertar el siguiente caracter y otra para indicarle a la INT 16H donde extraer el siguiente caracter.

Dirección del inicio del buffer	Dirección de fin del buffer	buffer
41A	41C	41E

Cuando se teclea un caracter la INT 9 la dirección de fin avanza. Cuando la INT 16H lee un caracter avanza la dirección de inicio. De esta manera se implementa una cola circular. La dirección de inicio y de fin son las mismas cuando el buffer esta vacio.

Servicios de **DOS**

INT **21H**
Servicio **01** Entrada por el teclado con eco

La operación acepta un caracter del buffer del teclado o, si no hay, espera una entrada del teclado. La operación regresa en AL el caracter ASCII y lo despliega en la pantalla o si se si AL es cero significa que se presionó una tecla de función como Inicio, F1, Av Pág. Si se repite inmediatamente este servicio se obtiene su Scan Code. La operación también responde a una petición Control-Inter

INT **21H**
Servicio **06** Entrada y salida por Consola

Para una entrada se carga DL con 0FFH, Si no existen caracteres en el buffer, la bandera de cero tiene el valor de 1 y no espera ninguna entrada. Si existe un caracter en el buffer, se almacena en el registro AL y la bandera de cero = 0. AL es cero significa que se presionó una tecla de función como Inicio, F1, Av Pág. Si se repite inmediatamente este servicio se obtiene su scan code. Esta operación no verifica una petición Control-Inter ni despliega el caracter en la pantalla.

INT **21H**
Servicio **07** Entrada por el teclado sin eco

Esta operación trabaja como el servicio 01 excepto que el caracter que entra no se despliega en la pantalla ni responde a una petición de Ctrl-Inter.

INT **21H**
Servicio **08** Entrada por el teclado sin eco

Esta operación trabaja como el servicio 01 excepto que el caracter no se despliega en la pantalla.

INT **21H**
Servicio **0AH** Entrada de cadena

INT **21H**
Servicio **0BH** Obtener estado de teclado

Esta operación regresa en el registro AL el valor de FFH si un caracter está disponible o 00H si no hay un caracter disponible

INT **21H**
Servicio **0CH** Reinicialización del buffer del teclado y entrada

La operación limpia el buffer del teclado primero y ejecuta el servicio en AL, y espera un caracter.

Servicios de **BIOS**

INT **16H**
Servicio **00** Leer del teclado

La operación verifica si existe un caracter en el buffer del teclado. Si existe regresa el registro AL el caracter y en AH el Scan Code. Si no existe, espera la entrada de un caracter. Si la tecla presionada es una función como Inicio o F1 regresa en AL el valor 0 y en AH el Scan Code.

INT **16H**
Servicio **01** Verifica si hay que leer una tecla

LA operación es similar al servicio **0**. Si existe un caracter en el buffer del teclado, la bandera de cero tiene el valor 0 , el registro AL el caracter y AL el scan code. Si no existe caracter disponible ZF=1.

INT **16H**
Servicio **02** Estado del teclado

La operación regresa en AL el estado del teclado del área de datos del BIOS

INT **16H**
Servicio **05** Escribir en buffer del teclado

Esta operación permite almacenar caracteres en el buffer del teclado como si un usuario hubiera oprimido una tecla. Carga en CH el caracter ASCII y en CL su Scan Code

El siguiente programa recibe el scan code que se genera al oprimir las teclas Alt-a, Alt-e, Alt-i, Alt-o, Alt-u e imprime á, é, í, ó, ú respectivamente.

Definiciones scan code/ASCII de teclas comunes

```
AltA    equ     1e00h   ; Alt-a
CodA    equ     160     ; á         Alt-160
AltE    equ     1200h   ; Alt-e
CodE    equ     130     ; é         Alt-130
AltI    equ     1700h   ; Alt-i
CodI    equ     161     ; í         Alt-161
AltO    equ     1800h   ; Alt-o
CodO    equ     162     ; ó         Alt-162
AltU    equ     1600h   ; Alt-u
CodU    equ     163     ; ú         Alt-163
Altz    equ     2c00h   ;Alt-z
```

Área de teclado -- Área de comunicaciones del ROM BIOS

```
Kbd     Segment at 0040h
        org     017h
KbdSt   DB      ?
        org     01ah
Pt1     DW      ?               ; Cola de la cola circular
Pt2     DW      ?               ; Frente de la cola circular
Buf     DW      16 DUP (?)      ; "buffer" de la cola circular
Kbd     EndS
```

Esqueleto del programa

```
MiKbd   Segment
        Assume  cs:MiKbd
        org     100h
IniTSR  Proc    Near
        Jmp     Instala
        DB      13,9
Mensaje DB      13,10
        DB      'MKBD--Arnoldo Salazar Aragón, PAAEUNAM.920630',13,10
        DB      'MIKBD -- Instalado.',13,10,'$',26
Kbd9    Label   word
DOS9    DD      ?

                PROCEDIMIENTOS

IniTSR  EndP
MiKBD   EndS
        End     IniTSR
```

Nueva Interrupción de teclado

```
INT9        Proc    Near
            pushf
            call    DOS9
            push    ax
            push    bx
            push    cx
            push    si
            push    di
            push    ds
            push    es
            mov     ax,Kbd
            mov     es,ax
            Assume  es:Kbd
            mov     bx,Pt1
            cmp     bx,Pt2
            jne     noSale
            jmp     sale
nosale:     mov     ax,CodA     ; á
            cmp     es:(bx),AltA
            je      Subst
            mov     ax,CodE     ; é
            cmp     es:(bx),AltE
            je      Subst
            mov     ax,Codi     ; í
            cmp     es:(bx),Alti
            je      Subst
            mov     ax,CodO     ; ó
            cmp     es:(bx),AltO
            je      Subst
            mov     ax,CodU     ; ú
            cmp     es:(bx),AltU
            je      Subst
            cmp     es:(bx),Altz
            je      Atrib
            jmp     Sale
Subst:              ; Otros casos, desecha Shift/CapsLock
            test    KbdSt,01000011B
            jnz     Sale
            mov     es:(bx),ax
            jmp     Sale
Atrib:      mov     ax,0b800h
            mov     ds,ax
            mov     di,1
            mov     ah,07
            mov     cx,2000
ciclo :     mov     ds:(di),ah
            add     di,2
            loop    ciclo
Sale:
            pop     es
            pop     ds
            pop     di
            pop     si
            pop     cx
            pop     bx
            pop     ax
            Iret
INT9        EndP
```

Rutina de instalación

```
Instala     Proc    Near
            mov     dx,offset Mensaje
            mov     ah,9
            int     21h
            mov     ax,3509h
            int     21h
            mov     Kbd9,bx
            mov     Kbd9+2,es
            mov     dx,Offset INT9
            mov     ax,2509h
            int     21h
            mov     dx,offset Instala+1
            int     27h
Instala     EndP
```

Este programa se queda residente en memoria, intercepta la INT 9 antes de que el BIOS lo haga e interpreta el scan code de un conjunto de combinaciones de teclas para ajustarlas a ciertos caracteres.

En la siguiente sección se explica la manera en que un programa se queda residente en memoria.

4.4.2 PROGRAMAS RESIDENTES EN MEMORIA

Existen programas que después de cargarlos permanecen en memoria permitiendo accesarlos desde otras aplicaciones, los cuales se les conoce como TSR (Terminate and Stay Resident).

En general un programa residente consiste en:

1. Una sección que redefine posiciones en la tabla de interrupciones.

2. Un procedimiento que se ejecute sólo la primera vez que se corra el programa y que realice las siguientes tareas:

   Reemplazar la dirección de la tabla de interrupciones con su propia dirección.
   Establecer el tamaño de la porción que se va a quedar residente.
   Utilizar una interrupción que le diga al DOS que termine el programa y que mantenga en memoria la porción especificada

3. Un procedimiento que se quede residente y que sea activado, por ejemplo, por cierta entrada de teclado o por el reloj.

El programa de la sección anterior intercepta la INT 09H, (entrada por teclado) para verificar si se ha oprimido una tecla y utiliza dos llamadas a servicios de la INT 21H para manipular la tabla de interrupciones:

INT **21H**
Servicio **35** Obtener la dirección de la interrupción.

Esta operación regresa la dirección de la interrupción en ES:BX.

INT **21H**
Servicio 25 Reemplazar la dirección de una interrupción.

Esta operación se establece el valor de la nueva dirección (DX) a la interrupción (AL)

```
Codigo      Segment
            Assume cs:Codigo
            org     100h
InITSR      Proc    Near
            Jmp     Instala

                    AREA DE DATOS

Dir_Intvieja Label  word
Entrada     DD      ?

Int_nueva     Proc          Near
              pushf
              call          Entrada
              push          ax
              push          bx
              push          cx
              push          si
              push          dl
              push          ds
              push          es

              PROGRAMA QUE SE QUEDA RESIDENTE

              pop           es
              pop           ds
              pop           dl
              pop           si
              pop           cx
              pop           bx
              pop           ax
              Iret
Int_nueva     EndP

Instala       Proc          Near

              CODIGO DE INICIALIZACION

              mov           ah,35h
              mov           al, No_Int
              Int           21h

              mov           Dir_Intvieja,bx
              mov           Dir_Intvieja+2,es

              mov           dx,Offset Int_nueva
              mov           ah,25h
              mov           al, No_Int
              Int           21h

              mov           dx,offset Instala+1
              Int           27h
Instala       EndP

InITSR        EndP
Codigo        EndS
              End           InITSR
```

Descripción

CODIGO empieza el segmento de código del programa .COM. LA primera instrucción ejecutable, JMP INITSR, transfiere el control a la rutina INITSR la cual utiliza el servicio 35H de DOS para localizar la dirección de la interrupción en la tabla de servicios. La operación regresa la dirección en ES:BX, la cual es guardada en DIR_INTVIEJA. El Servicio 25H establece la dirección de INT_NUEVA en la tabla de interrupciones.

4.5 PROGRAMACIÓN DE UN MICROPROCESADOR DE 32 BITS

En esta sección se presentan características de los Microprocesadores 386 y 486 que afectan a la programación y las diferencias más importantes entre los microprocesadores de 16 bits y los de 32 bits.

En un Microprocesador 386 y 486 se puede trabajar en dos modos:

Modo real
Modo protegido

En modo real trabaja esencialmente igual que una máquina 8086/286 con la excepción que se encuentran registros de 32 bits disponibles, En este modo se puede accesar un máximo de 1 Mb de memoria.

En modo protegido se puede accesar hasta 4 Gb de memoria, pero se requiere de un sistema operativo complejo. (UNIX y OS/2)

Una de las diferencias más importantes es el tamaño de los registros. Mientras que en un Microprocesador 8086/286 el tamaño el registro es de 16 bits, en una 386/486 el tamaño máximo se extiende a 32 bits.

Por ejemplo un 386/486 contiene registros AL, AH, y EAX con 8,8,16 y 32 bits respectivamente. En un 8088/286 se puede accesar el registro AX como AL, AH o AX, en un 386/486 sólo se puede accesar el registro EAX como AL, AH, AX o EAX, es decir, no se puede accesar el registro alto de EAX como un registro separado.

Además de los registros de segmento CS,DS,SS, y ES, existen dos registros nuevos; FS y GS, haciendo posible accesar 384 bytes (6 x 64 = 384).

Generales

Nombre	Bits
EAX,EBX,ECX,EDX	32
AX,BX,CX,DX	16
AH,AL,BH,BL,CH,CL,DH,DL	8

Apuntadores

Nombre		Bits
ESP	Extended SP	32
EBP	Extended BP	
SP	Stack Pointer	16
BP	Base Pointer	

Índices

Nombre		Bits
ESI	Extended SI	32
EDI	Extended DI	
SI	Source Index	16
DI	Destination Index	

Segmentos

Nombre		Bits
CS	Code Segment	16
DS	Data Segment	
SS	Stack Segment	
ES	Extra Segment	
FS		
GS		

Instrucción

Nombre		Bits
EIP	Extended Instruction Pointer	32

Bandera

Nombre		Bits
EFR	Extended Flag Register	32

Control

Nombre	Bits
CR0, CR1,CR2,CR3	32

Otro cambio importante es que los registros generales (EAX, ECX, EDX) pueden ser utilizados como apuntadores. En una arquitectura de 16 bits los registros AX, CX, y DX no pueden ser utilizados como apuntadores, sólo los registros BX, SI, DI y BP.

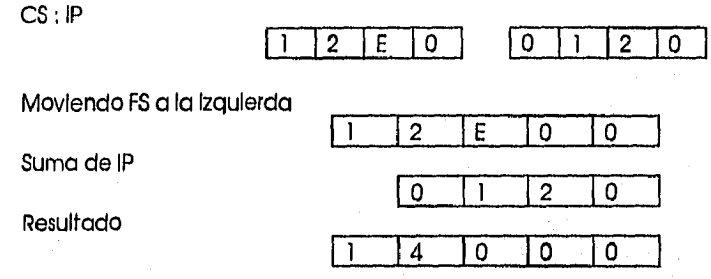
Segmentos por default y override

Cuando se utiliza EAX, ECX, o EDX como direcciones el registro por default es DS, para ESP y EBP es SS, para EIP es CS, y para los demás registros es DS. El caracter : puede utilizarse para cambiar el registro del segmento por default.
Dirección Física

El cálculo de la dirección física en modo real es el mismo que en una arquitectura de 16 bits.

Para la dirección lógica FS = 12E0 y ECX = 0000120, la dirección física para FS:ECX será 12E00 + 0120 = 14000H

Se mueve FS un dígito hexadecimal a la izquierda y se suma el offset ECX.



Modos de direccionamiento.

Modo de direccionamiento	Operando	Segmento
Registro	registro	ninguno
Inmediato	dato	ninguno
Directo	offset	DS
Registro Indirecto	(BX)	DS
	(SI)	DS
	(DI)	DS
	(EAX)	DS
	(EBX)	DS
	(ECX)	DS
	(EDX)	DS
	(ESI)	DS
	(EDI)	DS
Con base	(BX) + desplazamiento	DS
	(BP) + desplazamiento	SS
	(EAX) + desplazamiento	DS
	(EBX) + desplazamiento	DS
	(ECX) + desplazamiento	DS
	(EDX) + desplazamiento	DS
	(EBP) + desplazamiento	SS
Con índice	(DI) + desplazamiento	DS
	(SI) + desplazamiento	DS
	(EDI) + desplazamiento	DS
	(ESI) + desplazamiento	DS
Con base e índice	(R1) (R2)+ desplazamiento donde R1 y R2 son cualquier registro de arriba	Si se utiliza BP el segmento es SS de otra manera el segmento es DS

Acceso de memoria en modo real.

El programador debe tener cuidado en el uso de registros de 32 bits ya que en modo real el rango máximo de memoria es 1M (00000 a FFFFFH) y con registros de 32 bits el rango es mayor a 1M (00000000H a FFFFFFFFH)

Por lo general las razones para aprender a programar en Lenguaje Ensamblador son un tema de discusión, ya que a diferencia de otros, éste no provee las funciones que generalmente se utilizan en las aplicaciones y, tal vez los programas de alto nivel son más fáciles de aprender, por lo que utilizarlos es una buena elección, sin embargo, existen ocasiones en que sólo el Lenguaje Ensamblador puede realizar ciertas tareas con mayor rapidez y eficiencia.

El campo de la computación es tan extenso que existen analistas que emplean lenguajes de alto nivel para desarrollar aplicaciones donde sólo es necesario llamar rutinas que el compilador proporciona y donde el tiempo no es importante, también hay analistas que se especializan en realizar rutinas en lenguajes de bajo nivel por las ventajas que se obtienen.

Aunque el aprendizaje del Lenguaje Ensamblador es al principio un poco difícil y tal vez el tiempo que se tenga que invertir sea mayor a otro lenguaje, los resultados de lo que se puede crear y la disciplina que se adquiere tienen un valor agregado.

Al aprender Lenguaje Ensamblador se adquiere una disciplina para programar de manera clara, se comprende mejor la manera en que se manejan, a bajo nivel, las estructuras de datos permitiendo un mayor control de Hardware y un acceso al procesador de manera óptima y eficiente, por otro lado, al poder hacer programas residentes en memoria, es posible cambiar la manera en que el Sistema Operativo DOS trabaja, lo cual es una ventaja que solo los programadores de Lenguaje Ensamblador pueden utilizar.

El porqué desarrollar en Lenguaje Ensamblador, depende de la aplicación en donde se quiera utilizar y del lenguaje de alto nivel en el que esta aplicación esté realizada, ya que aunque el Lenguaje Ensamblador es más rápido, debido a que los mnemónicos se traducen directamente a Lenguaje Máquina, no es garantía de tener un programa rápido, se pueden tener rutinas lentas, igual que en otros lenguajes, por lo que una vez que se ha escrito un programa, es necesario determinar que parte está consumiendo más tiempo para poder optimizar el código. El Lenguaje Ensamblador no se debe emplear sólo porque se piense que la aplicación se va a ejecutar más rápido, ya que como se sabe, una secuencia escrita en PASCAL, C, FORTRAN puede realizar el trabajo igual de rápido que un algoritmo codificado en Ensamblador.

Hay que recordar que todos los lenguajes de alto nivel están creados en Lenguaje Ensamblador y dado que no existen lenguajes de alto nivel perfectos, se puede utilizar lenguaje de bajo nivel para aumentar sus capacidades. Se debe utilizar lenguajes de alto nivel para aquellas tareas que éstos realizan mejor, pero es importante reconocer que existen rutinas que se ejecutan con mayor ventaja en Lenguaje Ensamblador.

ASCII	American Standard Code for Information Interchange
ASCIIZ	Secuencia de caracteres seguida de un byte con valor 0
Atributo	Características especiales de archivos, datos
BCD	Binary Coded Decimal
Biblioteca	Conjunto de módulos objeto
BIOS	Basic Input/Output System Es una serie de programas almacenados en ROM que proveen servicios de bajo nivel esenciales
BIOS	Basic Input Output System
Bit	Binary digit, Dígito binario (1 ó 0)
Boot	Iniciar el funcionamiento de una computadora
Buffer	Porción reservada de memoria que se utiliza para almacenar datos mientras son procesados
Bus	Un canal común entre dos dispositivos de Hardware
Byte	Unidad de memoria de 8 bits
CGA	Color Graphics Adapter
CISC	Complex Instruction Set Computer
Chip	Circuito integrado
Debug	Programa que ayuda a la depuración de un programa
Default	Por omisión, por defecto. Una acción estándar tomada por el Hardware o software si el usuario no lo ha especificado de otra manera.
Desensamblar	Reconstruir código de Lenguaje Ensamblador a partir de código máquina
Dirección	Número de una ubicación particular de memoria o de almacenamiento periférico.

Device driver	Controlador de dispositivos
Ensamblador	Programa que traduce programas de bajo nivel a código máquina
FAT	File Allocation Table
Handle	Número de 16 bits utilizado para identificar un archivo abierto
Interfaz	Conexión o interacción entre Hardware, software y usuario
Interrupción	Señal que capta la atención del CPU
Kernel	Parte fundamental de un programa. (núcleo)
Offset	Desplazamiento. Distancia desde una dirección de memoria.
Opcode	Parte de una instrucción en Lenguaje Máquina que le dice a la computadora que debe hacer.
Operando	Parte de una instrucción en Lenguaje Máquina que hace referencia a un dato o a un dispositivo periférico.
Párrafo	Bloque de 16 bytes
Pin	Patilla, aguja, terminal de un conector o de un circuito.
Pixel	Es el elemento más pequeño en una pantalla
PSP	Program Segment Prefix
Registro	Circuito pequeño que almacena direcciones de memoria y valores de operaciones internas.
Scroll	Moverse en forma continua hacia adelante, hacia atrás o hacia los costados a través de las imágenes en la pantalla o dentro de una ventana
Segment	Bloque de memoria de 64k que compone parte de una dirección.
Stack	Conjunto de registros de Hardware o cantidad de memoria principal que se usa para cálculos aritméticos o para el seguimiento de operaciones internas.

CONCEPTOS FUNDAMENTALES

ENSAMBLADORES

PROGRAMACIÓN EN LENGUAJE ENSAMBLADOR

TÓPICOS AVANZADOS

- Muhammad Alid Mazidi, Janice Gillispie Mazidi. "The 80x86 IBM PC & Compatible Computers Volume I. Assembly Language Programming on the IBM PC, PS and Compatibles", Prentice Hall, New Jersey, USA 1993.

- David C. Willen, Jeffrey I. Krantz. "8088 Assembler Language Programming: The IBM PC", SAMS, Indiana, USA 1989.

- Steven Holzner. "Advanced Assembly language on the IBM PC", Prentice Hall, New York, USA 1987.

- Allen L Wyatt. "Using Assembly Language", QUE Corporation, USA 1990.

- Nabalyoti Barkakati, Randall Hyde. "The Waite Groups's Microsoft Macro Assembler Bible", SAMS, Indiana, USA 1993.

- Ralf Brown, Jim Kyle. "PC Interrupts. A Programmer's Reference To BIOS, DOS and Third-Party Calls", ADISON WESLEY, USA 1993

- J. Terry Godfrey. "Lenguaje Ensamblador para Microcomputadoras IBM para Principiantes y Avanzados", Prentice Hall, México 1991.

- Kris Jamsa, "DOS, The Complete Reference", Osborne McGrawHill, California, USA 1991.

- John Woram. "The PC Configuration", Bantam Computer books. New york, USA 1990.

- Peter Abel. "IBM PC Assembly Language and Programming", Prentice Hall, New Jersey, USA 1991.