

Ll ensamblador puede parecer un lenguaje de programación en desuso, más si se tiene en cuenta el gran avance que han tenido los compiladores en los últimos años, ofreciendo cada vez mayores facilidades como la programación visual y la orientada a objetos. No obstante hay que tener en cuenta que todo compilador, incluso el más potente que existe, no es más que un conversor del lenguaje de programación que utiliza al lenguaje que entiende el procesador del ordenador, es decir, el ensamblador. De esta circunstancia se deduce que cualquier cosa, que cualquier facilidad o función que ofrezca un lenguaje de programación está porque puede hacerse en ensamblador. Esta circunstancia no se cumple al contrario, y son comunes las situaciones en las que un lenguaje de programación no ofrece solución, debiendo recurrir al lenguaje ensamblador como única alternativa.

CONTENIDO DEL CD-ROM



DIRECTORIO	CONTENIDO
CODIGO:	Rutinas de ayuda a la programación en ensamblador
DESENSAM:	Desensambladores y depuradores.
DOC:	Documentos, información, y tutoriales. Listas Funciones Bios y MS-Dos
EDITOR:	Editores de texto. Incluye versión de 32 bits del popular Vi de Unix
EJEMPLOS:	Programas creados en ensamblador con su código fuente.
ENSAMBLA:	Ensambladores, colección de ensambladores share y freeware, generan ejecutables
HEXEDIT:	Editores de fichero en formato hexadecimal, edición de bits y sectores de disco
UTIL:	Utilidades para programar en ensamblador. Incluye utilidades residentes

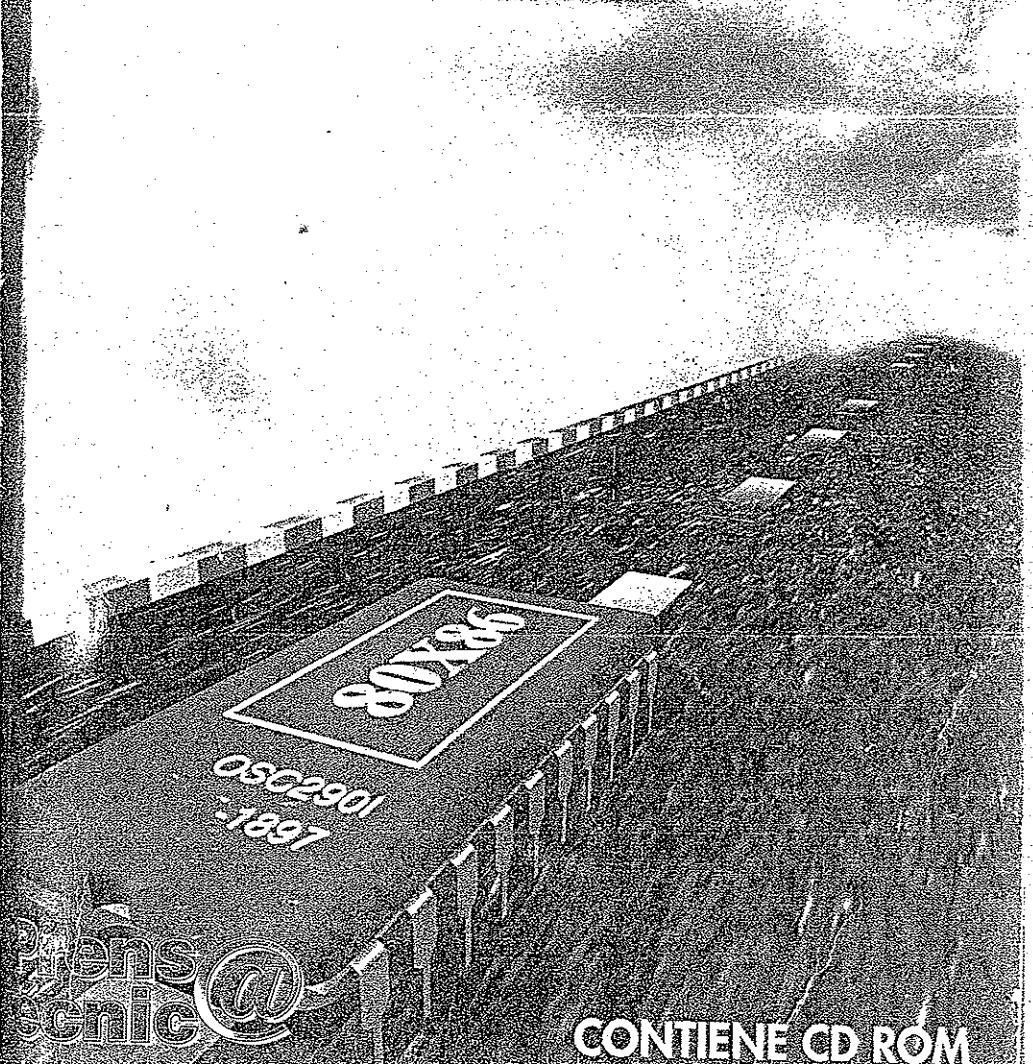


Síllante desde fuera de España
enviar el G.T por G.TI

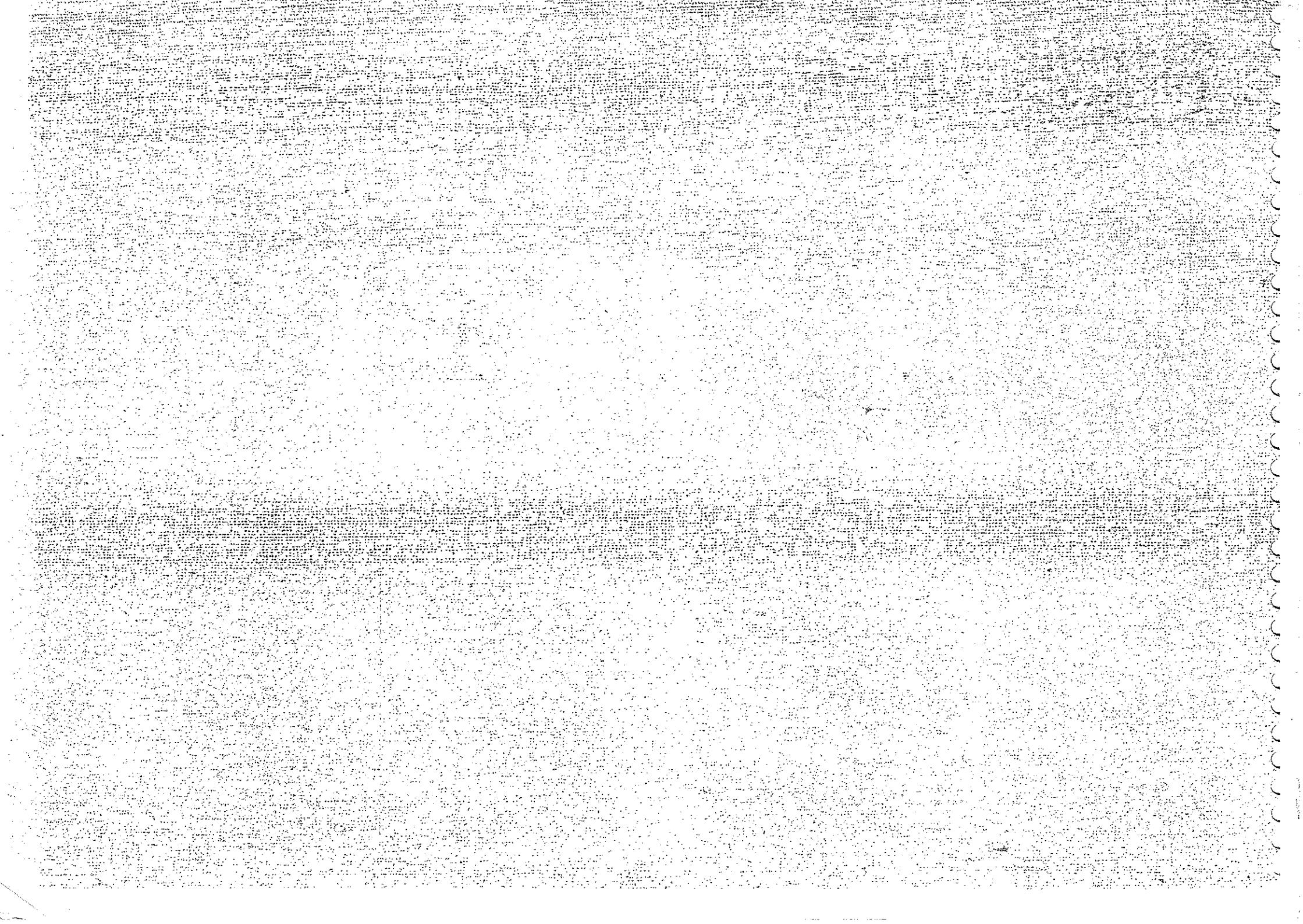
PRENSA TÉCNICA
C/ Vicente Muñoz S. 1º D
28045 Madrid (Spain)
Telf. (91) 519 23 53
Fax. (91) 413 55 77
EIS. (91) 519 75 75
Email: prensatecnica@ibercentroic.es

CÓMO PROGRAMAR EN ENSAMBLADOR

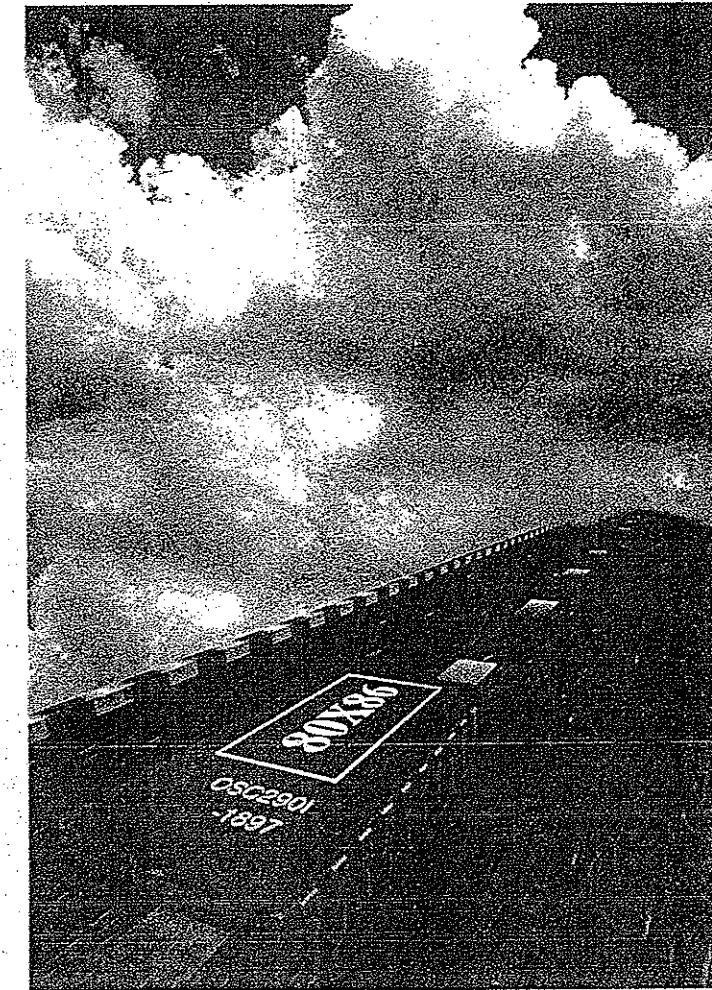
BOX86



CONTIENE CD ROM



Cómo Programar en Ensamblador



EDITOR:

Mario Luis

COORDINADOR DE LA COLECCIÓN:

Eduardo Toribio

REDACCIÓN DE ESTE VOLUMEN:

Tomás Tejón

EDICIÓN:

Miguel Cabezuelo

DISEÑO Y MAQUETACIÓN:

Carlos Sánchez

EDITA:

Prensa Técnica. C/ Alfonso Gómez, 42 - Nave 1-1-2

MADRID 28037

Tfno: (91) 304 06 22 - Fax: (91) 304 17 97

FILMACIÓN:

Grafo Print

IMPRESIÓN:

Eurocolor

DISTRIBUCIÓN:

SGEL

DUPLICACIÓN DE CD:

MPO

ISBN de la obra: 84-89245-11-8

Depósito Legal: M-17351-1998

IMPRESO EN ESPAÑA © 1998

Todos los derechos reservados. Se prohíbe total o parcialmente la reproducción, memorización en sistemas de archivo, transmisión en medio electrónico alguno, fotocopia o cualquier otro sin la previa autorización del editor.

Prensa Técnica no tiene por qué estar de acuerdo con las opiniones escritas por sus colaboradores.

Prefacio

El ensamblador puede parecer un lenguaje de programación en desuso, más si se tiene en cuenta el gran avance que han tenido los compiladores en los últimos años, ofreciendo cada vez mayores facilidades como la programación visual y la orientada a objetos. No obstante, hay que tener en cuenta que todo compilador, incluso el más potente que existe, no es más que un conversor del lenguaje de programación que utiliza al lenguaje que entiende el procesador del ordenador, es decir, el ensamblador. De esta circunstancia se deduce que cualquier cosa que cualquier facilidad o función que ofrezca un lenguaje de programación está porque puede hacerse en ensamblador. Esta circunstancia no se cumple al contrario, y son comunes las situaciones en las que un lenguaje de programación no ofrece solución, debiendo recurrir al lenguaje ensamblador como única alternativa. Es lo que sucede, por ejemplo, con la creación de controladores de dispositivos o cuando se necesita acceder directamente al hardware.

Por ello, la totalidad de los compiladores existentes ofrecen algún mecanismo para integrar en sus programas rutinas realizadas en lenguaje ensamblador, contando incluso algunos de ellos con instrucciones especiales que permiten colocar directamente el código ensamblador dentro del propio lenguaje de programación. Los detalles de esta integración son propios de cada lenguaje, aunque básicamente siguen las mismas reglas que las explicadas en el capítulo de programación modular en ensamblador.

Otras ventajas del ensamblador son su reducido tamaño y su mayor velocidad. Puesto que el código final del programa es creado y optimizado por el programador, éste siempre será más eficiente que el generado por un compilador que utiliza siempre unas estructuras de código

prefijadas en lugar de escoger la más conveniente en cada ocasión..

A pesar de sus ventajas, el lenguaje ensamblador es también el más complejo de todos, pero no por el lenguaje en sí, sino porque requiere poseer una gran cantidad de conocimientos adicionales acerca de los dispositivos conectados al ordenador y del sistema operativo. Además, al tratarse de un lenguaje de muy bajo nivel, está más propenso a los errores de programación que en otros lenguajes. A lo largo de los capítulos se indican los errores más frecuentes y algunas técnicas que permitirán evitarlos en la medida de lo posible.

La información contenida en este libro abarca tres áreas fundamentales: el conjunto de instrucciones del procesador 8086, el lenguaje ensamblador y la creación de programas bajo el sistema operativo MS-DOS. Todas las ideas expuestas son aplicables a procesadores más avanzados de la familia 80x86 y las nuevas versiones del sistema operativo MS-DOS, pues su única diferencia es que integran un mayor número de instrucciones y funciones de apoyo a la programación. Se ha intentado explicar los conceptos y ofrecer la información necesaria para la creación de programas en lenguaje ensamblador, incluyéndose ejemplos descriptivos que ayudarán a comprenderlos mejor, incluso a personas con pocos conocimientos de informática. Por cuestiones de espacio, en ocasiones no se abarcan todos los detalles, omitiéndose aquello que se encuentra en desuso o que se utiliza únicamente en situaciones especiales. De todas formas, una vez que haya adquirido los conceptos básicos, le será muy fácil profundizar en estos detalles a través de los manuales que acompañan al programa ensamblador que utilice y a todo el material adicional que se incluye en el CD-ROM adjunto.

CONCEPTOS BÁSICOS

EL ORDENADOR.....	14
SISTEMAS DE NUMERACION: CONVERSIÓN.....	15
LOS SISTEMAS DE NUMERACION OCTAL Y HEXAGONAL	17
NUMEROS POSITIVOS Y NEGATIVOS.....	18
OPERACIONES ARIMÉTICAS Y LOGICAS.....	20
CÓDIGO MAQUINA Y CÓDIGO DESENSEMBLADO. MNEMONICOS.....	20

EL MICROPROCESADOR 8086/8088

CARACTERÍSTICAS GENERALES.....	25
REGISTROS.....	26
PERIFÉRICOS	26
MEMORIA COMPARTIDA	27
PUERTOS DE ENTRADA Y SALIDA.....	27
INTERRUPCIONES	27
TRANSFERENCIA DE DMA.....	28
TIPOS Y TAMAÑO DE LOS DATOS. ALMACENAMIENTO EN MEMORIA.....	28
DIRECCIONES DE MEMORIA Y SEGMENTACIÓN.....	29
LA PILA.....	30
REGISTROS DEL PROCESADOR.....	32
DE DATOS.....	32
DE ÍNDICE Y PUNTERO	33
DE SEGMENTO	34
ESPECIALES (SP, IP, FLAGS).....	35
FORMATO DE LAS INSTRUCCIONES.....	36
TIPOS DE DIRECCIONAMIENTO.....	37
IMPÍCITO	37
REGISTRO	37
INMEDIATO	37
DIRECTO O ABSOLUTO.....	37
INDIRECTO	38
RELATIVO A BASE	38
RELATIVO A ÍNDICE O INDEXADO	39
INDEXADO CON BASE	39
PREFIJOS DE SEGMENTO	39
CONSIDERACIONES ACERCA DE LOS DIRECCIONAMIENTOS	40
CONJUNTO DE INSTRUCCIONES	41
TRANSFERENCIA DE DATOS.....	42
GENERALES.....	42
MOV.....	42
XCHG.....	43
XLAT.....	43
NOP	44
DE ENTRADA/SALIDA.....	45
IN.....	46
OUT.....	46
DE BANDERAS.....	46



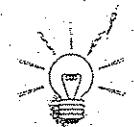
LAHF.....	46
SAHF.....	47
PUSHF.....	47
POPF.....	47
DE DIRECCIONES.....	48
LEA.....	48
LDS.....	48
LES.....	49
DE PILA	49
PUSH	49
POP	50
ÁRITMÉTICAS	50
BINARIAS.....	50
ADD.....	50
ADC.....	51
SUB.....	51
SBB	52
INC	53
DEC	53
MUL.....	53
IMUL	54
DIV	55
IDIV	55
NEG	56
BCD DESEMPAQUETADO	56
AAA.....	57
AAS.....	57
AAM.....	58
AAD	58
BCD EMPAQUETADO	59
DAA	59
DAS	59
DE CONVERSIÓN	60
CBW	60
CWD	60
DE COMPARACIÓN	60
CMP	61
LÓGICAS	61
NOT	61
AND	61
OR	62
XOR	62
TEST	63
DE DESPLAZAMIENTO	63
SAL/SHL	64
SAR	65
SHR	65

ROL	66
ROR	67
RCL	67
RCR	68
DE TRANSFERENCIA DE CONTROL	68
DISTANCIA DE SALTO	69
SALTO INCONDICIONAL	71
JMP	71
SALTO CONDICIONAL	71
LLAMADA A SUBRUTINAS	74
CALL	75
RET	75
DE BUCLES	76
LOOP	76
LOOPE/LOOPZ	77
LOOPNE/LOOPNZ	77
REP	78
REPE/REPZ	78
REPNE/REPNZ	79
DE MANEJO DE CADENAS	79
LODS	80
STOS	81
MOVS	81
SCANS	82
CMPS	83
DE CONTROL	84
ESC	84
HLT	84
WAIT	85
DE INTERRUPCIÓN	85
INT	88
INT3	88
INTO	89
IRET	89
DE BANDERAS	89
DE ACARREO (STC, CLC Y CMC)	90
DE DIRECCIÓN (STD, CLD)	90
DE INTERRUPCIÓN (STI, CLI)	91
PREFIJOS	92
LOCK	92
SEGMENTO	92
EL PROGRAMA DEBUG	92
3. PROGRAMACIÓN EN ENSAMBLADOR	99
PROGRAMAS NECESARIOS PARA PROGRAMACIÓN EN ENSAMBLADOR	101
CONSTRUCCIÓN DE UN PROGRAMA EN ENSAMBLADOR	101
UTILIZACIÓN DE LOS PROGRAMAS	103
FORMATOS DE FICHEROS EJECUTABLES EN MS-DOS	105

INICIO Y FINALIZACIÓN DE UN PROGRAMA ENSAMBLADOR	106
VALORES NUMÉRICOS Y SÍMBOLOS EN ENSAMBLADOR	108
ESTRUCTURA DE UN PROGRAMA EN ENSAMBLADOR	109
ETIQUETAS	111
DEFINICIÓN DE DATOS	115
MODOS DE DIRECCIONAMIENTO EN LENGUAJE ENSAMBLADOR	116
CONSTANTES Y VARIABLES DE ENSAMBLADO	119
EXPRESIONES CONSTANTES	120
OPERADORES DE VARIABLES Y ETIQUETAS	121
PROCEDIMIENTOS Y SUBRUTINAS	122
PASO DE PARÁMETROS A SUBRUTINAS	125
VARIABLES LOCALES	130
SEGMENTOS EN ENSAMBLADOR	131
ESTRUCTURA DE UN PROGRAMA DE TIPO .EXE	136

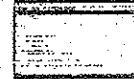
4. FUNCIONES DE LA BIOS Y EL DOS 139

LA ROM BIOS	140
INT 10H: PANTALLA	141
INT 11H Y 12H: CHEQUEO DE LA CONFIGURACIÓN Y MEMORIA	145
INT 13H: MANEJO DE DISCOS	146
INT 14H: PUERTO SERIE	148
INT 16H: TECLADO	149
INT 17H: IMPRESORA	150
OTRAS INTERRUPCIONES DE LA BIOS	151
EL MS-DOS (INT. 21H)	151
ENTRADA / SALIDA ESTÁNDAR (TECLADO/PANTALLA)	152
ACCESO A FICHEROS	154
ABRIR EL FICHERO	155
LECTURA Y ESCRITURA EN EL FICHERO	156
CIERRE DE FICHERO	157
OTRAS OPERACIONES CON FICHEROS	160
MANEJO DE DIRECTORIOS	161
PUERTO SERIE Y DE IMPRESORA	161
SERVICIOS DE FECHA Y HORA	162



5 CARACTERÍSTICAS AVANZADAS DEL ENSAMBLADOR 165

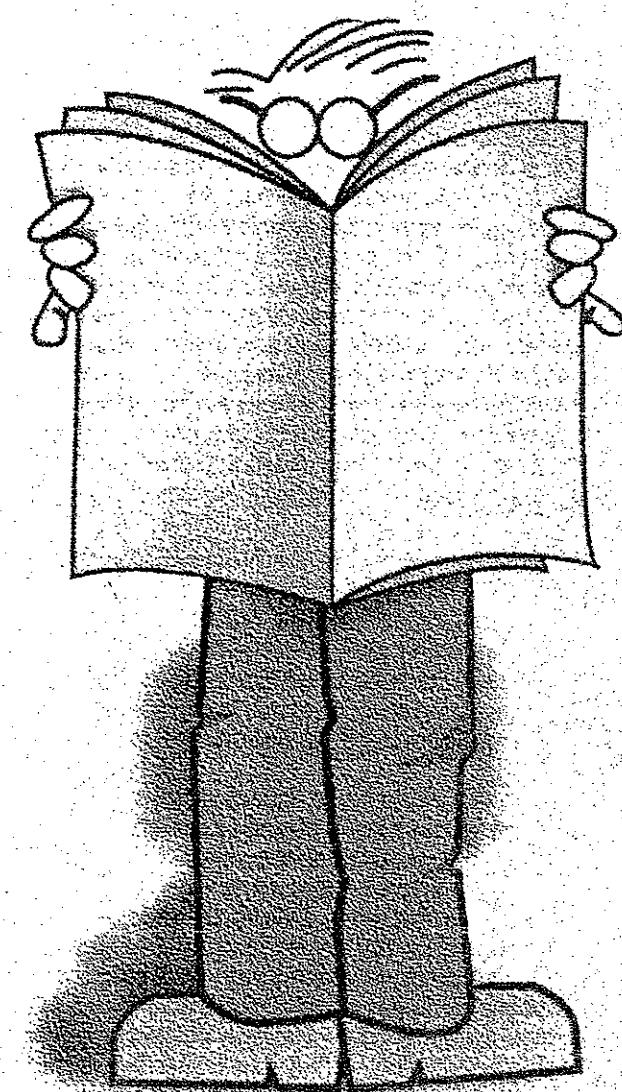
DIVISIÓN EN MÓDULOS	166
MACROS	167
MACROS DE REPETICIÓN	170
VARIABLES DE TIPO REGISTRO (STRUC) Y UNIONES (UNION)	172
CAMPOS DE BITS (RECORD)	173



6 GLOSARIO 177

CONTENIDO DEL CD-ROM 183





Conceptos Básicos

Al hablar de un ordenador la mayoría de las personas se imaginan una caja a la que se conecta un teclado, un monitor, un ratón y, en algunos casos, una impresora. Aun así, esto no es del todo exacto, pues el ordenador en sí no está compuesto únicamente por una placa de circuito impreso sobre la que se encuentran un gran número de componentes electrónicos formando lo que se denomina comúnmente placa base. El resto de dispositivos, como disco duro, unidad de disquete, monitor, etc... se conocen como periféricos y, aunque tienen misiones importantes, no son exclusivos de un tipo determinado de ordenador y pueden utilizarse en diferentes sistemas. Por ejemplo, un disco duro puede conectarse igual a un ordenador tipo PC que a un Macintosh o a un Amiga, a pesar de que son máquinas totalmente diferentes unas de otras. Una puntualización a esta diferencia son las tarjetas de ampliación (gráfica, sonido, red...), ya que no se consideran periféricos, puesto que al depender de forma muy directa de cómo se encuentra diseñada la placa base se consideran extensiones de ésta. Como prueba de ello basta observar que existen modelos de placa base que ya incluyen en ellas una o varias de estas tarjetas.

El ordenador

La placa base se compone fundamentalmente de cuatro elementos: procesador, memoria, bus y circuitería de control. De estos cuatro el más importante es, sin duda, el procesador, pues es en realidad el corazón del ordenador, el que controla el resto de las partes y el que, en definitiva, hace funcionar todo el sistema. La función de este pequeño componente es la de ir leyendo de la memoria las instrucciones que componen el programa y llevarlas a cabo enviando para ello los mensajes necesarios al resto de componentes.

El segundo gran componente, la memoria, es un conjunto muy grande de casillas; cada una de las cuales puede almacenar un valor. Todas estas celdillas están numeradas de forma secuencial, de manera que para referirse a cualquiera de ellas basta con dar su número de posición, al que se denomina "Dirección de memoria". Dentro del ordenador existen dos tipos de memoria: RAM (Random Access Memory) y ROM (Read Only Memory), de las que la más abundante es la del primer tipo. La memoria de tipo RAM se caracteriza porque los valores que contiene en sus posiciones pueden tanto leverse como escribirse, pero cuenta con el inconveniente de que cuando le falta la alimentación eléctrica su contenido se pierde. Es por ello que en la placa base se encuentra otro tipo de memoria, la ROM que, aunque no se puede escribir, su contenido no se pierde al desconectar el ordenador.

Cuando se enciende el ordenador, lo primero que hace el procesador es comenzar la ejecución de un programa especial contenido en la memoria

ROM del ordenador, el cual realiza en primer lugar un chequeo del sistema y a continuación carga en RAM el sistema operativo, pasándole el control del sistema. Es este sistema operativo el que nos muestra la conocida línea de comandos desde donde se pueden llevar a cabo tanto tareas administrativas (borrar, copiar, etc...) como iniciar programas.

El tercer componente, llamado bus, no es en realidad un componente electrónico, sino un conjunto de cables, normalmente integrado como pistas dentro de la placa de circuito impreso, cuya misión es la de transmitir datos entre los diferentes componentes de la placa. Existen tres tipos de buses: de datos, de direcciones y de control, que funcionan de forma conjunta a la hora de transmitir o recibir la información. Así pues, mientras que el bus de datos lleva el dato, el de direcciones indica dónde hay que dejarlo o cogerlo, y el de control especifica algunos detalles referentes a la transacción, como puede ser si se trata de una lectura, una escritura, etc... Por ejemplo, si el procesador desease leer una posición de memoria, especificaría en el bus de direcciones la dirección que desea leer, y en el de control le indicaría a la memoria que se trata de una operación de lectura. La memoria, por su parte, colocaría en el bus de datos el contenido de la dirección seleccionada, y a través del bus de control señalaría al procesador que ya puede recoger la respuesta.

Por último, se encuentra un amplio conjunto de componentes que conforma la circuitería de control y ayuda a comunicarse con los periféricos o, también, habilita la comunicación entre el procesador y la memoria estableciendo mecanismos como gestión de interrupciones, transferencias de DMA y gestión en la comunicación con las tarjetas accesorias instaladas.

Sistemas de numeración: Conversión

En nuestro sistema de numeración, el decimal, contamos con diez dígitos que van desde el 0 al 9. Para componer los diferentes números lo que se hace es combinar entre sí estas diez cifras de forma que, según la posición en que se encuentre, tendrá un valor diferente, tanto mayor cuanto más a la izquierda esté. Por ejemplo, si se tiene la cifra 21 y la 201, en la primera el 2 tiene una aportación de 20 unidades al valor, que es menor al valor de la hecha por el 2 de la segunda, que es de 200. Por tanto, una cifra colocada una posición más a la izquierda que otra tiene un valor 10 veces superior, 100 si está dos posiciones y así sucesivamente. Esta relación puede observarse en el siguiente ejemplo :

$$24891 = 2 \times 10000 + 4 \times 1000 + 8 \times 100 + 9 \times 10 + 1$$

y si se expresan los pesos como potencias de 10 queda:

$$24891 = 2 \times 10^4 + 4 \times 10^3 + 8 \times 10^2 + 9 \times 10^1 + 1 \times 10^0$$

De esta forma se ve cómo cualquier número puede expresarse como la suma de los productos de las cifras por potencias sucesivas de la base, que en el ejemplo es la base decimal o 10.

Aunque el sistema decimal es al que estamos habituados los humanos, para las máquinas es muy complejo, debido a problemas de diseño. Para los ordenadores, que trabajan con información digital, lo ideal es manejar únicamente dos cifras, el 1 y el 0, que corresponden a un valor determinado de tensión en una línea y a la no existencia de tensión alguna respectivamente. Dado que sólo existen dos cifras, se trabaja en un sistema de numeración en base 2 conocido como binario, sistema que funciona de forma análoga al decimal explicado anteriormente. Según esto, el número binario 10011101 equivaldría a:

$$1 \times 2^7 + 0 \times 2^6 + 0 \times 2^5 + 1 \times 2^4 + 1 \times 2^3 + 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0$$

Si ahora se quisiese transformar dicho número al sistema decimal bastaría con realizar la suma de potencias en sistema decimal, quedando:

$$\begin{aligned} &1 \times 128 + 0 \times 64 + 0 \times 32 + 1 \times 16 + 1 \times 8 + 1 \times 4 + 0 \times 2 + 1 \times 1 = \\ &128 + 16 + 8 + 4 + 1 = 157 \end{aligned}$$

Para realizar la operación opuesta, es decir, el paso de decimal a binario, se va dividiendo sucesivamente el número por la base (2 en este caso) anotando el resto y repitiendo el proceso con el nuevo cociente obtenido hasta que se obtenga un cociente de 0. Por ejemplo, para devolver el número del anterior ejemplo a su representación binaria, se dividiría en primer lugar por 2:

Operación	Cociente	resto
157 / 2	78	1

Ahora se vuelve a dividir el cociente obtenido de nuevo por la base, repitiendo este proceso hasta que se obtenga un cociente de 0:

Operación	Cociente	resto
78 / 2	39	0
39 / 2	19	1
19 / 2	9	1
9 / 2	4	1
4 / 2	2	0
2 / 2	1	0
1 / 2	0	1

La representación binaria se obtiene cogiendo los restos de las divisiones en orden inverso al que se ha ido obteniendo:

$$157 \text{ (decimal)} = 10011101 \text{ (binario)}$$

A la hora de trabajar con números en diferentes sistemas de numeración surge un problema a la hora de diferenciar unos de otros. Como se puede observar, 10011101 es un número válido en sistema binario, pero también lo es en decimal. Por lo tanto, cuando se trabaja con cifras en diferentes bases por convenio se añade una letra tras la cifra para indicar cuál es su base. El sistema decimal tiene asignada la letra d, mientras que para el binario es la b. El ejemplo anterior, según este convenio quedaría:

$$157d = 10011101b$$

Los sistemas de numeración octal y hexadecimal

El sistema binario, aunque adecuado para el ordenador, es poco manejable para las personas, principalmente por la gran longitud que alcanzan rápidamente los números escritos en este sistema, lo cual unido al hecho de que sólo existen dos dígitos, hace que sea fácil cometer errores al trabajar con ellas.

Así pues, a la hora de manejar cifras en sistema binario se utilizan dos sistemas de numeración alternativos cuya base es una potencia de 2. Estos son el sistema de base 8, conocido como octal, y el de base 16, llamado hexadecimal. Al igual que sucedía con los valores decimales y binarios, cuando se mezclen números en octal y hexadecimal con otras bases se distinguirán posponiendo la letra o (o la q) a las cifras octales y la letra h a las hexadecimales.

Para representar números en sistema hexadecimal es preciso contar con 16 dígitos diferentes. Puesto que en nuestro sistema de numeración sólo disponemos de los diez primeros (0 al 9), lo que se hace es continuar la secuencia de dígitos con letras del alfabeto, comenzando con la A, a la que se le da un valor de 10, la B 11 y así sucesivamente hasta la letra F, que corresponde al valor 15. Las reglas de conversión entre este sistema y el decimal son análogas a las explicadas para el binario, con la única diferencia de sustituir el valor 2 de la base binaria por el 16 del hexadecimal a la hora de hacer las operaciones. Veamos esto con un ejemplo :

$$\begin{aligned} 1AE7h &= 1 \times 16^3 + 10 \times 16^2 + 14 \times 16^1 + 7 \times 16^0 = \\ &1 \times 4096 + 10 \times 256 + 14 \times 16 + 7 \times 1 = 6887d \end{aligned}$$

y la conversión en sentido inverso sería:

Cociente	Resto	Cifra	hex.
6887 / 16	430	7	7
430 / 16	26	14	E
26 / 16	1	10	A
1 / 16	0	1	1

Los números hexadecimales tienen una propiedad interesante y es que, al ser 16 una potencia entera de 2, la conversión entre sistema binario y hexadecimal puede hacerse de forma directa. Para pasar de binario a hexadecimal se separan los dígitos binarios en grupos de cuatro comenzando desde la derecha. Cada grupo de cuatro cifras puede tener un valor entre 0 (0000) y 15 (1111), es decir, equivale a una cifra hexadecimal. Para realizar la conversión bastará, por tanto, con sustituir cada grupo de dígitos binarios por su correspondiente cifra hexadecimal.

Por ejemplo, si se tiene el número:

1101011100111b

Se separan las cifras de cuatro en cuatro, asignando la cifra hexadecimal que corresponda a su valor:

$$0001:1010:1110:0111b = 1.10.14.7 = 1AE7h$$

La conversión inversa, de hexadecimal a binario, se realiza cambiando cada dígito hexadecimal por los cuatro bits binarios correspondientes a su valor. Puede observarlo en el ejemplo anterior simplemente siguiendo los pasos en orden inverso. Este sistema también puede aplicarse para convertir entre octal y binario, solo que en este caso los dígitos se deberán agrupar de tres en tres.

A la hora de trabajar en diferentes bases debe tenerse en cuenta que al cambiar un valor numérico de base, lo único que cambia es la forma de escribirlo, pero no su valor real, que sigue siendo el mismo. Una analogía que ayudará a aclarar este asunto es expresar el peso de un objeto en kilogramos o en libras. La cifra en una representación es diferente a la de la otra, pero el peso del objeto es el mismo.

Números positivos y negativos

En el sistema decimal, el signo de un número se expresa anteponiendo a la cantidad un guion para indicar que se trata de un

número negativo o nada en caso de que sea positivo. Pese a ello, los ordenadores sólo son capaces de almacenar dígitos binarios, no signos, por lo que hay que recurrir a algún sistema alternativo para indicar el signo de un número.

El sistema más comúnmente utilizado se denomina "complemento a dos" y se basa en las siguientes reglas:

- Un número positivo tiene su bit más significativo (el de la izquierda) siempre a cero, y uno negativo a uno.
- Para convertir un número positivo en negativo (o viceversa) se invierten todos los bits del número, y luego se le suma uno al resultado.

Por ejemplo, el número 01110b es positivo, mientras que el 11110b es negativo. Para conocer el valor del positivo se hace de la forma habitual, mientras que para el negativo habría que convertirlo primero en positivo para saber qué cifra representa:

$$\begin{aligned} 01110b &= 8 + 4 + 2 = 14 \\ 11110b &= -(00001b + 1) = -(1 + 1) = -2 \end{aligned}$$

Las razones de utilizar este tipo de representación son fundamentalmente las propiedades matemáticas que poseen este tipo de números, la más importante de las cuales es que la suma o resta de dos números en esta representación, realizada por el sistema habitual de cantidades sin signo, da siempre el resultado correcto, sin necesidad de tener en cuenta si se trata de números con signo o sin él y de cuál es dicho signo.

$$\begin{aligned} 01110b &= 14 \\ 11110b &= -2 \\ 101100b &= 12 \end{aligned}$$

El bit que se desborda (el que sobra por la izquierda) no se tiene en cuenta.

Antes de realizar operaciones con números con signo es preciso cerciorarse de que todos tengan el mismo número de bits. En caso de que esto no sea así, entonces habrá que alargar el tamaño de los más cortos al tamaño del mayor. Para ello se repite la cifra a la izquierda tantas veces como sea necesario, es decir, a los negativos se les añaden unos a la izquierda, mientras que a los positivos se les añaden ceros. A esta operación se le denomina "extensión de signo".

Aunque la forma de trabajar con estas cantidades parezca complicada, no debe preocuparse, pues el programa ensamblador se encarga de

manejar todo a través de valores decimales con signo menos (-) de la forma habitual, encargándose luego él de transformarlo al formato correcto. Además, el procesador dispone también de instrucciones especiales para realizar la conversión a complemento a dos (cambio de signo) y de extensión de signo.

Operaciones aritméticas y lógicas

Al igual que ocurre en el sistema decimal, es posible realizar operaciones matemáticas, tales como suma, resta, multiplicación y división con números en otras bases mediante un procedimiento análogo al utilizado en base decimal. Sin embargo, si alguna vez necesita realizar alguna operación a mano entre números en otra base distinta de la decimal, el procedimiento recomendado es que en primer lugar transforme todos los números a sistema decimal, realice las operaciones pertinentes y, por último, transforme el resultado a la base deseada.

Además de las conocidas operaciones matemáticas, el sistema binario define otra serie de funciones especiales conocidas como operaciones lógicas, que surgen de la idea de asignar el valor "falso" al dígito 0 y el valor "verdadero" al 1. Estas operaciones se definen en principio para cantidades de un único dígito. La primera de estas operaciones es la negación (NOT), que toma un único operando y transforma su valor en el opuesto, tal y como se ve en la tabla siguiente:

X	NOT X
0	1
1	0

La operación "y" (AND) trabaja sobre dos valores devolviendo el equivalente a la conjunción "y" utilizada en nuestra lengua. Devuelve, por tanto, el valor "cierto" (1) sólo si ambos operandos son ciertos. La operación "o" (OR) devuelve un valor de cierto si alguno o ambos de los operandos lo son. Por último, la operación "or exclusivo" (XOR) es similar a la anterior, pero con la diferencia de que si ambos operandos son ciertos, el resultado es falso.

La siguiente tabla resume las operaciones indicadas:

X	y	x AND y	x OR y	x XOR y
0	0	0	0	0
0	1	0	1	1
1	0	0	1	1
1	1	1	1	0

Esto es correcto para operaciones realizadas sobre cifras de un único dígito. Pero ¿qué sucede cuando las cifras son mayores?. Para la operación NOT se cambia cada uno de los dígitos por su opuesto, mientras que para el resto lo que se hace es aplicar la operación a cada dígito de la primera cantidad con el que ocupa la misma posición en la segunda. Por ejemplo:

$$\begin{array}{l} \text{01110100 AND } 11011101 = 01110100 \\ \text{01110100 OR } 11011101 = 11111101 \\ \text{01110100 XOR } 11011101 = 10101001 \end{array}$$

La principal utilidad de este tipo de operaciones es la de modificar el valor de algún dígito de una cifra binaria sin afectar al resto, realizando para ello una operación lógica del número con un valor llamado máscara que define qué dígitos serán los que cambiarán, dejando el resto sin modificar.

La operación AND es útil para poner al valor 0 en unas posiciones determinadas. Para esto hay que utilizar una máscara que tenga el valor 1 para los dígitos que se desea dejar sin modificar y el valor 0 para los que se quiera poner a 0. Por ejemplo:

$$01110100 \text{ AND } 11100111 = 01100100$$

El OR realiza la operación contraria, colocando a 1 las cifras que en la máscara tienen el valor 1 y dejando el resto inalteradas.

$$01110100 \text{ OR } 00011000 = 01111100$$

Por último la operación XOR realiza una inversión de los dígitos que en la máscara aparecen como 1, dejando inalterados el resto.

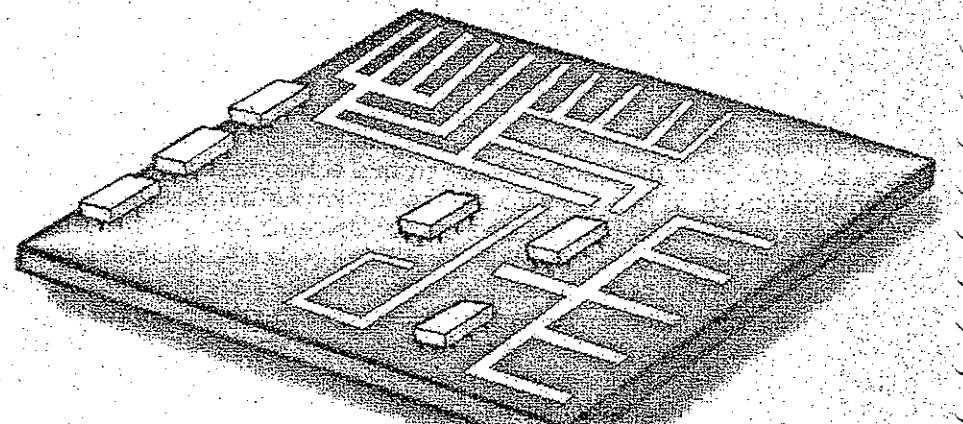
$$01110100 \text{ XOR } 10011100 = 11101000$$

Código máquina y código desensamblado. Mnémicos

Durante su funcionamiento, el procesador va extrayendo de la memoria las instrucciones que componen el programa, las decodifica y envía las señales necesarias a través de los buses para llevarla a cabo, pasando a continuación a ejecutar la siguiente instrucción del programa. Estas

instrucciones que maneja el procesador son valores numéricos en binario, en los cuales se encuentra contenido tanto el tipo de operación a realizar como de dónde extraer los datos necesarios para realizarla y el lugar donde depositar el resultado. Es a este conjunto de instrucciones a lo que se denomina "código máquina" y, como se puede suponer, es muy críptico y difícil de manejar.

Es por ello que a la hora de realizar un programa en código máquina lo que se hace es no trabajar con los valores binarios directamente, sino utilizar una representación en forma de texto, asignando a cada instrucción binaria una palabra identificativa de la acción que realiza y colocando a continuación de ella los datos que necesita, el lugar de donde obtenerlos; así como dónde depositar el resultado. Es a esta representación a lo que se denomina código desensamblado, una forma de representación más comprensible que la serie de cifras binarias que componen el código máquina. Por poner un ejemplo, la secuencia binaria 01000000 equivale, desensamblada, a la instrucción INC AX, la cual suma 1 al contenido del registro AX. Se ve claramente cómo la segunda forma de expresarlo es comprensible directamente, mientras que la primera no lo es.



El Microprocesador 8086/8088

Cuando en el año 1981 el procesador de 16 bits 8086 fabricado por Intel fue escogido por IBM para su nuevo modelo de ordenador personal, sin duda, ninguna de las dos compañías sospechaba el gran impacto que iba a tener este producto, llegando a dominar el mercado de la informática. La clave del éxito no fue sólo el ofrecer un buen producto para la época, sino el hacerlo totalmente extensible y ampliable, lo que le ha permitido ir evolucionando a lo largo de los años con nuevos modelos que iban incorporando todas aquellas innovaciones que se realizaban en el campo de la informática personal.

Aunque hoy en día ya no existen ordenadores basados en 8086 y 8088, sino sus descendientes directos, la forma de programarlos no ha variado sustancialmente, sino que se ha ampliado y mejorado. Por tanto, se comenzará explicando el funcionamiento de estos procesadores para, más adelante, esbozar las diferencias con las versiones actuales de sus sucesores.

Características generales

El 8086 es un procesador de 16 bits, lo cual quiere decir que internamente es capaz de trabajar con cifras binarias de 16 dígitos de longitud. Las direcciones de memoria con las que trabaja tienen una longitud de 20 bits, lo que le permite direccionar un total de 1 Megabyte de memoria. De este procesador existe una segunda versión llamada 8088, cuya única diferencia consiste en que el bus de datos es de 8 bits en lugar de 16, lo que le obliga a leer y escribir los datos en dos etapas, con la consiguiente pérdida de tiempo. Únicamente se utilizó en los primeros modelos de PC, pues a comienzos de la década de los 80 la tecnología de 16 bits estaba poco desarrollada y los componentes de este tipo eran demasiado caros, siendo sustituida en corto tiempo por el 8086.

REGISTROS

Dentro del procesador existen ocho contenedores especiales de 16 bits llamados registros, cuya finalidad es la de guardar datos de forma temporal durante el transcurso del programa. La ventaja de los registros es que, si una instrucción del programa utiliza un dato contenido en un registro, ésta se ejecutará más rápido, pues el tiempo necesario para acceder a un registro es mucho menor que el necesario para leer ese mismo dato si estuviese contenido en la memoria RAM del ordenador. Además, el 8086 tiene ciertas peculiaridades que se verán más adelante y que obligan a utilizar los registros en ciertos tipos de instrucciones.

PERIFÉRICOS Y TARJETAS

Si el ordenador fuese simplemente un procesador que lee programas de una memoria, los ejecuta y almacena unos resultados en la misma,

sería de poca utilidad. Se necesita, por tanto, toda una serie de elementos auxiliares que permitan, por ejemplo, obtener datos por parte del usuario o mostrarle los resultados obtenidos, además de ofrecer la posibilidad de guardarlos en algún dispositivo en el que no se pierdan, como es el caso de un disquete, un disco duro o una impresora.

Por esta razón, al procesador se le añade toda una gama de complementos y dispositivos diseñados para facilitar la comunicación con el usuario (pantalla, teclado, ratón, impresora, puerto serie...) y el almacenamiento permanente de los datos, el programa y sus resultados. Estos dispositivos se conectan físicamente al ordenador en forma de tarjetas de ampliación, o bien enchufándolos en algún conector especial de la placa base. Para comunicarse con ellos, el procesador utiliza uno o varios de estos cuatro métodos: a través de memoria compartida, por puertos de entrada/salida, interrupciones y transferencias de DMA.

MEMORIA COMPARTIDA

El sistema de memoria compartida es el utilizado, por ejemplo, en las tarjetas de vídeo. Mediante este método la tarjeta coloca al alcance del procesador una zona de memoria propia de ella, de tal forma que aquél la ve como si fuese memoria normal. Se trata de un sistema muy simple pero que no se utiliza mucho, dado que el 8086 tiene muy limitada la cantidad de memoria a la que puede acceder y, si hubiese muchos dispositivos que utilizasen este sistema, quedaría menos espacio para los programas.

PUERTOS DE ENTRADA/SALIDA

Dada la limitación de memoria del 8086, se le añadió un sistema de acceso a periféricos denominado "puertos de entrada/salida". Su funcionamiento es similar al de la memoria compartida, pero en este caso las tarjetas colocan su memoria en un espacio de direcciones especial separado de la RAM principal del ordenador. El acceso a estas posiciones se lleva a cabo mediante instrucciones especiales de entrada/salida, instrucciones que se encuentran bastante limitadas en sus posibilidades; lo que provoca que las tarjetas que utilizan este sistema ocupen muy pocas direcciones de E/S. Este es el sistema más utilizado, y no existe ninguna tarjeta del PC que no lo utilice.

INTERRUPCIONES

Algunas veces, un dispositivo necesita llamar la atención del procesador para indicarle, por ejemplo, que tiene un dato disponible o que ha terminado de realizar la acción que se le había encargado. Este es el sistema utilizado, por ejemplo, por el teclado, el cual envía una interrupción al procesador cada vez que se pulsa una tecla, o también el utilizado por la unidad de disco para señalar que ha terminado de escribir los datos en el disquete. A este tipo de interrupciones también se les denomina *disturbios*.

Registros del procesador

Los registros del procesador son unas posiciones especiales de memoria de acceso ultrarrápido que se encuentran contenidas dentro del procesador y que sirven para contener datos de utilización frecuente por parte del programa, o que el procesador necesita para su funcionamiento. Aunque en principio cualquier registro puede utilizarse para este propósito, cada uno de ellos tiene una finalidad específica que permite clasificarlos en cuatro categorías: de datos, de índice y de puntero, de segmento y especiales.

DE DATOS

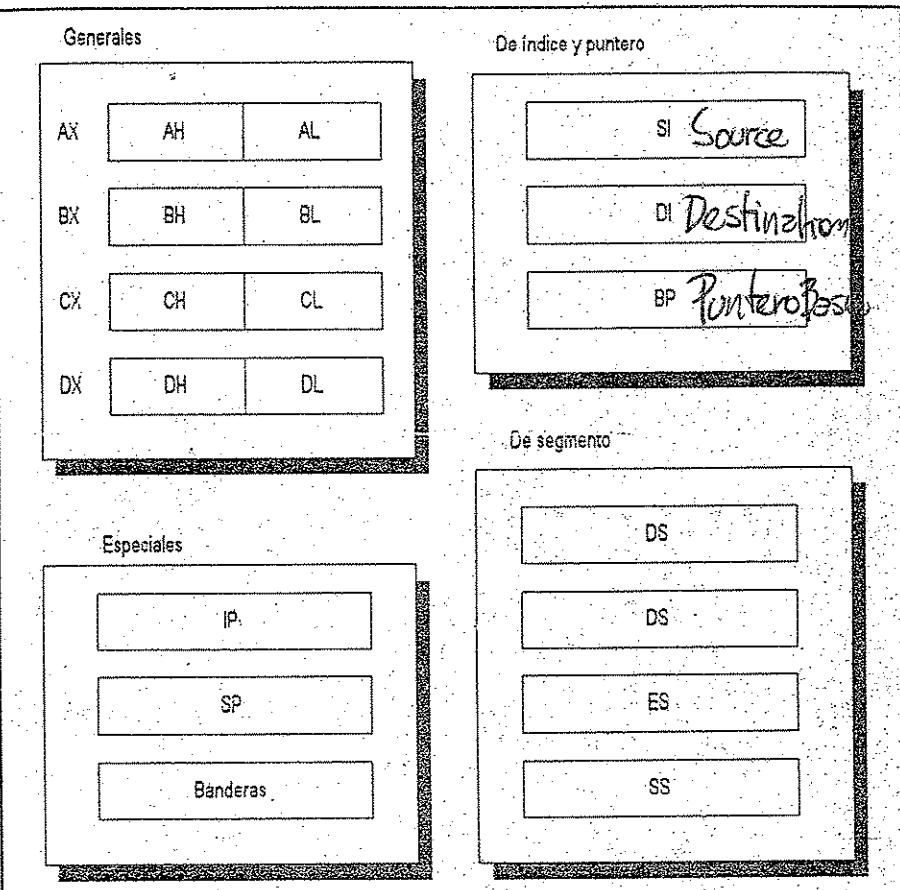
Los registros más utilizados durante el programa son los denominados "de datos". Su nombre es debido a que su utilidad fundamental es la de almacenar datos que se usan de forma frecuente durante un fragmento de programa, evitándose así su reiterada lectura y escritura en la memoria principal del programa. En total hay cuatro registros de este tipo, de un tamaño de 16 bits (1 word) que se denominan AX, BX, CX y DX (las cuatro primeras letras del alfabeto terminadas con una X). Cada uno de estos registros se subdivide además en dos registros de tipo byte, que se denominan con la misma letra que el registro completo pero terminada en H si se refiere al byte superior o HIGH (8 bits más significativos del registro) o en L si se refiere al byte inferior o LOW (los 8 bits menos significativos). Así pues, si hay un dato de tipo word almacenado en el registro AX, es posible acceder a su byte superior mediante el registro AH y al inferior a través del AL. Los otros tres registros también cuentan con esta propiedad, por lo que existe un BL y BH, CL y CH y DL y DH. [ver gráfico.1]

Aunque almacenar datos temporales es la principal finalidad de los registros de datos, cada uno de ellos tiene usos específicos que son propios de ellos y que ningún otro registro puede realizar.

El registro AX (Acumulador) es utilizado por algunas operaciones aritméticas como origen o destino de los datos de forma implícita, es decir, que siempre utilizan este registro y no hay posibilidad de cambiarlo por otro. Por su parte, el registro BX (Base) se utiliza en algunos modos de direccionamiento para formar la dirección de memoria de la que obtiene o en la que almacena los datos una determinada instrucción.

El registro CX (Contador) se utiliza con las instrucciones de repetición y de bucle, almacenando el número de veces que se repetirá una determinada instrucción o fragmento de programa. Por último, el registro DX (Datos) se combina con el AX en algunas instrucciones que manejan cantidades de 32 bits y especifica la dirección de un puerto de entrada/salida.

GRÁFICO.1



Estas utilizaciones especiales pueden recordarse fácilmente observando que la primera letra del nombre del registro puede asociarse a su función, que se ha indicado entre paréntesis.

DE ÍNDICE Y DE PUNTERO

El segundo grupo de registros corresponde a los que se denominan "de índice y de puntero". Aunque al igual que los de datos pueden utilizarse para almacenar datos de forma temporal, su principal finalidad es la de almacenar la posición en memoria donde se encuentra algún dato necesario para las instrucciones del programa. En total hay tres registros de este tipo que reciben los nombres de SI, DI y BP.

Como ocurría con los registros de datos, éstos también tienen ciertas utilizaciones especiales, de donde vienen derivados sus nombres. Si se utilizan como índices en los fragmentos de programa que necesitan acceder a varias posiciones consecutivas de memoria, calidad por la que sus nombres acaban por la letra "I" de índice. Por otra parte, si es utilizado por algunas instrucciones como el origen de los datos que necesita, mientras que DI se utiliza como destino de los resultados que producen. Es esta última finalidad la que da origen a la primera letra de su nombre, pues en inglés ~~source significa origen y Destination destino~~.

El último registro, BP (o puntero base), es utilizado por instrucciones que quieren acceder a datos contenidos en el interior de la pila, y es utilizado en los usos alternativos que se indicaron para la pila, que se verán con más detalle en capítulos posteriores.

DE SEGMENTO

Cuando se habló de la organización de memoria ya se indicó que una posición de memoria quedaba determinada por una dirección de segmento y un desplazamiento relativo a dicho segmento denominado offset, donde el segmento se encontraba contenido en unos registros especiales denominados "de segmento".

Cuando una instrucción hace referencia a una posición de memoria, ésta sólo especifica el offset de dicha posición, mientras que el segmento en que está contenida se obtiene de uno de los registros de segmento. Existen cuatro registros de este tipo, que reciben el nombre de CS, DS, ES y SS.

El registro CS (Code Segment) apunta siempre al segmento donde se encuentra la instrucción que se está ejecutando. Este registro no se puede modificar directamente y debe hacerse mediante las denominadas "instrucciones de transferencia de control".

El registro DS (Data Segment) apunta al segmento de memoria donde las instrucciones leen los datos o los almacenan. Este registro puede cambiarse durante la ejecución del programa para así poder acceder a toda la memoria instalada en el sistema. El registro ES (Extra Segment), por su parte, indica una zona de memoria para datos adicionales, mientras que el SS apunta al segmento de memoria donde se encuentra la pila del procesador.

Cuando una instrucción necesita leer o escribir un dato en la memoria, lo hace por defecto en el segmento apuntado por DS. Este funcionamiento puede variarse de forma que la instrucción obtenga el dato de cualquiera de los otros tres segmentos indicándolo de forma explícita en la instrucción,

mediante los denominados "prefijos de segmento", que se verán cuando se expliquen los modos de direccionamiento.

ESPECIALES (SP, IP, FLAGS)

El último grupo de registros lo constituyen los denominados "especiales", que son utilizados por el procesador para almacenar sus propios datos y que raramente deben ser modificados por el programa.

SP (Stack Pointer) es el nombre del registro que contiene el puntero de la pila, almacenando la dirección de memoria donde se guardará el siguiente dato en la pila.

Cuando el procesador necesita una nueva instrucción de programa, la obtiene de la dirección indicada por el registro IP (Instruction Pointer), que contiene el desplazamiento dentro del segmento apuntado por el registro CS desde donde se leerá dicha instrucción. Al igual que sucedía con el registro CS, éste sólo puede modificarse mediante las instrucciones de transferencia de control.

Por último, el registro de banderas (o flags) se utiliza para indicar al programa ciertas condiciones que han tenido lugar como resultado de la última operación indicada. Cada uno de los bits del registro de banderas marca si se ha producido una de estas condiciones: con un valor 1 si se produjo o 0 si no tuvo lugar. Es por esta razón que a cada uno de estos bits se le denomina Bandera o, en inglés, flag.

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
				OF	DF	IF	TF	SF	ZF		AF		PF		CF

Las banderas contenidas en este registro son las siguientes:

CF (Carry Flag): Bandera de acarreo. Bit 0. Toma el valor 1 si en la última operación de suma o de resta el resultado no cabía en el destino, es decir, si tras realizar la operación se acaba "llevándose una unidad". En caso de que el resultado de la operación quepa en el destino, este indicador toma el valor 0.

PF (Parity Flag): Bandera de paridad. Bit 2. Toma el valor 1 cuando el resultado de la última operación tiene un número par de bits al valor 1 y 0 en caso contrario.

AF (Auxiliary Flag): Bandera auxiliar. Bit 4. Es un tipo de acarreo especial que se utiliza para indicar un acarreo entre las cifras de un número en formato BCD.

ZF (Zero Flag): Bandera de Cero. Bit 6. Adopta el valor 1 en caso de que la última operación haya tenido un resultado igual a cero.

Para resultados distintos de cero toma el valor de 0.

SF (Sign Flag): Bandera de signo. Bit 7. Indica si el resultado de la última operación ha sido negativo, situación en la cual toma el valor 1.

TF (Trace Flag): Bandera de paso a paso. Bit 8. Si esta bandera tiene el valor 1, entonces el procesador genera una interrupción interna por cada instrucción que ejecuta, posibilitando así a un programa de depuración el seguir el transcurso de un programa paso a paso.

IF (Interrupt Flag): Bandera de interrupciones. Bit 9. Cuando se coloca el valor 0 en esta bandera, el procesador ignora las peticiones de interrupción que le hagan los dispositivos externos. Esto es útil para proteger secciones críticas del código que podrían dejar el sistema en un estado inestable en caso de que se produjese una interrupción durante su ejecución.

DF (Direction Flag): Bandera de dirección. Bit 10. Si tiene el valor 0, entonces las instrucciones de proceso de cadenas realizan su trabajo de forma ascendente, es decir, avanzando hacia arriba en la memoria. Si toma el valor 1 se realizan de forma inversa, es decir, desde el final hacia el principio.

OF (Overflow Flag): Bandera de desbordamiento. Bit 11. Cualquier operación que provoque un resultado que no quiera en el destino colocará el valor 1 en este indicador.

Formato de las instrucciones

Las instrucciones del 8086 se componen de un nombre de instrucción denominado mnemónico, que va seguido de ninguno, uno o dos operandos. En caso de que la instrucción requiera que se le indique un lugar donde depositar su resultado, deberá indicársele siempre este dato como primer operando. Por ejemplo, la instrucción PUSH AX no necesita que se le especifique un destino para los datos, porque utiliza como destino la pila, por lo que consta de dos partes: el mnemónico PUSH, que corresponde a la instrucción de "guardar un dato en la pila", y un único operando que le dice que debe obtener el dato a guardar del registro AX. Por otra parte, la instrucción MOV AX,BX sí que necesita un destino de datos, por lo que dicho destino será el primer operando, es decir, el registro AX. Según esto, la instrucción consta del mnemónico MOV, que corresponde a la instrucción "copiar un dato", y dos operandos, por lo que el origen de la copia será el segundo operando y el destino el primero. El resultado de esta instrucción será, por tanto, que el valor contenido en el registro BX se copia en el AX.

Tipos de direccionamiento

Las instrucciones del programa en código máquina que utiliza el ensamblador necesitan unos datos para funcionar y generan unos resultados que es necesario almacenar. Por ejemplo, una operación de suma necesitará conocer dónde se encuentran las dos cantidades que tiene que sumar y el lugar donde deberá almacenar el resultado una vez calculado. Existen varias formas de indicar el lugar donde se encuentran los datos que necesitan las instrucciones o donde deben depositar éstas su resultado, que reciben el nombre de "modos de direccionamiento". Los más sencillos simplemente indican un registro o una posición numérica de memoria como origen o destino de un dato. Existen además diversos métodos adicionales para el caso de posiciones de memoria, especialmente diseñados para realizar tareas frecuentes y que facilitan la tarea de programación de aplicaciones.

IMPÍCITO

El modo de direccionamiento más simple es el implícito y se da en el caso que la instrucción toma siempre los datos de un mismo lugar, por lo que no es necesario indicarselo. Un ejemplo de este tipo de direccionamiento es la instrucción LAHF, que mueve algunas de las banderas del registro de estado al registro AH.

REGISTRO

El direccionamiento a registro indica el nombre de un registro como origen o destino de los datos. La forma de indicarlo es escribiendo simplemente el nombre de dicho registro. Por ejemplo, para guardar el contenido del registro AX en la pila se utilizaría la instrucción PUSH con este modo de direccionamiento: PUSH AX

INMEDIATO

Otro de los direccionamientos más frecuentemente utilizados es el denominado "inmediato", y consiste simplemente en indicar el valor numérico necesario para la operación. Por ejemplo, la instrucción MOV AX,1234h utiliza el direccionamiento inmediato para indicar el valor (1234h) que se guardará en el registro AX. Es evidente que este modo de direccionamiento sólo puede utilizarse como origen de datos y no como destino.

DIRECTO O ABSOLUTO

Cuando lo que se indica en la instrucción es una dirección absoluta de memoria, entonces se dice que se está utilizando un direccionamiento "directo" o "absoluto", y la forma de expresarlo es encerrando la dirección de memoria de la que se obtendrá o en la que se guardará el dato entre corchetes. Por ejemplo, MOV AX,[1234h] toma el contenido de la posición de memoria 1234h y lo guarda en el registro AX.

Este modo de direccionamiento puede utilizarse también para indicar un desplazamiento como en la instrucción `MOV 1234h, AX` que mueve el contenido del registro AX a la posición de memoria numero 1234h del segmento de datos.

INDIRECTO

En ocasiones se necesita trabajar con varios datos, pero cada uno en una dirección distinta. Repetir el mismo código cambiando únicamente en cada caso la dirección desde la que obtener el dato sería altamente ineficaz. Por ello, existe el modo de direccionamiento indirecto, en el que la dirección de memoria desde la que se obtiene o en la que se guarda el dato se encuentra contenida dentro de un registro. Este sistema se utiliza, por ejemplo, para aplicar un mismo proceso a un conjunto de datos consecutivos en memoria, donde incrementando el registro se va accediendo a todos los datos.

La forma de representar este modo de direccionamiento es colocando el nombre del registro entre corchetes, como se puede ver en el siguiente ejemplo:

MOV AX,[CX]

Si, por ejemplo, CX tuviese el valor CX=1234h, se cogería el valor almacenado en la posición de memoria 1234h y se almacenaría en el registro AX.

RELATIVO A BASE

acceso a los miembros del registro

El direccionamiento relativo a base se basa en un registro que almacena una dirección de memoria, valor al cual se le añade un desplazamiento absoluto para obtener la dirección final de memoria de la que obtener el dato. El registro sólo puede ser el BX o el BP no pudiéndose utilizar ningún otro registro como base en este modo de direccionamiento.

Por ejemplo, la instrucción `MOV AX,[BX+1234h]`, suponiendo que el registro BX contiene el valor BX=2, sumaría este valor (2) al desplazamiento indicado (1234h) y el resultado (1236h) indicaría la dirección de memoria cuyo contenido se depositará en el registro AX.

Una utilización frecuente de este modo de direccionamiento es el acceso a los miembros de una variable de tipo registro. En este caso se almacena en BX la posición de inicio de la variable registro y el valor constante indica el desplazamiento del campo relativo al principio del registro. Cuando se utiliza BP como base, entonces el procesador utiliza como segmento para obtener los datos el segmento de pila, lo que hace útil este modo de direccionamiento para acceder a datos almacenados en la pila sin necesidad de recuperarlos uno a uno.

RELATIVO A ÍNDICE O INDEXADO

Una variación del direccionamiento anterior es la conocida como relativa a base o indexado. En él la dirección de memoria que contiene el dato se obtiene sumando un valor constante al contenido de un registro, pero en este caso se trata o bien del registro SI o del DI, no pudiéndose utilizar con ningún otro. Este tipo de direccionamiento se suele utilizar para acceder a matrices de una dimensión, caso en el que el valor de desplazamiento apunta a la dirección de comienzo de la matriz, y en SI se guarda el resultado de multiplicar el índice del elemento por el tamaño de cada uno de ellos. Por ejemplo, si se tiene una matriz de words que comienza en la dirección 1234h y se desea acceder al cuarto elemento, se colocaría en SI el resultado de multiplicar 4 (cuarto elemento) por 2 (el tamaño de un word), es decir, 8. Si ahora se utiliza la instrucción `MOV AX,[SI+1234h]`, se obtendría el valor del cuarto elemento.

INDEXADO CON BASE

El direccionamiento indexado con base se obtiene combinando los dos anteriores, y en él participan dos registros: uno actuando como base (BX o BP) y otro como índice (SI o DI), a cuya suma se le añade también un valor de desplazamiento para obtener la dirección de memoria que contiene el dato que se solicita. La forma de escribirlo es análoga al caso anterior, solo que ahora la suma estará compuesta por estos tres elementos. Este método se puede utilizar para acceder a una matriz de registros (BS almacena el inicio de la matriz, SI el desplazamiento del elemento y el valor constante el desplazamiento del campo dentro del registro), o también puede ser útil para acceder a matrices bidimensionales (el valor constante apunta al inicio de la matriz, BX contiene el número de fila multiplicado por el número de elementos en cada fila y su longitud y SI contiene el desplazamiento dentro de la fila).

PREFIJOS DE SEGMENTO

Cuando se especifica una dirección de memoria desde la que se obtendrá un dato, por defecto el procesador la obtiene siempre del segmento por defecto; es decir, en los modos de direccionamiento que utilizan BP como base, este segmento es el de pila (SS), mientras que para el resto se utiliza el segmento de datos (DS). Pero también es posible modificar este comportamiento simplemente añadiendo lo que se denomina "prefijos de segmento" a la instrucción. Existen cuatro prefijos diferentes, uno por cada segmento, bastando con colocar uno de ellos antes de la instrucción para que el procesador, al ejecutarla, utilice el segmento indicado en el prefijo para obtener o guardar los datos. A la hora de escribir el código, la existencia de un prefijo se representa con el nombre del segmento seguido por dos puntos colocado bien antes de un corchete abierto, bien como primer elemento dentro del corchete.

MOV AX,[BX+1234h] lee el dato de DS :BX+1234h
 MOV AX,CS :[BX+1234h] lee el dato de CS : BC+1234h

MOV AX,[BP+SI+12] lee el dato de SS :BP+SI+12
 MOV AX,[ES :BP+SI+12] lee el dato de ES :BP+SI+12

CONSIDERACIONES ACERCA DE LOS DIRECCIONAMIENTOS

Existen ciertas anomalías que es preciso tener en cuenta a la hora de trabajar con los modos de direccionamiento. La primera de ellas es que una instrucción no puede utilizar modos de direccionamiento que accedan a la memoria tanto para obtener los datos como para almacenar los resultados. Es decir, no se puede hacer que una misma instrucción acceda a dos posiciones de memoria diferentes. Será, por tanto, necesario en estos casos utilizar un registro auxiliar para guardar el resultado, que en la siguiente instrucción se enviará a memoria.

Por ejemplo, la instrucción MOV [BX],[SI+12] no es válida, pues utiliza dos direccionamientos con acceso a memoria. La forma correcta de hacer esto sería la siguiente:

MOV AX,[SI+12]
 MOV [BX],AX

Aun así, hay que señalar que el direccionamiento directo no se considera un acceso a memoria, pues el dato que se necesita se lleva al procesador al leer la instrucción. Por tanto la instrucción:

MOV [1234h],12h

es perfectamente legal. De todas formas, en este caso se plantea una ambigüedad, ya que no se puede saber si lo que se está almacenando en la posición de memoria es un valor de tamaño byte (12h) o de tamaño word (0012h). Cuando se utiliza direccionamiento inmediato con registros no sucede este problema, pues el tamaño del registro ya indica el tamaño del dato:

MOV AX,12h tipo word
 MOV AH,12h tipo byte

A pesar de ello, con la memoria existen las dos posibilidades. Para solucionar este problema se añade delante del direccionamiento la expresión "byte ptr " o "word ptr", según se quiera indicar que se almacenará un byte o un word respectivamente.

MOV byte ptr [1234h],12h Almacena un byte
 MOV word ptr [1234h],12h Almacena un word

La segunda consideración a tener en cuenta es que, al trabajar el 8086 con desplazamientos de 16 bits, es posible que la suma a partir de la que se obtiene la dirección del dato desborde esta capacidad. En este caso, lo que sucede es que se ignora dicho desbordamiento, quedándose sólo con los 16 bits menos significativos del resultado. Por ejemplo, si se tiene la instrucción:

MOV AX,[BX+1234h]

Y el registro BX tiene el valor FFFFh, entonces la suma de ambos valores sería FFFFh+1234h=11233h, que no cabe en 16 bits. Por tanto, se desecharía la primera cifra del resultado, con lo cual el dato se obtendría de la posición de memoria 1233h del segmento de datos.

No hay que preocuparse por estos desbordamientos, pues forman parte habitual del funcionamiento del 8086 y no provocan ninguna modificación del estado del registro de banderas del procesador.

Conjunto de instrucciones

El procesador 8086 cuenta con un total de 98 instrucciones diferentes, cada una de las cuales realiza una acción específica y que cuenta con su propio nombre o mnemónico. Algunas instrucciones disponen también de uno o varios mnemónicos alternativos, que si bien no varían la acción de la instrucción, se encuentran integrados dentro del lenguaje ensamblador para facilitar tanto la realización como la posterior comprensión del programa. Por ejemplo, la instrucción de salto condicional JGE provoca un salto en el programa en caso de que la última comparación tuviese como resultado que el primer valor era mayor o igual que el segundo. Por otra parte, la instrucción JNL provoca un salto si en la última comparación el primer valor no era menor que el segundo. Como se puede apreciar, ambas instrucciones son iguales, ya que si el primer número no es menor que el segundo entonces significa que es mayor o igual. Será, por tanto, elección del programador el seleccionar cuál de las dos formas se adapta a la condición que quiere expresar.

Cada una de las instrucciones admite unos modos de direccionamiento determinado, tanto para indicar de dónde recoge los datos como dónde deposita los resultados, existiendo varias opciones según la instrucción. Así pues, algunas instrucciones utilizan siempre determinados registros como fuente o destino de sus datos, sin posibilidad de cambio. Otras permiten utilizar algunos modos de direccionamiento determinados y no admiten la

utilización de otros. Cada instrucción es diferente y funciona de una manera determinada, que se verá en el estudio concreto de cada una de ellas.

La longitud de cada instrucción es variable y depende principalmente de los modos de direccionamiento que utilice, pudiendo abarcar desde un byte (el mínimo) hasta un total de siete bytes en el caso de instrucciones que utilizan el direccionamiento indexado con base en el destino e inmediato para el origen y que tenga además un prefijo de segmento. Junto a cada instrucción se ha incluido una tabla que indica los modos de direccionamiento que admite, tanto como fuente como para resultado. No obstante, debe tenerse en cuenta el hecho de que una instrucción que coge un dato de memoria no puede depositar un resultado en la memoria, y deberá utilizar un registro auxiliar. La siguiente tabla resume las combinaciones posibles de modos de direccionamiento:

Fuente Destino	Implicito	Registro	Inmediato	Otro
Implicito	SI	SI	SI	SI
Registro	SI	SI	SI	SI
Otro	SI	SI	SI	NO

TRANSFERENCIA DE DATOS

El primer gran grupo de instrucciones, y también las más utilizadas, son las denominadas de transferencia de datos. Su misión es únicamente la de copiar datos sin modificarlos de un lugar a otro. Estas instrucciones son capaces de trabajar tanto con datos de 8 como de 16 bits.

GENERALES

Las instrucciones de transferencia generales se utilizan para transferir datos entre la memoria y los registros del procesador o entre los mismos registros. Existen en total tres instrucciones que realizan este traslado de forma diferente, a las que se les ha añadido la instrucción de "no operación", dado que puede considerarse como una operación que en realidad no realiza ninguna transferencia.

MOV

La instrucción MOV es la más utilizada, fundamentalmente debido a la imposibilidad del 8086 de acceder a dos posiciones de memoria en la misma instrucción y a que algunas instrucciones necesitan que sus operandos se encuentren en ciertos registros o que dejen su resultado en otros, lo cual obliga a realizar frecuentes trasvases de datos entre la memoria y los registros o entre registros.

Esta instrucción cuenta con dos operandos, el segundo de los cuales indica el lugar donde se encuentra el dato, y el primero dónde se va a copiar dicho dato.

Un detalle importante es que si el destino es un registro de segmento, entonces el origen no puede ser un valor inmediato, sino que debe ser otro registro o una referencia a memoria.

Nombre: MOV (MOVE = mover)

Acción: destino <- fuente

Modos de direccionamiento: Registro, Inmediato, Directo, Indirecto, Relativo a base, relativo a índice, indexado con base

Banderas modificadas: Ninguna.

Ejemplo:

```
MOV AX,1234h ;AX = 1234h
MOV [SI+1234h],AX ;DS:[SI+1234h] = 1234h
MOV BX,AX ;BX = 1234h
MOV [BP+SI+1234h],BX ;SS:[BP+SI+1234h]=1234h
```

XCHG

Algunas veces no sólo se desea hacer una copia de un dato en otro lugar, sino que además se desea que suceda el proceso inverso, es decir, la copia del operando destino en el fuente, de forma que se produzca un intercambio de los datos. Esta instrucción recibe el nombre de XCHG y resulta útil en situaciones donde hay que colocar un dato en un lugar pero, a la vez, se desea preservar el valor anterior que allí había.

Nombre: XCHG (eXCHanGe = intercambiar)

Acción: destino <-> fuente

Modos de direccionamiento: Registro, Directo, Indirecto, Relativo a base, Relativo a índice, Indexado con base

Banderas modificadas: Ninguna.

Ejemplo:

```
MOV AX,1234h ;AX= 1234h
MOV BX,5678h ;BX= 5678h
XCHG AX,BX ;AX= 5678h, BX=1234h
MOV [1111h],AX ;DS:[1111h] = 5678h
XCHG BX,[1111h] ;DS:[1111h] = 1234h, BX = 5678h
```

XLAT

La instrucción XLAT utiliza el valor sin signo contenido en el registro AL como índice dentro de una tabla de 256 bytes apuntada por la pareja de registros DS:BX, recuperando el byte cuyo índice es el indicado en dicho

CÓMO PROGRAMAR EN ENSAMBLADOR

registro y depositándolo en el mismo registro AL. Puede observarse, por tanto, que esta instrucción utiliza como operandos implícitos los registros AL y BX, y que la instrucción no requiere ningún operando, salvo en el caso que BX apunte a una tabla contenida en un segmento diferente al de datos; en cuyo caso puede colocarse como operando una referencia a una posición de memoria, de la que sólo se tendrá en cuenta el prefijo de segmento.

Esta instrucción resulta útil, por ejemplo, para realizar conversiones de unos juegos de caracteres a otros.

Nombre: XLAT (TranslATE = traducir)

Acción: AL <- DS :[BX+AL]

Modos de direccionamiento: Implicito.

Banderas modificadas: Ninguna

Ejemplo: (se supone que DS y ES apuntan a diferentes segmentos)

```
MOV BX,1234h ;BX= 1234h
MOV byte ptr [1236h],12h ;DS:[1236h]=12h
MOV byte ptr ES :[1236h],34h ;ES:[1236h]=34h
MOV AL,2 ;AL=2
XLAT ;AL=DS :[1234h+2]=DS :[1236h]=12h
MOV AL,2 ;AL=2
XLAT ES :[1111h] ;AL=ES :[1234h+2]=ES :[1236h]=34h (el valor 1111h no se utiliza)
```

NOP

La ultima de las instrucciones de transferencia, o quizás sería más adecuado describirla como de "no transferencia", pues no realiza ninguna función, es la instrucción NOP, la cual no necesita ningún parámetro ni modifica ninguna bandera del procesador. Aunque parezca que no posee ninguna utilidad, esta instrucción se utiliza frecuentemente para llenar código o para crear retardos en el programa, como los que necesitan algunas tarjetas entre dos envíos de datos consecutivos a un puerto de entrada/salida. En este último caso se debe tener en cuenta que una instrucción NOP tarda tres ciclos de reloj en ejecutarse, salvo en los procesadores 486 y superiores, que sólo tarda uno.

En realidad, la instrucción NOP equivale a XCHG AX,AX, pero se utiliza este nombre por resultar más descriptivo de su función.

Nombre: NOP (No Operation = sin operación)

Acción: Ninguna.

Modos de direccionamiento: Ninguno

Banderas modificadas: Ninguna

Ejemplo:

MOV AX,1001h	;AX=1001h
OUT [80h],AL	;Envía al puerto 80h el valor 01h
MOV AL,AH	;AL=10h
NOP	;retardo
NOP	;retardo
OUT [80h],AL	;Envía al puerto 80h el valor 10h

DE ENTRADA/SALIDA

Instrucciones

Ya se comentó que el 8086 dispone de una memoria auxiliar destinada especialmente a la comunicación con dispositivos y periféricos auxiliares. Esta memoria no existe en un principio, y sus posiciones las van llenando los dispositivos que la utilizan para comunicarse con el procesador. Aunque su comportamiento es similar al de la memoria convencional, no pueden utilizarse las funciones habituales de transferencia para acceder a ella, y deben utilizarse un par de instrucciones especiales, que además cuentan únicamente con un modo de direccionamiento, lo que dificulta su utilización.

Las instrucciones de entrada/salida pueden transferir tanto datos de 8 como de 16 bits. No obstante, el dato debe encontrarse en el registro AL o AX en el caso de envío o, en el caso de recepción, se obtiene también en dichos registros.

Con AL o AX

IN

Puerto

E/S

La instrucción IN recibe un dato de un puerto de entrada/salida, que se recibe en el registro AL en caso que sea un dato de 8 bits, o en el AX si es de 16. Hay que tener en cuenta que en este último caso (16 bits) los 8 bits inferiores se obtienen del puerto indicado, mientras que los 8 superiores se obtienen del siguiente puerto de entrada/salida. Es un error frecuente pensar que los 16 bits se obtienen del mismo puerto, primero los ocho bajos y mediante una segunda lectura los 8 altos, por lo que hay que tener este hecho muy en cuenta.

El puerto desde el que se reciben datos puede indicarse de dos formas: la más general consiste en colocar en el registro DX la dirección del puerto a la que se desea acceder; pero si además se trata de un puerto entre 0 y 255 puede especificarse directamente su dirección mediante direccionamiento directo.

Nombre: IN (IN = Entrada)

Acción: AL <- Puerto(DX) y si 16 bits: AH <- Puerto(DX+1)

Modos de direccionamiento: Implícito, Directo.

Banderas afectadas: Ninguna.

Ejemplo:

IN AL,[80h]	; Coge un dato de tipo byte del puerto 80h
MOV BL,AL	; y lo guarda en BL
MOV DX,1234h	; Para leer de puertos altos hay que poner su dirección en DX
IN AX,DX	; AL se coge del puerto 1234h y AH del puerto 1235h

• OUT

La operación inversa de IN, es decir, el envío de datos a puertos, es llevada a cabo a través de la instrucción OUT, complementaria de la anterior. Su funcionamiento es análogo al de la instrucción IN, sólo que en este caso los argumentos invertirán su orden debido al hecho de que en todas las instrucciones el destino de la operación debe aparecer como primer argumento.

Nombre: OUT (OUT=Salida)

Acción: Puerto(DX) <- AL y si 16 bits : puerto(DX+1) <- AH

Modos de direccionamiento: Implícito, directo

Banderas afectadas: Ninguna

Ejemplo:

MOV AL,12h	; AL=12h
OUT [80h],AL	; Envía al puerto 80h el valor 12h
MOV AX,5678h	; AX=5678h
MOV DX,1234h	; Coloca en DX el número de puerto (1234h)
OUT DX,AX	; Envía al puerto 1234h el valor 78h ó al 1235h el valor 56h

DE BANDERAS

Instrucciones

Las instrucciones de transferencia de banderas mueven todas o algunas de la banderas hacia otro lugar, de forma que puedan recuperarse más tarde sus valores o modificar algunos de las banderas de control que no pueden cambiarse de forma directa, como es el caso de la bandera de depuración paso a paso.

• LAHF

Transfiere el byte bajo de banderas al registro AH, es decir, las banderas que contienen el resultado de la última operación; como si fue cero, negativa o con acarreo. Su principal finalidad es, por tanto, guardar el resultado de una comparación para utilizarlo más tarde en el programa.

Nombre: LAHF (Load AH with Flags = Carga AH con las banderas)

Acción: AH <- byte bajo de banderas

Modos de direccionamiento: Implícito

Banderas modificadas: Ninguna

• SAHF

La operación inversa a la instrucción anterior la lleva a cabo SAHF, colocando el valor contenido en el registro AH en el byte bajo del registro de banderas.

Nombre: SAHF (Store AH into Flags = Almacena AH en las banderas)

Acción: Byte bajo de banderas <- AH

Modos de direccionamiento: Implícito

Banderas modificadas: Ninguna

Ejemplo:

LAHF	; Guarda las banderas en AH
OR AH,1	; Coloca el indicador de acarreo a 1 (bit 0)
SAHF	; y lo almacena de nuevo en las banderas

• PUSHF

En caso de que se quieran guardar todas las banderas, el único sistema es enviarlas a la pila mediante la instrucción PUSHF. Esta instrucción funciona de la misma forma que la instrucción PUSH, pero en este caso no hay ningún argumento y lo que se envía a la pila es siempre el registro completo de banderas. Una vez en la pila pueden consultarse o modificarse las banderas mediante acceso relativo a base o recuperarlos a un registro. Este sistema es la única forma de modificar algunas banderas, como por ejemplo la de depuración paso a paso.

Nombre: PUSHF (PUSH Flags = guarda banderas en la pila)

Acción: SP-< SP-2 , SS:[SP]<- Banderas

Modos de direccionamiento: Implícito

Banderas modificadas: Ninguna

• POPF

Recupera el valor de tipo word situado en la cima de la pila y lo almacena en el registro de banderas.

Nombre: POPF (POP Flags = Recupera banderas de la pila)

Acción: Flags<-SS:[SP] , SP-<SP+2

Modos de direcciónamiento: Implícito

Banderas modificadas: Ninguna

Ejemplo:

PUSHF	;Guarda banderas en la pila
POP AX	;Y las lee al registro AX
XOR AX,1	;Invierte la bandera de acarreo
PUSH AX	;Guarda el registro de banderas modificado en la pila
POPF	;Y lo recupera al registro de banderas.

DE DIRECCIONES Instrucciones

Las instrucciones de transferencia de direcciones cargan datos en registros, pero en este caso lo que se cargan son direcciones que pueden utilizarse a continuación para acceder a otros datos mediante direccionamiento relativo a registro, a base, a índice o indexado con base. El destino de todas estas instrucciones es siempre un registro, y el origen de los datos es la memoria.

ORIGEN = MEMORIA
DESTINO = REGISTRO

• LEA

El formato de la instrucción LEA es similar al de la instrucción MOV, pero en este caso el destino es siempre un registro de 16 bits y el origen es una referencia a memoria. La función de esta instrucción es la de cargar en el registro la dirección de memoria a la que hace referencia el segundo operando de la instrucción. Esta instrucción resulta útil sobre todo para calcular a qué dirección de memoria hace referencia un tipo de direccionamiento complejo, como puede ser el indexado con base.

Nombre: LEA (Load Effective Adress = Cargar dirección efectiva)
Acción: registro16 <- dirección (operando2)

Modos de direccionamiento: Destino: registro 16, origen: relativo a registro, relativo a base, indexado e indexado con base

Banderas modificadas: Ninguna.

Ejemplo:

LEA AX,[1234h]	;AX=1234h
MOV SI,1234h	;SI=1234h
LEA AX,[SI+4h]	;AX=1234h+4h=1238h
MOV BP,12h	;BP=12h
LEA AX,[BP+SI+3h]	;AX=12h+1234h+3h=1249h

• LDS

La instrucción LDS sirve para cargar una dirección completa, compuesta por un segmento y un desplazamiento, en un registro del procesador. Para ello, el word hacia el que apunta el segundo operando (fuente) de la instrucción se considera el desplazamiento de la dirección y se carga en el registro indicado como primer operando. El siguiente word, dos bytes más adelante, se carga en el registro de segmento DS. De esta forma, a continuación se puede realizar un direccionamiento relativo a registro para acceder a un dato en memoria.

Nombre: LDS (Load pointer with DS= Cargar puntero con DS)

Acción: registro16<-[operando2], DS<-[operando2+2]

Modos de direccionamiento: Destino: registro16, Origen: relativo a registro, relativo a base, indexado e indexado con base

Banderas modificadas: Ninguna

Ejemplo:

MOV AX,0 ;Carga el valor 0 en AX

MOV DS,AX ;Selecciona el segmento 0

LDS SI,[0004h] ;Carga la dirección de inicio de la interrupción 01h

• LES

La instrucción LES, por su parte, es análoga a LDS, con la única diferencia que el registro de segmento utilizado para cargar el segmento de la dirección es el ES en lugar del DS.

Nombre: LES (Load pointer with ES = Cargar puntero con ES)

Acción: registro16<-[operando2], ES<-[operando2+2]

Modos de direccionamiento: Destino: registro16, Origen: relativo a registro, relativo a base, indexado e indexado con base

Banderas modificadas: Ninguna.

DE PILA Instrucciones

Aunque ya se trató el funcionamiento de la pila en el capítulo anterior, no se indicaron en aquel momento las instrucciones de que dispone el procesador para almacenar y recuperar datos en ella. Únicamente existen dos instrucciones, una para guardar datos en la pila y otra para recuperarlos. Ambas llevan un único parámetro que hace referencia al dato que se desea guardar o al lugar donde dejar el dato recuperado de la pila.

• PUSH 16 bits = 2 bytes = 1 Registro Completo

Es la instrucción que se utiliza para guardar un dato en la pila. Para hacerlo coge el word al que hace referencia el único parámetro de la instrucción y lo almacena en la cima de la pila, modificando el puntero de pila para que apunte hacia este nuevo dato. El parámetro de la instrucción PUSH puede ser un registro de 16 bits o una referencia a memoria, pero no puede tratarse de un valor inmediato.

Nombre: PUSH (PUSH operand onto the stack = Guarda operando en la pila)

Acción: SP<-SP-2, SS:[SP]<- operando

Modos de direccionamiento: Registro, indirecto, relativo a registro, relativo a base, indexado, indexado con base

Banderas modificadas: Ninguna.

• POP

16 bits = 2 bytes = 1 Registro Completo

La instrucción POP es la complementaria de PUSH, recogiendo el dato de 16 bits situado en la cima de la pila y colocándolo en el lugar indicado por el único operando de la instrucción. Este destino puede ser tanto un registro de 16 bits como una posición de memoria cualquiera.

Nombre: POP (POP word from stack = Coge word de la pila)

Acción: destino<-SS:[SP], SP<-SP+2

Modos de direccionamiento: Registro, indirecto, relativo a registro, relativo a base, indexado, indexado con base.

Banderas modificadas: Ninguna

Ejemplo:

```
MOV AX,1234h ;AX=1234h
PUSH AX
MOV AX,[SP] ;AX=5678h
PUSH AX
MOV AX,1A74h ;AX=1A74h
POP AX ;AX=5678h
POP BX ;BX=1234h
```

• ARITMÉTICAS

INSTRUCCIONES

El segundo grupo en orden de importancia de instrucciones lo constituyen las instrucciones aritméticas. Su misión es la de realizar operaciones matemáticas con los datos, tales como suma, resta, multiplicación o división. Existen dos grandes grupos de operaciones: las que se realizan con cantidades binarias y las que se realizan sobre cantidades expresadas en sistema decimal codificado en binario, conocido como BCD.

• BINARIAS

Instrucciones Aritméticas binarias

Las instrucciones aritméticas binarias son las más utilizadas y, como su nombre indica, trabajan con cifras expresadas en formato binario, tanto de 8 como de 16 bits. Estas cifras pueden ser tanto cantidades con signo como sin él, existiendo en el caso de la multiplicación y la división instrucciones específicas para cada uno de estos dos casos.

• ADD

La instrucción ADD realiza la suma de las cantidades enteras, tanto de 8 como de 16 bits, indicadas por sus operandos fuente y destino, y deposita el resultado dentro del lugar indicado por el primer operando. Las banderas son modificadas para indicar el resultado de la operación.

Nombre: ADD (ADD = sumar)

Acción: Destino<-destino+origen

Modos de direccionamiento: Destino: registro, indirecto, relativo a base, indexado, indexado con base. Origen: inmediato, registro, indirecto, relativo a base, indexado, indexado con base.

Banderas modificadas: OF, SF, ZF, AF, PF, CF

Ejemplo:

MOV AX,1234h	;AX=1234h
MOV BX,1111h	;BX=1111h
MOV [1234h],2222h	;DS:[1234h]=2222h
ADD AX,BX	;AX=1234h+1111h=2345h
ADD [1234h],AX	;DS:[1234h]=2222h+2345h=4567h

• ADC = ADD + ACARREO

ADC es similar en su funcionamiento a la instrucción ADD; pero a la suma de sus dos operandos se le añade también la bandera de acarreo. Puesto que el bit de acarreo toma el valor 1 si una operación de suma produce un desbordamiento, esta instrucción es útil para realizar sumas de cantidades de cualquier longitud. Las banderas son modificadas de la misma forma que en la instrucción ADD.

Nombre: ADC (ADd with Carry = Sumar con acarreo)

Acción: Destino<-Destino+Origen+CF

Modos de direccionamiento: Destino: registro, indirecto, relativo a base, indexado, indexado con base. Origen: inmediato, registro, indirecto, relativo a base, indexado, indexado con base.

Banderas modificadas: OF, SF, ZF, AF, PF, CF

Ejemplo: Suma dos cantidades de 48 bits, la primera situada en [1234h] y la segunda en [5678h]. Obsérvese que la operación se hace a nivel de word (podría haberse hecho byte a byte, pero en ese caso se necesitarían el doble de instrucciones de suma). Puesto que el 8086 almacena las cantidades en orden inverso, con los bytes menos significativos en primer lugar, la suma se hace desde el primer word en memoria hacia posiciones de memoria.

MOV AX,[5678h]	;Coge el word bajo del segundo operando
ADD [1234h],AX	;Lo suma al word bajo del destino
MOV AX,[567Ah]	;Lee el siguiente word
ADC [1236h],AX	;Lo suma al segundo del destino, esta vez con acarreo
MOV AX,[567Ch]	;Lee el último word
ADC [1238h],AX	;Y lo suma al word más significativo del destino

• SUB

La resta entre dos cantidades enteras de 8 o de 16 bits se lleva a cabo

mediante la instrucción SUB. Esta instrucción deduce del primer operando la cantidad indicada por el segundo, almacenando el resultado de nuevo en el lugar del primer operando. En caso que el segundo valor sea mayor que el primero, se colocará la bandera de acarreo a uno, indicando que se "debe" una unidad en el resultado.

Nombre: SUB (SUBtract = restar)

Acción: Destino <- Destino - Origen

Modos de direccionamiento: Destino: registro, indirecto, relativo a base, indexado, indexado con base. Origen: inmediato, registro, indirecto, relativo a base, indexado, indexado con base.

Banderas modificadas: OF, SF, ZF, AF, PF, CF

Ejemplo:

```
MOV AX,1234h ;AX=1234h
MOV BX,1111h ;BX=1111h
SUB AX,BX ;AX=0123h
MOV byte ptr [1234h],12h ;DS:[1234h]=12h
SUB [1234h],AH ;DS:[1234h]=12h-01h=11h
```

• SBB

= SUB + ACARREO

Al igual que ocurría con la suma, es posible enlazar varias sustracciones, de forma que se obtenga la resta de cantidades de tamaño superior al tipo word. Para ello, la instrucción SBB añade en primer lugar el bit de acarreo al segundo operando y a continuación resta el resultado del primero, depositando en este último el resultado de la operación.

Nombre: SBB (Subtract with Borrow = Restar con débito)

Acción: Destino <- Destino-Origen+CF

Modos de direccionamiento: Destino: registro, indirecto, relativo a base, indexado, indexado con base. Origen: inmediato, registro, indirecto, relativo a base, indexado, indexado con base

Banderas modificadas: OF, SF, ZF, AF, PF, CF

Ejemplo: que resta dos cantidades de 48 bits, una situada a partir de [1234h] y otra en [5678h]. La operación se realiza word a word, dejándose el resultado en [1234h]:

```
MOV AX,[5678h] ;Carga en AX el word bajo
SUB [1234h],AX ;Lo resta del destino
MOV AX,[S67Ah] ;Carga siguiente word
SBC [1236h],AX ;Lo resta
NOTA
MOV AX,[567Ch] ;Carga word alto
SBC [1238h],AX ;Lo resta del destino
```

• INC Contador.

El propósito de la instrucción INC es el de sumar una unidad a su operando y, aunque es equivalente a una instrucción de suma cuyo segundo operando es el valor inmediato 1, se encuentra mucho más optimizada y tarda menos en ejecutarse, lo que la hace especialmente indicada para su uso con contadores de bucles. Aunque conviene tener en cuenta que esta instrucción no actualiza la bandera de acarreo en caso de que el resultado supere la capacidad máxima del destino, condición que puede ser comprobada a través de la bandera de cero.

Nombre: INC (INCrement = Incrementar)

Acción: Destino <- Destino+1

Modos de direccionamiento: Registro, indirecto, relativo a base, indexado, indexado con base.

Banderas modificadas: OF, SF, ZF, AF, PF

Ejemplo:

```
MOV AX,1234h ;AX=1234h
INC AX ;AX=1235h
```

• DEC

DEC se utiliza para decrementar el contenido del primer operando en una unidad. Es equivalente a la instrucción SUB, cuyo segundo parámetro es el valor inmediato 1, con la única salvedad de que la bandera de acarreo no es actualizada por esta instrucción. Su principal utilidad es la de actualizar la variable de control de un bucle en cada pasada.

Nombre: DEC (DECrement = Decrementar)

Acción: Destino <- Destino-1

Modos de direccionamiento: Registro, indirecto, relativo a base, indexado, indexado con base.

Banderas modificadas: OF, SF, ZF, AF, PF

Ejemplo:

```
MOV AX,1234h ;AX=1234h
DEC AX ;AX=1233h
```

• MUL

La multiplicación de dos cantidades sin signo la lleva a cabo la instrucción MUL. Su funcionamiento es un tanto especial, pues únicamente requiere un parámetro que indica una de las cifras que intervienen en el producto. En caso de que dicho operando sea de tipo byte, el otro operando necesario para la multiplicación deberá encontrarse en el registro AL y el resultado de la multiplicación quedará almacenado en el registro AX.) Si el operando de la instrucción MUL es de tipo word, entonces el

segundo operando deberá encontrarse en el registro AX, y el resultado del producto se almacenará en la pareja de registros DX:AX, conteniendo AX los 16 bits menos significativos del resultado y DX los 16 superiores.

Esta instrucción no acepta el modo de direccionamiento inmediato, por lo que el número a multiplicar deberá encontrarse en memoria o en un registro del procesador. Además, puede observarse que el resultado siempre tiene doble tamaño que los operandos (16 bits para multiplicación de 8 y 32 para la de 16). Si el resultado de una multiplicación de 8 bits cabe en 8 bits (es decir, AH=0), entonces las banderas OF y CF toman el valor 0. Para la multiplicación de 16 bits sucede de la misma forma, que si el resultado excede de los 16 bits (DX distinto de 0), entonces las banderas OF y CF toman el valor 1.

Nombre: MUL (Multiplication = multiplicación)

Acción: byte: AX<-AL*operando word: DX :AX<-AX*operando

Modos de direccionamiento: Registro, indirecto, relativo a base, indexado, indexado con base.

Banderas modificadas: OF, CF. Además SF,ZF,AF y PF quedan con valores indefinidos.

Ejemplo que multiplica dos cantidades y almacena el resultado en memoria:

```
MOV AX,[1234h] ;Guarda primer operando en AX
MOV BX,2 ;BX=2
MUL BX ;Multiplica por 2
MOV [5678h],AX ;Guarda la parte baja del resultado
MOV [567Ah],DX ;Guarda la parte alta
```

• IMUL

IMUL tiene un funcionamiento análogo a la instrucción MUL, pero en este caso las cantidades a multiplicar son enteros con signo. Las banderas OF y CF se utilizan igual que en la instrucción MUL para indicar que fue necesario ampliar la precisión del resultado (el número de bits) para poder contener el producto de ambos operandos.

Nombre: IMUL (Integer Multiplication = multiplicación de enteros)

Acción: byte: AX<-AL*operando word: DX :AX<-AX*operando

Modos de direccionamiento: Registro, indirecto, relativo a base, indexado, indexado con base

Banderas modificadas: OF, CF. Además SF,ZF,AF y PF quedan con valores indefinidos.

• DIV

La división de dos cantidades binarias la realiza la instrucción DIV, la cual, al igual que sucedía con la multiplicación, necesita un único argumento que indica el lugar donde se encuentra el divisor. En caso de que el divisor sea de 8 bits, se toma como dividendo el valor depositado en el registro AX, dejando el cociente en el registro AL y el resto de la división en AH. Si el divisor es una cantidad de 16 bits, entonces se toma como dividendo el valor de 32 bits formado por la concatenación de los registros DX:AX, conteniendo DX el word más significativo de la cantidad y AX el menos. Tras la operación, el registro AX contendrá el cociente y el DX el resto.

En caso que se produzca un desbordamiento o una división por cero, se generará una interrupción 0, que por defecto muestra un error en pantalla y finaliza.

Nombre: DIV (Divide = Dividir)

Acción: byte: AL<-cociente(AX / operando), AH<-resto (AX/operando)

word: AX<-cociente(DX :AX / operando), DX<-resto (DX:AX/operando)

Modos de direccionamiento: Registro, indirecto, relativo a base, indexado, indexado con base

Banderas modificadas: OF,SF,ZF,AF, PF y CF quedan con valores indefinidos.

Ejemplo:

```
MOV AX,[5678h] ;Lee parte baja del dividendo
MOV DX,[567Ah] ;Lee parte alta del dividendo
MOV BX,2 ;Guarda el divisor en BX
DIV BX ;Realiza la división. Ahora AX contiene el
;cociente y DX el resto
```

• IDIV

Cuando las cantidades que se van a dividir tienen signo, entonces debe utilizarse la instrucción IDIV. El funcionamiento es idéntico al de DIV, pero se tienen en cuenta los signos de dividendo y divisor para calcular el resultado. El cociente de la división se redondea siempre hacia cero, el resto tiene el mismo signo que el dividendo y su valor absoluto es siempre menor que el valor absoluto del divisor.

Nombre: IDIV (Integer Divide = Dividir enteros)

Acción: byte: AL<-cociente(AX / operando), AH<-resto (AX/operando)

word: AX<-cociente(DX :AX / operando), DX<-resto (DX:AX/operando)

Modos de direccionamiento: Registro, indirecto, relativo a base, indexado, indexado con base

Banderas modificadas: OF,SF,ZF,AF, PF y CF quedan con valores indefinidos.

• NEG

La función de la instrucción NEG es la de cambiar el signo del número que se pasa como argumento, es decir, obtener su complemento a dos. Tras la conversión, la bandera de acarreo se encuentra siempre al valor 1, salvo que el operando sea cero, en cuyo caso el acarreo toma el valor 0.

Nombre: NEG (NEGation : Opuesto)

Acción: operando<-0-operando

Modos de direccionamiento: Registro, indirecto, relativo a base, indexado, indexado con base.

Banderas modificadas: OF, SF, ZF, AF, PF y CF.

Ejemplo:

```
MOV AX,1234h ;AX=1234h
NEG AX; ;AX=EDCCh
ADD AX,1234h ;AX=0
```

BCD DESEMPAQUETADO

El sistema binario es la mejor opción para la gran mayoría de los programas. No obstante, cuando se trabaja con cifras que contienen decimales surgen algunos inconvenientes. El principal de ellos es que algunas cifras con decimales que tienen representación exacta en base 10 no la tienen en binario, por lo que al realizar la conversión decimal-binario-decimal se produce una pérdida de precisión. Estas inexactitudes no tienen importancia salvo en casos muy especiales, como son las aplicaciones financieras.

Por ello, el 8086 ofrece la posibilidad de trabajar con una forma especial de representación de números decimales, denominada BCD (Decimal codificado en binario), y que consiste en utilizar varios bits para representar las diez cifras del sistema decimal. El sistema más sencillo de hacer esto recibe el nombre de BCD desempaquetado, y consiste en guardar cada cifra del número decimal en un byte, que puede adoptar los valores del 0 al 9, dejándose el resto de valores posibles del byte sin utilizar.

La forma de operar con estas cifras es análoga a la realizada para las cantidades binarias, pero tras cada operación hay que realizar una serie de correcciones al resultado para ajustar los valores ilegales que se hayan producido a los correctos. Estas instrucciones de ajuste no poseen operandos, y suponen que el valor que deben corregir se encuentra en el registro AX.

Otra ventaja de los valores BCD desempaquetados es que su conversión desde ASCII a BCD es muy sencilla.

Para pasar de ASCII a BCD hay que realizar un AND con el valor CFh, mientras que para pasar de BCD a ASCII hay que hacer un OR con 30h.

Utilizando este sistema, las cantidades pueden expresarse en orden normal, aunque a la hora de realizar operaciones hay que aplicar el proceso a cada dígito de derecha a izquierda. Por ejemplo, si se quisiese guardar el valor 9768 decimal en BCD se haría así:

9	7	6	8
byte1	byte2	byte3	byte4

• AAA

AAA realiza el ajuste BCD necesario tras una suma. Para hacer esto, coge el valor depositado en el registro AX y si AL contiene un valor incorrecto lo corrige, y si la bandera de acarreo decimal tiene el valor 1, se le añade una unidad al valor contenido en AH. En este último caso además se activan los indicadores de acarreo normal y auxiliar.

Nombre: AAA (ASCII Adjust after Addition = Ajuste ASCII tras suma)

Acción: AL<-AL ajustado

Modos de direccionamiento: Implícito.

Banderas modificadas: AF, CF. Además OF, SF, ZF y PF quedan con valores indefinidos.

Ejemplo:

```
MOV AL,03h ;AL=3 (BCD)
MOV BL,08h ;BL=8 (BCD)
ADD AL,BL ;AL=0Bh
AAA ;AL=1 (BCD) AH=1 (Hay acarreo)
```

• AAS

La instrucción AAS se utiliza tras una resta de dos cantidades en BCD desempaquetado para ajustar su resultado. Para ello ajusta el valor contenido en el registro AL y, si hubo acarreo decimal, se decrementa el registro AH. Las banderas de acarreo y acarreo auxiliar toman el valor del acarreo decimal.

Nombre: AAS (ASCII Adjust after Subtraction = Ajuste ASCII tras resta)

Acción: AL<-AL ajustado

Modos de direccionamiento: Implícito.

Banderas modificadas: AF, CF. Además OF, SF, ZF y PF quedan con valores indefinidos.

Ejemplo:

```

MOV AL,02h    ;AL=2 (BCD)
MOV BL,09h    ;BL=9 (BCD)
SUB AL,BL     ;AL=FAh
AAS           ;AL= 4 (BCD) AH=FF (Hay acarreo)

```

• AAM

Ajusta el valor contenido en el registro AX tras una multiplicación de dos números en BCD desempaquetado. Puesto que el resultado es como mucho 81 (9*9), el resultado cabe completamente en AL, y lo único que debe hacer esta instrucción es dividir esta cantidad entre diez, colocando el cociente en AH y el resto en AL. Puede apreciarse que esta instrucción lo que hace en realidad es una conversión de un valor en binario a uno en decimal, por lo que en ocasiones se utiliza para este propósito.

Nombre: AAM (ASCII Adjust after Multiply = Ajuste ASCII tras multiplicación)

Acción: AX<-AL en BCD desempaquetado

Modos de direccionamiento: Impícito.

Banderas modificadas: SF,ZF y PF. Además OF,AF y CF quedan con valores indefinidos.

Ejemplo:

```

MOV AL,08h    ;AL=8 (BCD)
MOV BL,04h    ;BL=4 (BCD)
MUL BL        ;AL=20h
AAM           ;AX=0302h=32 (BCD)

```

• AAD

La instrucción de ajuste para la división se denomina AAD, y tiene la peculiaridad de que debe utilizarse antes de la operación de división sobre el dividendo y el divisor. En realidad, su función es convertir el valor en BCD desempaquetado a su equivalente binario, por lo que a veces es utilizado para esta función alternativa.

Nombre: AAD (ASCII Adjust before Division = Ajuste ASCII antes de división)

Acción: AL<-AX en binario

Modos de direccionamiento: Impícito.

Banderas modificadas: SF,ZF y PF. Además OF,AF y CF quedan con valores indefinidos.

Ejemplo:

```

MOV AX,0108h  ;AX=0108h=18 (BCD)
AAD           ;AL=12h=18d

```

BCD EMPAQUETADO

Existe una forma más óptima de almacenar los números en formato BCD, consistente en utilizar únicamente cuatro bits para cada dígito. De esta manera, se pueden almacenar dos dígitos en cada byte, ahorrando espacio de almacenamiento pero a costa de complicar los algoritmos de las operaciones. De hecho, el 8086 sólo propone dos instrucciones para este tipo de representación para ajustar los resultados de la suma y de la resta de cantidades en este formato. El formato BCD empaquetado puede considerarse como un sistema hexadecimal en el que sólo son válidas las cifras del 0 al 9.

• DAA

Realiza el ajuste necesario tras una suma de dos bytes que contienen cada uno de ellos dos cifras en BCD empaquetado.

Nombre: DAA (*Decimal Adjust after Addition = Ajuste decimal tras suma*)

Acción: AL<-AL ajustado.

Modos de direccionamiento: Impícito.

Banderas modificadas: SF,ZF,AF, PF y CF. Además, OF queda con un valor indefinido.

Ejemplo:

```

MOV AL,12h    ;AL=12 (BCD)
MOV BL,29h    ;BL=29 (BCD)
ADD AL,BL     ;AL=3Bh
DAA           ;AL=41h=41(BCD)

```

• DAS

Tras la resta de números en BCD empaquetado debe utilizarse la instrucción DAS, que ajusta su resultado, quedando en AL un valor válido, así como las banderas correspondientes actualizadas.

Nombre: DAS (*Decimal Adjust after Subtraction = Ajuste decimal tras resta*)

Acción: AL<-AL ajustado

Modos de direccionamiento: Impícito

Banderas modificadas: SF,ZF,AF, PF y CF. Además, OF queda con un valor indefinido.

Ejemplo:

```

MOV AL,41h    ;AL=41 (BCD)
MOV BL,18h    ;BL=18 (BCD)
SUB AL,BL     ;AL=29h
DAS           ;AL=23h=23 (BCD)

```

que se hace es eliminar tantas cifras de la derecha como indica la potencia de 10.

El sistema binario cuenta con esta misma propiedad para multiplicar o dividir por potencias de 2. Así, para multiplicar una cantidad binaria por 4 basta añadir dos ceros a la derecha de la cifra binaria ($4 = 2^2$). La división se realiza de la misma forma eliminando cifras de la parte derecha de la cantidad.

Las instrucciones de desplazamiento de bits realizan esta operación, resultando por tanto especialmente indicadas para realizar multiplicaciones y divisiones por potencias de dos de una forma muy rápida. Aparte de esto, a estas instrucciones se les puede dar otras múltiples aplicaciones, como procesar bit a bit una cantidad o realizar un scroll de una imagen gráfica.

Existen dos tipos de instrucciones de desplazamiento: de desplazamiento aritmético y de desplazamiento lógico. Las de tipo lógico trabajan con cantidades sin signo y, por tanto, rellenan los bits con 0. Por su parte, las de tipo aritmético están diseñadas para aplicarse a cantidades con signo, por lo que la de desplazamiento hacia la derecha rellena copiando el bit más significativo; es decir, realizando la extensión de signo necesaria para que el resultado sea correcto.

Todas las instrucciones de desplazamiento cuentan con dos parámetros, de los que el primero indica dónde se encuentra la cantidad de 8 o 16 bits que hay que desplazar y un segundo que contiene el número de posiciones que se desplazarán. Este valor de desplazamiento puede darse de forma inmediata sólo en el caso de que se trate del valor 1. Para valores superiores a 1 es necesario cargarlos previamente en el registro CL y especificar este registro como segundo operando de la instrucción.

• SAL / SHL

Puesto que el desplazamiento hacia la izquierda (multiplicación por potencias de 2) es idéntico tanto si se trata de cantidades con signo como sin él, las instrucciones de desplazamiento aritmético y lógico son idénticas y realizan la misma función: desplazar el operando hacia la izquierda enviando el bit superior a la bandera de acarreo y llenando el bit menos significativo con 0.

Nombre: SAL (Shift Arithmetic Left = Desplazamiento aritmético a la izquierda)

SHL (Shift Left = Desplazamiento a la izquierda)

Acción: CF<-bit superior de operando1, operando1<-2*operando1

Modos de direccionamiento: Registro, indirecto, relativo a registro, relativo a base, indexado, indexado con base
 Banderas modificadas: OF,SF,ZF,PF y CF. Además, AF queda con resultado indefinido.

Ejemplo:

```
MOV AL,00100010b ;AL=00100010b
SHL AL,1           ;AL=01000100b, CF=0
MOV CL,2           ;Para desplazar más de una posición hay
                  ;que almacenar el contador en CL
SAL AL,CL          ;AL=00010000, CF=1
```

• SAR

El desplazamiento aritmético a la derecha se utiliza para dividir números con signo entre potencias de 2. El redondeo de la división se realiza hacia menos infinito y se obtiene únicamente la parte entera. Para realizar esto, la instrucción SAR simplemente desplaza hacia la derecha el valor indicado por el primer operando, colocando el bit menos significativo que sale en la bandera de acarreo y utilizando el bit más significativo para llenar la nueva posición de la izquierda.

Nombre: SAR (Shift Arithmetic Right = Desplazamiento aritmético a la derecha)

Acción: CF<-bit inferior de operando1, operando1<-operando1/2 (división con signo)

Modos de direccionamiento: Registro, indirecto, relativo a registro, relativo a base, indexado, indexado con base.

Banderas modificadas: OF,SF,ZF,PF y CF. Además, AF queda con resultado indefinido.

Ejemplo:

```
MOV AL,00100010b ;AL=00100010b
SAR AL,1           ;AL=000100010b, CF=0
MOV AL,10000110b ;AL=10000110b
MOV CL,2           ;Para desplazar más de una posición
                  ;hay que almacenar el contador en CL
SAR AL,CL          ;AL=11100001, CF=1
```

• SHR

La instrucción SHR realiza un desplazamiento del valor indicado por el primer operando hacia la derecha, obteniendo el equivalente a una división sin signo por una potencia de 2. El bit que se pierde por la derecha se guarda en la bandera de acarreo y el bit superior es llenado con un cero.

Nombre: SHR (Shift Right = Desplazamiento a la derecha)

Acción: CF<-bit inferior de operando1, operando1<-operando1/2
(división sin signo)

Modos de direccionamiento: Registro, indirecto, relativo a registro,
relativo a base, indexado, indexado con base.

Banderas modificadas: OF,SF,ZF,PF y CF. Además, AF queda con
resultado indefinido.

Ejemplo:

MOV AL,00100010b

;AL=00100010b

SHR AL,1

;AL=000100010b, CF=0

MOV AL,10000110b

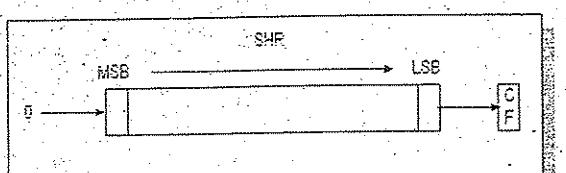
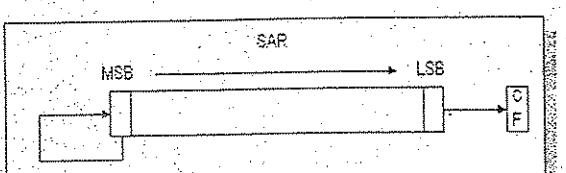
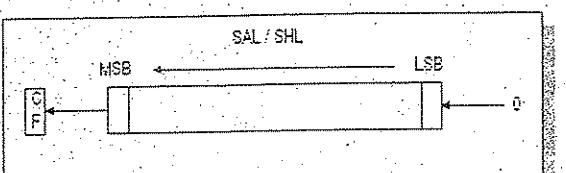
AL=10000110b

MOV CL,2

;Para desplazar más de una posición
hay que almacenar el contador en CL

SHR AL,CL

;AL=00100001, CF=1



• ROL

La instrucción ROL efectúa una rotación de bits hacia la izquierda, es decir, a medida que se desplazan los bits, aquellos que se pierden por la izquierda son introducidos de nuevo por la derecha. Además, la bandera

de acarreo recibe una copia del bit que se traslada desde la izquierda (bit más significativo) hacia la derecha (bit menos significativo).

Nombre: ROL (Rotate Left = Rotar a la izquierda)

Acción: CF<-bit superior de operando1. Operando1<-operando1*2+bit
superior de operando1

Modos de direccionamiento: Registro, indirecto, relativo a registro,
relativo a base, indexado, indexado con base.

Banderas modificadas: OF y CF

Ejemplo:

MOV AL,00100010b

;AL=00100010b

ROL AL,1

;AL=01000100b, CF=0

MOV AL,10000110b

AL=10000110b

MOV CL,2

;Para rotar más de una posición hay
que almacenar el contador en CL

ROL AL,CL

;AL=00011010b, CF=0

• ROR

ROR realiza la operación inversa a ROL, rotando los bits de la cantidad indicada por el primer operando hacia la derecha y utilizando el bit que se pierde por la derecha para llenar el hueco creado a la izquierda. Además, este bit desplazado se coloca en la bandera de acarreo.

Nombre: ROR (Rotate Right = Rotar a la derecha)

Acción: CF<-bit inferior de operando1. Operando1<-operando1/2. Bit
superior operando1<- CF

Modos de direccionamiento: Registro, indirecto, relativo a registro,
relativo a base, indexado, indexado con base.

Banderas modificadas: OF y CF

Ejemplo:

MOV AL,00100010b

;AL=00100010b

ROR AL,1

;AL=00010001b, CF=0

MOV AL,10000110b

AL=10000110b

MOV CL,2

;Para rotar más de una posición hay que
almacenar el contador en CL

ROR AL,CL

;AL=10100001b, CF=1

• RCL

La instrucción RCL realiza una rotación hacia la izquierda para utilizar el bit de acarreo como si formase parte del propio dato. De esta forma, se desplaza hacia la izquierda el dato indicado por el primer operando y el bit que sale por la izquierda es colocado en la bandera de acarreo, mientras que el anterior valor de esta bandera es utilizado para llenar el bit menos

significativo del valor que se rota.

Nombre: RCL (Rotate Carry Left = Rotar a la izquierda a través de acarreo)

Acción: CF<-bit superior de operando. Operando1<-operando1*2+valor anterior de CF

Modos de direccionamiento: Registro, indirecto, relativo a registro, relativo a base, indexado, indexado con base

Banderas modificadas: OF y CF

Ejemplo:

MOV AL,00100010b ;AL=00100010b CF=0

RCL AL,1 ;AL=01000100b, CF=0

MOV AL,10000110b ;AL=10000110b, CF=0

MOV CL,2 ;Para rotar más de una posición hay que almacenar el contador en CL

RCL AL,CL ;AL=00011001b, CF=0

• RCR

RCR realiza una rotación hacia la derecha a través del acarreo del dato indicado como primer argumento. El bit más significativo tras la rotación se completa con el contenido de la bandera de acarreo, mientras que el bit que sale por la derecha se envía a la bandera de acarreo. [ver gráfico. 2]

Nombre: RCR (Rotate Carry Right = Rotar a la derecha a través de acarreo)

Acción: CF<-bit superior de operando. Operando1<-operando1*2+valor anterior de CF

Modos de direccionamiento: Registro, indirecto, relativo a registro, relativo a base, indexado, indexado con base

Banderas modificadas: OF y CF

Ejemplo:

MOV AL,00100010b ;AL=00100010b CF=0

RCR AL,1 ;AL=00010001b, CF=0

MOV AL,10000110b ;AL=10000110b, CF=0

MOV CL,2 ;Para rotar más de una posición hay que almacenar el contador en CL

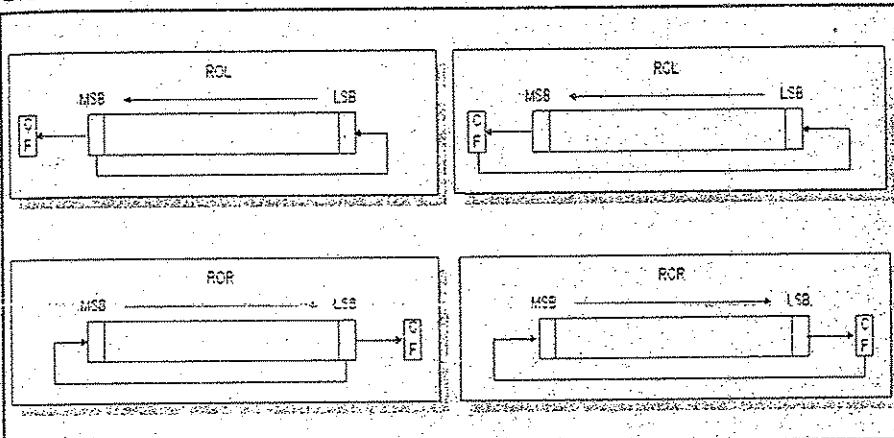
RCR AL,CL ;AL=00100001b, CF=1

DE TRANSFERENCIA DE CONTROL

Si los programas que ejecuta el ordenador fuesen únicamente secuencias lineales de instrucciones, éste sería de poca utilidad. Precisamente el gran número de posibilidades que ofrece el ordenador

se debe a su capacidad de "tomar decisiones", es decir, de decidir qué acción llevar a cabo en función de, por ejemplo, que el usuario seleccione una opción determinada de un menú o de que el resultado de una operación haya tenido un resultado positivo o negativo.

GRAFICO.2



A las instrucciones que pueden romper el flujo secuencial de instrucciones y hacer que el procesador pase a ejecutar, en lugar de la siguiente instrucción, otro grupo contenido en otra parte de la memoria, se les denomina "instrucciones de transferencia de control", y constituyen el grupo más numeroso dentro del lenguaje del procesador 8086.

La forma en que el procesador realiza la transferencia de control, es decir, el paso para ejecutar una instrucción cualquiera, no necesariamente consecutiva a la actual, es simplemente cargando en el registro IP (el puntero de instrucción) la posición de memoria en la que se encuentra la siguiente instrucción que se desea ejecutar. Aunque esto podría considerarse como una instrucción de transferencia de datos que toma como destino implícito el registro IP, se cuenta con toda una serie de formas diferentes de realizar esta acción, por lo que se justifica un nuevo grupo de instrucciones.

DISTANCIA DE SALTO

A la hora de indicar a una instrucción de transferencia de control la posición nueva en la que debe continuar la ejecución del programa, es necesario tener en cuenta la distancia en bytes existente entre la instrucción actual y la nueva. A esta distancia se le denomina "distancia de salto", existiendo únicamente tres tipos diferentes: salto lejano (FAR), salto cercano (NEAR) y salto corto (SHORT), realizándose cada uno de ellos de una forma distinta.

Además de por su longitud, las instrucciones de transferencia de control pueden clasificarse como absolutas y relativas. Las de tipo absoluto indican la posición de memoria en la que deben continuar la ejecución del programa, mientras que las relativas indican la distancia que hay entre la siguiente instrucción que se ejecutaría y la nueva que debe utilizarse, es decir, el valor que hay que sumar al registro IP para obtener la dirección de la siguiente instrucción que se debe ejecutar. Si el salto es hacia adelante, el valor indicado será un desplazamiento positivo, mientras que si es hacia atrás el desplazamiento será negativo. Hay que tener en cuenta en los saltos relativos que el desplazamiento no se da respecto a la dirección de la instrucción de salto, sino a la instrucción siguiente a la que produce la transferencia de control, por lo que un desplazamiento de 0 provoca un salto inútil, puesto que saltaría hacia la siguiente instrucción.

Aunque en principio puedan parecer más complejos los saltos de tipo relativo, son los más utilizados en la creación de programas, pues cuentan con la ventaja de que un fragmento de programa que utiliza saltos relativos puede colocarse en cualquier parte de la memoria y ejecutarse sin realizar cambios, lo cual facilita la tarea de reutilizar código de unos programas a otros. Sólo en algunas ocasiones será preciso utilizar saltos de tipo absoluto.

Los saltos de tipo corto SHORT son siempre relativos y utilizan un único byte para indicar la distancia del salto. Por tanto, este tipo de saltos sólo admite transferencias de control de 128 bytes hacia atrás o 127 hacia adelante (el rango de un byte con signo es de -128 a 127). Los de tipo cercano, por su parte, son también de tipo relativo, pero utilizan un word para indicar su desplazamiento de salto, lo que les permite transferir el flujo del programa a cualquier parte dentro de un mismo segmento de memoria de 64 Kilobytes. Aunque un salto de tipo cercano sólo puede saltar -32768 bytes hacia atrás y 32767 hacia adelante, gracias al hecho de que los segmentos del 8086 son circulares es posible acceder a todo el segmento desde cualquier otro punto.

Para realizar saltos entre diferentes segmentos es necesario utilizar el tipo de salto absoluto lejano (FAR), el cual utiliza dos words para indicar la dirección de salto, cargándose el primero de estos words en el registro IP y el segundo en el registro de segmento CS.

Más adelante, en el capítulo dedicado al lenguaje ensamblador se explicará cómo éste, al utilizar direcciones simbólicas, facilita la tarea de indicar los destinos de salto de estas instrucciones, puesto que el ensamblador se ocupa de seleccionar el tipo de salto más adecuado en cada caso y de calcular los desplazamientos necesarios.

SALTO INCONDICIONAL

• JMP

La instrucción de salto incondicional provoca que la ejecución del programa continúe a partir de una cierta dirección de memoria, la cual puede indicarse de forma relativa con saltos de tipo SHORT o NEAR o de forma absoluta mediante un tipo FAR. En los saltos de tipo SHORT el desplazamiento sólo puede indicarse como un valor inmediato, mientras que los otros dos tipos admiten tanto un valor inmediato como un direccionamiento relativo a registro o a memoria.

Nombre: MP (JUMP = salto)

Acción: SHORT, NEAR: IP<-IP+operando. FAR: IP<-[operando]

CS<-[operando+2]

Modos de direccionamiento: SHORT: inmediato, Resto: inmediato, registro, indirecto, relativo a registro, relativo a base, indexado, indexado con base.

Banderas modificadas: Ninguna

Ejemplo que incrementa el registro AX en un bucle sin fin:

MOV AX,0 ;AX=0000	
xxxx : INC AX ;AX=AX+1	
JMP xxxx ;Vuelve a ejecutar la instrucción anterior	

SALTO CONDICIONAL

Las instrucciones de salto condicional son muy numerosas. Cada una de ellas examina una o varias banderas del registro de indicadores y, si sus valores son los que esperan, producen el salto relativo indicado como argumento. En caso de que no se cumpla la condición, se continúa ejecutando la instrucción que sigue a la de salto. Por tanto, las instrucciones de salto condicional van precedidas de una instrucción que modifica las banderas (normalmente CMP), posibilitando tomar un camino u otro del programa en función de su resultado.

Por ejemplo, el siguiente código compara el registro AX con el valor 1234h y, si son iguales, salta a la posición xxxx de la memoria, desde donde seguirá ejecutando nuevas instrucciones. Para realizar esta comparación se utiliza la instrucción CMP AX,1234h que, en caso de que AX contenga dicho valor, colocará la bandera de cero a 1 (ZF=1). Esta condición puede ser chequeada a continuación mediante el salto condicional JE (salta si ZF=1, es decir, salta si igual) para cambiar el flujo del programa a la posición xxxx cuando se cumpla dicha condición.

CMP AX,1234h ;Compara el valor en AX con 1234h	
JE xxxx ;Si es igual, salta a la posición xxxx	

CÓMO PROGRAMAR EN ENSAMBLADOR

xxxx : NOP

;Código que se ejecuta si AX distinto de 1234h
;Aquí se salta si AX=1234h

Este tipo de saltos son siempre relativos y de tipo SHORT, por lo que únicamente es posible cambiar a instrucciones hasta 128 bytes hacia atrás o de 127 hacia adelante. Este pequeño margen de salto es suficiente en la mayoría de las ocasiones, pero en caso de que se necesite un salto más largo es posible hacer que la instrucción de salto condicional salte a otra instrucción de salto incondicional que lleve al destino deseado, según se muestra en el siguiente código:

CMP AX,1234h
JE xxxx

;Si se cumple la condición se salta a la posición xxxx

JMP falso

;Salta al código cuando no se cumple la condición

xxxx : JMP cierto

;Aquí se lleva a cabo el salto a la posición deseada realmente

Nombre: Jxx (Jump if xx = Salta si xx)

Acción: Si se cumple xx entonces IP<-IP+operando

Modos de direccionamiento: Inmediato.

La siguiente tabla resume todas las instrucciones de salto condicional, las banderas que chequean, así como para qué comprobación es útil cada una de ellas. [ver gráfico. 3]

En la tabla puede observarse cómo algunas instrucciones cuentan con dos mnemónicos diferentes, pues en realidad ambas comprueban la misma condición, pero con distintas palabras. Esto está pensado para que los programadores utilicen la forma que resulte más descriptiva para el programa. Por ejemplo, si lo que se intenta comprobar es si un número es menor que otro, se utilizaría una instrucción CMP de ambos números y a continuación una instrucción JB. Si lo que se desease fuese simplemente tomar una decisión en función del estado de la bandera de acarreo (algunas funciones de la BIOS y del MS-DOS indican que se produjo un error colocando la bandera de acarreo a 1), entonces se debería utilizar la instrucción JC.

Un caso curioso dentro de las instrucciones de salto condicional lo supone JCXZ, que no chequea banderas, sino el registro del procesador CX, saltando en caso de que contenga el valor cero. El motivo de esta instrucción se debe a que, como se verá, el registro CX se utiliza con las instrucciones de repetición como contador, por lo que esta instrucción permite chequear en caso de que el bucle se ejecute cero veces

Gráfico. 3]

Mnemónico	Significado	Banderas	Salta si
JA	Jump if Above	CF=0 y ZF=0	Primer número superior que el segundo (números sin signo)
JNBE	Jump if Not Below or Equal		Primer número no inferior o igual que el segundo (números sin signo)
JAE	Jump if Above or Equal	CF=0	Primer número superior o igual que el segundo (números sin signo)
JNB	Jump if Not Below		Primer número no inferior que el segundo (números sin signo)
JB	Jump if Below	CF=1	Primer número inferior que el segundo (números sin signo)
JNAE	Jump if Not Above or Equal		Primer número no superior o igual que el segundo (números sin signo)
JBE	Jump if Below or Equal	CF=1 o ZF=1	Primer número inferior o igual que el segundo (números sin signo)
JNA	Jump if Not Above		Primer número no superior que el segundo (números sin signo)
JC	Jump if Carry	CF=1	Salta si acarreo
JE	Jump if Equal	ZF=1	Salta si primer número igual al segundo
JZ	Jump if Zero		Salta si cero
JNC	Jump if Not Carry	CF=0	Salta si no acarreo
JNE	Jump if Now Equal	ZF=0	Salta si primer número distinto de segundo
JNZ	Jump if Not Zero		Salta si no cero
JNP	Jump if Not Parity	PF=0	Salta si no paridad
JPO	Jump if Parity Odd	ZF=0 y SF=OF	Salta si paridad impar
JP	Jump if Parity	PF=1	Salta si paridad
JPE	Jump if Parity Even		Salta si paridad par
JG	Jump if Greater	ZF=0 y SF=OF	Primer número mayor que el segundo (números con signo)
JNLE	Jump if Not Lower or Equal		Primer número no menor o igual que el segundo (números con signo)
JGE	Jump if Greater or Equal	SF=OF	Primer número mayor o igual que el segundo (números con signo)
JNL	Jump if Not Lower		Primer número no menor que el segundo (números con signo)
JL	Jump if Lower	SF < OF	Primer número menor que el segundo (números con signo)
JNGE	Jump if Not Greater or Equal		Primer número no mayor o igual que el segundo (números con signo)
JLE	Jump if Lower or Equal	ZF=1 y SF < OF	Primer número menor o igual que el segundo (números con signo)
JNG	Jump if Not Greater		Primer número no mayor que el segundo (números con signo)
JNO	Jump if Not Overflow	OF=0	Salta si no desbordamiento
JNS	Jump if Not Sign	SF=0	Salta si no signo
JO	Jump if Overflow	OF=1	Salta si desbordamiento
JS	Jump if Sign	SF=1	Salta si signo
JCXZ	Jump if CX Zero	CX=0	Salta si CX contiene el valor 0

(ninguna vez) y posibilitar la omisión del código completo del bucle.

LLAMADA A SUBRUTINAS

Las subrutinas no son más que fragmentos de un programa que realizan una tarea concreta y que pueden llamarse desde varios puntos de un mismo programa. Las ventajas de utilizar subrutinas son dos fundamentalmente. La primera de ellas es que evitan tener que repetir un fragmento de código muy utilizado en un programa, colocándolo como una subrutina y llamándolo desde tantos puntos del programa como sea necesario. La otra gran ventaja, más importante incluso que la primera, es que permite reducir el programa a un conjunto de módulos que realizan una parte del trabajo del programa, facilitando así su comprensión y la localización de fallos. Estos módulos, además, pueden servir en la realización de otros programas totalmente distintos. Si por ejemplo el programa que se está creando necesita imprimir un número en pantalla, se podría realizar una subrutina que recibiese el número en el registro AX y lo mostrase en la pantalla para que el usuario lo pudiese ver. Si esta subrutina se utilizase en varios puntos del programa, bastaría con colocar el valor que se desea imprimir en cada caso en el registro AX y llamar a la subrutina correspondiente. Más aún, aunque sólo se realizase la impresión del valor una sola vez durante el programa, convendría ponerlo como una subrutina, pues de esa forma, en caso de que el programa no mostrase en pantalla el valor de forma correcta, se sabría que el fallo está en dicha subrutina. Además, esta subrutina se podría coger y, sin cambios, utilizar en otro programa diferente.

Para realizar su función, la subrutina recibe unos datos de entrada denominados parámetros, los utiliza para realizar su trabajo y, por último, devuelve unos resultados de salida. La transferencia de esta información puede realizarse de tres formas diferentes: a través de registros, por posiciones fijas de memoria o a través de la pila. El sistema más rápido es el primero, a través de registros, pues evita el acceso a la memoria del sistema, por lo que es el que se suele utilizar principalmente en la programación en ensamblador. En algunos casos especiales puede utilizarse alguno de los otros dos métodos, como por ejemplo si no se cuenta con registros suficientes para pasar todos los datos o no se desea modificar su contenido. El último de estos métodos, el paso de parámetros en la pila, es el utilizado por los lenguajes de programación de alto nivel (C, Pascal, etc...).

El funcionamiento de las instrucciones de transferencia de control a subrutinas se basa en dos instrucciones. Cuando el procesador encuentra la primera de ellas (CALL) guarda la dirección de la siguiente instrucción al CALL en la pila, y transfiere la ejecución del programa a la

posición de memoria indicada como argumento de la instrucción CALL. Dentro de la subrutina se ejecutan sus instrucciones hasta que se encuentra la instrucción RET, en cuyo punto el procesador extrae de la pila la dirección previamente almacenada y salta a ella, por lo que la ejecución del programa continúa desde la siguiente instrucción al CALL que llamó a la subrutina.

Puesto que la dirección de retorno del CALL se guarda en la pila, es necesario que antes de la instrucción RET la pila se encuentre exactamente en el mismo estado en que se encontraba al comenzar la subrutina. En caso de que se hayan guardado datos en la pila que no se hayan retirado posteriormente, la instrucción RET recuperará una dirección de retorno inválida, volverá a una dirección errónea que, seguramente, provocará el fatídico "cuelgue" del ordenador.

• CALL

La instrucción CALL guarda la dirección de la instrucción que la sigue en la pila y transfiere el control a la posición de programa indicada por su único argumento. Esta dirección de salto puede ser un valor inmediato o una referencia a la posición de memoria donde se encuentra dicho valor. Si el salto a subrutina es de tipo FAR, entonces se guarda en la pila tanto el valor de los registros CS (en primer lugar) como el de IP. Si es de tipo NEAR, sólo se guarda el registro IP en la pila.

Nombre: CALL (CALL = llamada)

Acción: NEAR: PUSH IP, JMP argumento; FAR: PUSH CS, PUSH IP, JMP argumento

Modos de direccionamiento: Inmediato, indirecto, relativo a registro, relativo a base, indexado, indexado con base.

Banderas modificadas: Ninguna.

• RET

La instrucción RET marca el final de una subrutina y continúa la ejecución del programa a partir de la siguiente instrucción al CALL que la llamó. Hay dos variantes de la instrucción RET, una FAR y otra NEAR, debiéndose utilizar en cada caso el mismo tipo que el que utiliza la instrucción CALL que realiza la llamada. Es decir, si la subrutina utiliza un RET NEAR, entonces siempre debe ser llamada con un CALL NEAR, mientras que si utiliza un RET FAR deberá ser llamada siempre con un CALL FAR. Para marcar esta diferencia algunos ensambladores utilizan dos instrucciones auxiliares: RETN para expresar un RET de tipo NEAR y RETF para indicar un retorno de tipo FAR.

La instrucción RET puede ir seguida de un valor numérico inmediato, que indica el número de words que deben eliminarse de la pila tras retornar

de la subrutina. Esto posibilita el eliminar de la pila los datos de entrada de la subrutina en el caso de que se haya utilizado este método para el paso de parámetros.

Nombre: RET (RETurn = Retorna)

Acción: NEAR: POP IP, SP<-SP+parámetro. FAR: POP IP, POP CS, SP<-SP+parámetro

Modos de direccionamiento: Inmediato

Banderas modificadas: Ninguna

Ejemplo: Suma los valores del registro AX y BX y devuelve el resultado en AX

xxxx ADD AX,BX

RET

MOV AX,1234h ;AX=1234h

MOV BX,4444h ;BX=4444h

CALL xxxx ;AX=1234h+4444h=5678h

MOV CX,xxxx ;Carga en CX la dirección de la subrutina

CALL [CX] ;Llamada a la subrutina de forma indirecta

;AX=5678h+4444h=9ABCh

DE BUCLES

Los bucles, fragmentos de código que se repiten varias veces, aparecen con frecuencia dentro de los programas, generalmente para realizar cálculos repetitivos sobre un conjunto de datos. El 8086 ofrece varias instrucciones para este propósito.

• LOOP

La instrucción LOOP utiliza el registro CX como un contador que contiene el número de veces que ha de repetirse el bucle. Cada vez que la ejecución del programa llega a la instrucción LOOP se decrementa el valor del registro CX en una unidad y, si el contenido de este registro es distinto de cero, se hace un salto SHORT a la dirección inmediata indicada como parámetro de la instrucción LOOP. Hay que tener en cuenta que si el valor de CX es inicialmente 0, entonces el bucle se ejecuta 65536 veces. En caso de que se desee que si CX vale 0 no se ejecute el bucle, será necesario colocar una instrucción JCXZ precediendo todo el cuerpo del bucle.

Nombre: LOOP (LOOP=bucle)

Acción: CX<-CX-1, si CX distinto de 0 entonces IP<-IP+operando

Modos de direccionamiento: Inmediato

Banderas modificadas: Ninguna

Ejemplo: Suma las cifras desde 1 hasta una dada

MOV CX,0010h

;Carga un valor de ejemplo en CX (16 en este caso)

XOR AX,AX

;Inicializa variable de suma

JCXZ yyyy

;Si CX=0 se salta todo el bucle

xxxx ADD AX,CX

;Suma el valor del contador a AX

LOOP xxxx

;Cierra el bucle

yyyy NOP

;AX=0088h = 136 = 16+15+14...+3+2+1

• LOOPE / LOOPZ

La variante LOOPE es idéntica a la instrucción LOOP, sólo que se salta a la posición indicada por el primer argumento, es decir, se cierra el bucle sólo si el valor del registro CX es distinto de cero y la bandera de cero tiene el valor 1. Esta instrucción es útil, por tanto, para crear bucles controlados por la bandera de cero, pero en los que se especifica el número máximo de veces que puede ejecutarse en el registro CX. Este valor máximo evita que el programa se quede encerrado en un bucle sin fin en caso de que la bandera de cero nunca se ponga a cero.

Si tras la instrucción LOOPE se desea averiguar si el bucle finalizó porque el registro CX llegó a cero o porque la bandera de cero se puso a 0, deberán utilizarse las instrucciones de salto condicional JE, JZ, JNE, JNZ o JCXZ para diferenciar un caso del otro.

Nombre: LOOPE (LOOP while Equal = bucle mientras igual)

LOOPZ (LOOP while Zero = bucle mientras cero)

Acción: CX<-CX-1. Si CX distinto de 0 y ZF=1 entonces IP<-IP+operando
Banderas modificadas: Ninguna

Ejemplo: Espera a que por el puerto 80h se reciba un valor distinto de 0:

MOV CX,100h ;Guarda en CX el número máximo de comprobaciones

xxxx IN AL,[80h] ;Recibe un valor desde el puerto 80h

CMP AL,0 ;Lo compara con 0

LOOPE xxxx ;Si es igual, repite el bucle

JNE yyyy ;Salta si el bucle se acabó porque ZF=0

NOP ;Sigue por aquí si el bucle acabó porque CX=0

• LOOPNE / LOOPNZ

Si la condición adicional que se debe cumplir para repetir el bucle es que la bandera de cero debe tener el valor 0, entonces se utilizará la variante LOOPNE o su nombre alternativo, LOOPNZ.

Nombre: LOOPNE (LOOP while Not Equal = Bucle mientras no igual)

LOOPNZ (LOOP while Not Zero = Bucle mientras no cero)

Acción: CX<-CX-1. Si CX distinto de 0 y ZF=0 entonces

IP<-IP+operando

Banderas modificadas: Ninguna.

Ejemplo: Espera que se reciba un valor determinado por un puerto:

xxxx	MOV CX,100h	;Como máximo 100h comprobaciones
	IN AL,[80h]	;Lee el valor del puerto
	CMP AL,10h	;Lo compara con el valor esperado (10h)
	LOOPNE xxxx	;Si es diferente y CX no llegó a cero, repite el bucle]
	CXZ yyyy	;Salta si el bucle acabó porque CX llegó a cero
	NOP	;Sigue aquí si el bucle acabó porque se leyó el valor 80h del puerto

• REP

El bucle más pequeño es aquel que contiene una única instrucción. Este tipo de bucle lo lleva a cabo REP, que no es una instrucción, sino un prefijo que repite la instrucción que va a continuación tantas veces como indique el registro CX. Este prefijo se utiliza de forma casi exclusiva con las instrucciones de manejo de cadenas que se tratarán en el próximo apartado. La única aplicación alternativa de este prefijo es su utilización con la instrucción NOP para crear pequeños bucles de retardo dentro del código.

Al contrario de lo que sucedía con la instrucción LOOP, si inicialmente el registro CX contiene el valor 0, entonces la instrucción que sigue al prefijo REP no se ejecuta ninguna vez, evitando así el chequear este caso especial.

Nombre: REP (REPeat= Repite)

Acción: Mientras CX distinto de 0 (ejecuta siguiente instrucción y CX<-CX-1)

Banderas modificadas: Ninguna

• REPE/REPZ

La instrucción REPE (y su equivalente REPZ) es una variante de REP que para repetir la ejecución de la siguiente instrucción comprueba no sólo que CX sea distinto de cero, sino que la bandera de cero tenga el valor 1. Esta forma de REP se utiliza para crear bucles que deben ejecutarse mientras la bandera de cero tenga el valor 1 pero se debe especificar un número máximo de iteraciones para evitar que se caiga en un bucle sin fin.

Nombre: REPE (REPeat while Equal = Repite mientras igual)

REPZ (REPeat while Zero = Repite mientras cero)

Acción: Mientras CX distinto de 0 y ZF=1 (ejecuta siguiente instrucción y CX<-CX-1)

Banderas modificadas: Ninguna.

• REPNE / REPNZ

Si el bucle REP debe mantenerse mientras que la bandera de cero tenga el valor 0, se utilizará la instrucción REPNE o REPNZ, que chequean dicha condición.

Nombre: REPNE (REPeat while Not Equal = Repite mientras no igual)

REPZ (REPeat while Not Zero = Repite mientras no cero)

Acción: Mientras CX distinto de 0 y ZF=0 (ejecuta siguiente instrucción y CX<-CX-1)

Banderas modificadas: Ninguna.

DE MANEJO DE CADENAS

Las instrucciones de manejo de cadenas facilitan el trabajo con conjuntos de datos consecutivos en memoria. Para su funcionamiento no necesitan ningún parámetro, pues en caso de que precisen algún dato, toman el valor colocado en la posición de memoria DS:[SI], y si necesitan almacenar algún valor lo hacen en la posición apuntada por ES:[DI]. Es importante recordar que el registro de segmento DS va asociado con el registro índice fuente SI y que el registro ES se asocia con el índice destino DI. Tras la lectura de un dato la propia instrucción incrementa el registro SI para que apunte hacia el siguiente elemento de la cadena. Análogamente, si una instrucción de cadena almacena un dato, incrementará el valor del registro DI para que apunte hacia el lugar donde se guardará el siguiente elemento.

El incremento que se aplica a los registros índice depende del tamaño de los datos con los que se trabaje. Si, por ejemplo, las instrucciones de cadena están manejando bytes, los registros se incrementarán en una unidad, mientras que si se trabaja con words el incremento será de dos unidades. Es decir, tras una instrucción de cadena los registros índice siempre quedarán apuntando al siguiente dato a procesar, independientemente del tamaño de datos (byte o word) con los que se trabaje. La forma de indicar si la instrucción maneja bytes o words es añadiendo la letra B al final de la instrucción, para el trabajo con bytes, o la letra W, para especificar trabajo con words. Según esto, si se desea que, por ejemplo, la instrucción LODS trabaje con datos de tipo byte, se escribiría LODSB, y si se quisiese que manejase words se utilizaría la forma LODSW.

Puesto que las instrucciones de cadena reúnen el proceso de datos con la actualización de los índices de origen y destino de los datos, a menudo

Banderas modificadas: OF, SF, ZF, AF, PF y CF.

Ejemplo: Compara los diez word a partir de 1234 con los contenidos en 5678h, comprobando si son iguales:

PUSH DS	;Coloca ES apuntando el mismo segmento que DS
POP ES	
MOV SI,1234h	;Inicializa los registros índice
MOV DI,5678h	
MOV CX,10	;Comparar 10 words
REPE CMPSW	;Realiza la comparación
JE xxxx	;Salta si eran iguales
NOP	;Llega aquí si eran diferentes. Además SI y DI apuntan al siguiente word al que difería

DE CONTROL

Existe un pequeño conjunto de instrucciones, denominadas de control, cuya finalidad es la de sincronizar el 8086 con otros procesadores o dispositivos del sistema. Se trata de tres instrucciones que raramente son utilizadas por los programas, quedando reservadas para propósitos muy específicos.

- ESC

La instrucción ESC permite la comunicación del 8086 con un procesador auxiliar, como es el caso de un coprocesador matemático o uno de entrada/salida. Esta instrucción lleva dos parámetros. El primero es un valor inmediato de 0 a 63, que especifica tanto la operación a realizar como el coprocesador que la llevará a cabo. El segundo operando es una referencia a una posición de memoria cuyo contenido se colocará en el bus de datos para que el otro procesador pueda recogerlo.

Nombre: ESC (ESCAPE = escape)

Acción: bus de datos <operando2>

Modos de direccionamiento: Indirecto, relativo a registro, relativo a base, indexado, indexado con base

Banderas modificadas: Ninguna

Ejemplo:

ESC 21,[1234h]

- HLT

La instrucción de control HLT detiene al procesador hasta que se produzca una interrupción externa. Se utiliza raramente para esperar por una comunicación de un dispositivo externo.

Nombre: HLT (HalT = parar)

Acción: Espera a que haya una interrupción

Modos de direccionamiento: Ninguno

Banderas modificadas: Ninguna

- WAIT

Espera hasta que el coprocesador auxiliar se encuentre desocupado. Se utiliza precediendo a las instrucciones ESC para asegurar que el coprocesador se encuentra preparado para recibir nuevos datos.

Nombre: WAIT (WAIT= esperar)

Acción: Espera a que el coprocesador esté libre

Modos de direccionamiento: Ninguno

Banderas modificadas: Ninguna

Ejemplo:

WAIT
ESC 21,[1234h]

DE INTERRUPCIÓN

Las interrupciones son un mecanismo por el que un dispositivo externo puede provocar que el procesador interrumpa momentáneamente la ejecución del programa para atender a su petición, para luego continuar con el programa desde el punto en que lo había dejado. Pero el procesador 8086 amplía este concepto y permite que cualquier programa pueda generar una interrupción en cualquier momento. A este tipo de interrupciones se las conoce como "interrupciones software" para diferenciarlas de las "interrupciones hardware", generadas por dispositivos externos al procesador.

En un PC existen un total de 256 interrupciones software, de las cuales sólo 8 pueden ser ocupadas por interrupciones hardware (o 16 en máquinas con procesador 80286 o superior). Cada una de estas interrupciones hardware ocupa una interrupción software. La siguiente tabla muestra esta correspondencia, así como qué interrupciones utilizan los dispositivos del PC. [ver tabla 3, en página siguiente]

Esta tabla no contiene todos los dispositivos que pueden utilizar interrupciones, pues la lista sería innumerable. Dado el escaso número de interrupciones hardware, en ocasiones varios dispositivos deben compartir un mismo número de interrupción, a ser posible uno en el que el otro dispositivo que comparte no se encuentre activo. Este es el caso, por ejemplo, de muchas tarjetas de sonido que ocupan la interrupción hardware 5, pues muy pocas personas disponen de dos puertos de impresora en su equipo.

Un tipo especial de interrupciones hardware lo constituyen las denominadas "interrupciones internas", que son interrupciones generadas por el propio procesador para indicar situaciones especiales como una división por cero o ayudar a la depuración del programa. En total hay cinco interrupciones, que corresponden a las primeras cinco interrupciones del sistema (0 a la 4), y cuya finalidad se resume en la siguiente tabla: [ver tabla.4]

Aunque la forma en que se llevan a cabo las interrupciones pueda parecer casi mágica, en realidad es un proceso muy simple. Cuando se produce una interrupción, bien hardware o bien software, el procesador

TABLA.3

Int. Hardware	Int. Software
0	8
1	9
2	A
3	B
4	C
5	D
6	E
7	F
8	70
9	71
A	72
B	73
C	4
D	75
E	76
F	77

TABLA4

Interrupción	Nombre
0	División por cero
1	Paso a paso
2	No enmascarable
3	Punto de ruptura
4	Desbordamiento

obtiene en primer lugar el número de interrupción software, lo multiplica por cuatro y utiliza ese valor como desplazamiento en una tabla de punteros FAR (de ahí que se multiplique por 4) situada en el comienzo de la memoria RAM (segmento 0), que contiene la dirección de la subrutina que procesará la interrupción. Tras conocer la dirección de salto, guarda en la pila el registro de banderas y la dirección de retorno en la que continuará el programa tras la interrupción. Tras ello salta a la dirección donde se encuentra la rutina de proceso de la interrupción. Una vez completada la rutina de interrupción, el procesador saca de la pila el lugar donde abandonó la ejecución del programa, así como las banderas, con lo que el programa puede continuar como si no hubiese sucedido nada.

El propósito de las interrupciones hardware está claro: atender a los

Ocupada por (8086)	Ocupada por 0286 y superiores
Temporizador (timer)	Temporizador (timer).
Teclado	Teclado.
Disco duro	Cascada.
Segundo puerto serie (COM2)	Segundo puerto serie (COM2)
Primer puerto serie (COM1)	Primer puerto serie (COM1)
Impresora (LPT2)	Impresora (LPT2).
Controladora de disquete	Controladora de disquete.
Impresora (LPT1)	Impresora (LPT1).
	Reloj de tiempo real.
	No utilizada (enlace con IRQ2).
	No utilizada.
	No utilizada.
	No utilizada.
	Coprocesador matemático.
	Controladora IDE primaria.
	Controladora IDE secundaria.

Función
Se produce cuando las instrucciones DIV o IDIV encuentran que el resultado no cabe en el destino.
Si la bandera TF vale 1, tras cada instrucción el procesador genera esta interrupción para ayudar a los programas de depuración.
Esta interrupción no puede ignorarse, ni siquiera si IF=0. Se utiliza para indicar errores de memoria.
Se genera con la instrucción INT3, y es utilizada por programas de depuración para detener la ejecución del programa.
Se genera con la instrucción INTO si OF=1.

periféricos tan pronto como sea posible. Pero ¿para qué sirven las interrupciones software?. En el PC, estas interrupciones se utilizan para llamar a ciertas rutinas de propósito general, de forma que se encuentren disponibles de forma permanente para todos los programas. Otra ventaja adicional es que, al encontrarse definida la dirección de la rutina por un vector contenido en memoria, es posible cambiar estas rutinas simplemente haciendo que el vector de interrupción apunte a la nueva rutina, todo ello sin necesidad de cambiar el programa.

Este sistema de mantener accesibles rutinas útiles a través de interrupciones es utilizado por la ROM BIOS del PC y por el sistema operativo MS-DOS; entre otros muchos programas, y es la razón que posibilita, por ejemplo, que un programa creado cuando los PC'S sólo disponían de unidades de disco para almacenar sus datos funcione hoy en día en un disco duro o una unidad de red, simplemente instalando una versión más moderna del sistema operativo que coloque unas rutinas de acceso a disco actualizadas en la interrupción correspondiente.

- INT

Las interrupciones software pueden generarse mediante la instrucción INT, que dispone de un único parámetro entre 0 y 255 que indica el número de interrupción. Su funcionamiento consiste en guardar en la pila el registro de banderas, el segmento de código (CS) y el puntero de instrucción (IP). Tras ello, desactiva las interrupciones y la ejecución pasa a paso y salta a la dirección indicada por el vector de interrupción correspondiente.

Nombre: INT (INTerupt = Interrupción)

Modos de direccionamiento: Inmediato

Acción: PUSHF, PUSH CS, PUSH IP, IF<-0, TF<-0, IP<-[0:operando*4], CS<-[0:operando*4+2]

Banderas modificadas: Ninguna (IF y TF se ponen a cero durante la ejecución del código de la interrupción. Al retornar vuelven a tener sus valores originales)

Ejemplo:

INT 10h

- INT3

La interrupción 3 es utilizada por el procesador como apoyo a los programas de depuración para colocar puntos de ruptura dentro del programa. Por ello, el 8086 provee de una instrucción específica para generar dicha interrupción, cuya ventaja es que sólo ocupa un byte, lo que permite colocarla sobre cualquier otra instrucción de programa sin peligro.

Nombre: INT3 (INTerrupt 3 = interrupción 3)

Modos de direccionamiento: Ninguno.

Acción: PUSHF, PUSH CS, PUSH IP, IF<-0, TF<-0, IP<-[0:0Ch], CS<-[0:0Eh]

Banderas modificadas: Ninguna (IF y TF se ponen a cero durante la ejecución del código de la interrupción. Al retornar vuelven a tener sus valores originales).

Ejemplo:

INT3

- INTO

Cuando el procesador encuentra la instrucción INTO, comprueba la bandera de desbordamiento (OF) y, en DE caso que valga uno, provoca una interrupción 4.

Nombre: INTO (INTerrupt If Overflow = interrupción si desbordamiento)

Modos de direccionamiento: Ninguno

Acción: Si OF=1 entonces PUSHF, PUSH CS, PUSH IP, IF<-0, TF<-0, IP<-[0:10h], CS<-[0:12h]

Banderas modificadas: Ninguna (IF y TF se ponen a cero durante la ejecución del código de la interrupción. Al retornar vuelven a tener sus valores originales).

Ejemplo:

ADD AX,BX ;Realiza una suma

INTO ;INT4 si el resultado de la suma no cabe en AX

- IRET

Cuando en el código de una interrupción se encuentra la instrucción IRET, se da ésta por finalizada y se retorna al punto del programa donde se produjo la interrupción como si no hubiese sucedido nada. Para hacerlo, esta instrucción extrae de la pila la dirección de retorno FAR, colocándola en los registros IP y CS. Por último, saca de la pila también el registro de banderas almacenado por la instrucción INT.

Nombre: IRET (Interupt RETurn = Retorno de interrupción)

Acción: POP IP, POP CS, POPF

Banderas modificadas: Todas.

DE BANDERAS

El último grupo de instrucciones, las de banderas, modifican el valor de algunas de las banderas contenidas en este registro de estado del procesador. En algunas ocasiones, estas instrucciones se utilizan en

subrutinas para devolver algún resultado adicional. Por ejemplo, la bandera de acarreo se suele devolver con el valor 1 para indicar que se produjo un error. El código que realizó la llamada puede entonces utilizar la instrucción JC para realizar algún tipo de proceso especial debido al error.

DE ACARREO (STC, CLC Y CMC)

La bandera de acarreo puede colocarse al valor 1 con la instrucción STC, al valor 0 mediante CLC o invertir su valor a través de la instrucción CMC.

Nombre: STC (Set Carry = activar acarreo)

Acción: CF<-1

Modo de direccionamiento: Implícito

Banderas modificadas: CF

Nombre: CLC (CLear Carry = borrar acarreo)

Acción: CF<-0

Modo de direccionamiento: Implícito

Banderas modificadas: CF

Nombre: CMC (CoMplement Carry = complementar acarreo)

Acción: Si CF=0 entonces CF=1 si no CF=0

Modo de direccionamiento: Implícito

Banderas modificadas: CF

DE DIRECCIÓN (STD, CLD)

La bandera de dirección controla si las instrucciones de manejo de cadenas incrementan (DF=0) o decrementan (DF=1) los registros puntero tras su ejecución. Para cambiar el valor de esta bandera se utilizan las instrucciones STD y CLD, que colocan el valor 1 y 0 respectivamente en dicha bandera.

Nombre: STD (SeT Direction = Activar dirección)

Acción: DF<-1

Modo de direccionamiento: Implícito

Banderas modificadas: DF

Nombre: CLD (Clear Direction = Borrar dirección)

Acción: DF<-0

Modo de direccionamiento: Implícito

Banderas modificadas: DF

Ejemplo: Suma un número de 4 words contenido en 1234h con otro contenido en 5678h:

MOV DI,1234h	;DI apunta al destino
MOV SI,5678h	;SI apunta al origen
MOV CX,4	;sumar 4 words
CLC	;acarreo=0
CLD	
xxxx: LODSW	;carga word
ADC [DI],AX	;suma
ADD DI,2	;actualiza puntero destino
LOOP xxxx	

Obsérvese la utilización de la instrucción LODSW para evitar el tener que incrementar el registro SI de forma manual, y que se ha utilizado la instrucción CLC para colocar el acarreo inicialmente a 0 y que la primera vez que se llegue a la instrucción ADC se comporte como una ADD. Además, se utiliza CLD para indicar que la instrucción LODSW incrementa el puntero SI en cada paso.

DE INTERRUPCIÓN (STI, CLI)

El valor de la bandera de interrupción puede controlarse con las instrucciones STI (poner a 1) y CLI (poner a 0). Con ellas es posible desactivar las interrupciones hardware para proteger fragmentos críticos del programa y restaurarlas una vez finalizados.

Nombre: STI (SeT Interrupts = Activar interrupciones)

Acción: IF<-1

Modo de direccionamiento: Implícito

Banderas modificadas: IF

Nombre: CLI (CLear Interrupts = Desactivar interrupciones)

Acción: IF<0

Modo de direccionamiento: Implícito

Banderas modificadas: IF

Ejemplo: Cuando se inicializa la posición de la pila hay que cambiar tanto el registro de segmento de pila como el de puntero. Conviene, por tanto, cortar las interrupciones durante este proceso para evitar que se produzca una interrupción cuando sólo se ha cambiado uno de estos registros, pues la pila se encuentra a medio definir y su utilización provocaría el bloqueo del ordenador.

CLI ;corta las interrupciones

MOV AX,CS

MOV SS,AX ;Coloca la pila al final del segmento de código

MOV SP,FFFEh ;porque la pila crece hacia abajo

STI ;y reactiva las interrupciones

PREFIJOS

Los prefijos son unas instrucciones muy especiales que no realizan una acción por sí mismas, sino que lo que hacen es modificar la forma de actuar de la siguiente instrucción. Ya se ha explicado el funcionamiento de REP, el prefijo más complejo, por lo que sólo queda hablar de los prefijos de segmento y del de bloqueo.

• **LOCK**

El prefijo LOCK hace que el procesador active la línea de control LOCK#, bloqueando el acceso de otros dispositivos a la memoria del sistema durante la ejecución de la siguiente instrucción.

Nombre: LOCK (LOCK = bloquear)

Acción: línea LOCK# activa durante siguiente instrucción

Modos de direccionamiento: Ninguno

Ejemplo:

LOCK XCHG AX,[1234h]

SEGMENTO

Los prefijos de segmento modifican el valor del segmento por defecto que se utiliza para leer datos contenidos en memoria. Existen cuatro prefijos, uno por cada registro de segmento, que reciben los nombres CS, DS, ES y SS. En la escritura del código no se colocan delante de la instrucción, sino delante del direccionamiento que modifican.

Ejemplo:

MOV AX,CS :[1234h] ;defecto : DS, lee de CS

MOV byte ptr DS :[BP+1234h],12 ;defecto :SS, guarda en DS

El programa DEBUG

Tras los fundamentos teóricos de la programación en ensamblador, es hora de pasar a la práctica. Un programa muy útil para ello, y del que disponen todos los usuarios de PC, es DEBUG, pues viene incluido en todas las versiones de MS-DOS. Aunque no se trata de un programa muy completo, resulta útil para experimentar con las diferentes instrucciones de ensamblador y ver cómo se comportan. Tenga en cuenta que DEBUG trabaja sólo con valores hexadecimales, por lo que, aunque no vayan seguidas por la letra h, están siempre en esta base de numeración.

Para iniciar el programa debug teclee su nombre desde la línea de comandos del MS-DOS y pulse ENTER:

C:\> DEBUG

tras ello aparecerá el prompt de debug, un signo menos (-) invitándonos a introducir las órdenes que debe ejecutar. En este punto teclee la letra r y pulse ENTER, con lo que aparecerán unas líneas similares a las siguientes:

```
-r
AX=0000 BX=0000 CX=0000 DX=0000 SP=FFEE BP=0000 SI=0000
DI=0000
DS=0B82 ES=0B82 SS=0B82 CS=0B82 IP=0100 NV UP EI PL NZ NA
PO NC
0B82:0100 BC3412 MOV SP,1234
```

En ellas pueden verse los registros del procesador con sus valores. Las banderas se muestran al final de la segunda línea como parejas de letras que indican su estado. La tercera línea muestra la siguiente instrucción que se ejecutará, indicando primero el segmento (0B82) y desplazamiento (0100) donde se encuentra en memoria. A continuación aparece la representación binaria de la instrucción, compuesta del código de operación correspondiente a un MOV al registro SP de un valor inmediato (BC) y del valor inmediato en el formato invertido utilizado por intel (34 12). Por último, aparece la instrucción desensamblada tal y como la conocemos.

Pasemos ahora a ensamblar un pequeño programa. Primero se teclea el comando a seguido de la dirección de memoria a partir de la que se quiere comenzar a escribir el código. Una buena elección es comenzar en la posición 100 (256 decimal). Esto es debido a que los primeros 256 bytes los utiliza el MS-DOS para guardar cierta información y no deben modificarse. Por otra parte, al final del segmento actual se encuentra la pila, por lo que cuanto más abajo se comience, más espacio quedará para la pila. Por tanto, tecleamos a 100:

- a 100

Tras este comando, debug comienza la inserción de código, mostrando a la izquierda la dirección donde se ensamblará la instrucción. Tecleamos, pues, las siguientes líneas de un programa que cuenta el número de caracteres de una cadena acabada en un byte cuyo valor es cero. Al comienzo del programa se supone que el registro DI apunta al inicio de la cadena situada en el segmento apuntado por ES, y al final el registro CX contendrá la longitud de la cadena incluyendo el byte de finalización.

```
-a 100
0B82:0100 mov cx,ffff
0B82:0103 xor al,al
0B82:0105 repne scasb
0B82:0107 inc cx
0B82:0108 neg cx
0B82:010A
```

Una línea en blanco marca la finalización del modo de ensamblado de instrucciones. Es posible comprobar el código que se ha introducido mediante el comando u (desensamblar) seguido de la dirección desde la que se comenzará el desensamblado:

```
-u 100
0B82:0100 B9FFFF MOV CX,FFFF
0B82:0103 30C0 XOR AL,AL
0B82:0105 F2 REPNE
0B82:0106 AE SCASB
0B82:0107 41 INC CX
0B82:0108 F7D9 NEG CX
```

Obsérvese que en las instrucciones que poseen varios mnemónicos, como es el caso de REPNE, debug escoge la representación más genérica siempre (REPNZ). En caso de que hubiese algún error en el programa, sería posible corregirlo mediante el comando a seguido de la dirección de la instrucción que se desea cambiar.

El siguiente paso es colocar una cadena de caracteres de prueba. Como ejemplo se ha escogido la dirección 130h, pero podría ser cualquier otra. Para hacer esto se utiliza el comando e seguido de la dirección donde se guardarán los datos, y a continuación los datos. Es posible indicar tanto datos de tipo byte como de cadena. Los datos de cadena deben ir encerrados entre comillas, y pueden mezclarse datos de ambos tipos siempre que se separen por un espacio en blanco.

```
-e 130 'prueba' 00
```

Para comprobar que la cadena se ha guardado correctamente en memoria se utiliza el comando d (volcado):

```
-d 130
0B82:0130 70 72 75 65 62 61 00 77-E9 52 50 53 51 06 57 8B
prueba.w.RPSQ.W.
```

Antes de ejecutar el programa es necesario inicializar el registro DI para que apunte a la dirección 130. Para hacer esto se utiliza el comando r seguido del nombre del registro a cambiar:

```
-r di
DI 0000
:130
```

Ya está todo preparado para comenzar la ejecución del programa. Para esto se utilizará el comando t (traza), que ejecuta la siguiente instrucción, mostrando el estado de los registros tras ella.

```
-t
AX=0000 BX=0000 CX=FFFF DX=0000 SP=FFEE BP=0000 SI=0000
DI=0130
DS=0B82 ES=0B82 SS=0B82 CS=0B82 IP=0103 NV UP EI PL NZ NA
PO NC
0B82:0103 30C0 XOR AL,AL
```

Tras repetir el comando t varias veces se llega a la última instrucción del programa, NEG CX, tras la cual el registro CX contiene la longitud de la cadena

```
-t
AX=0000 BX=0000 CX=0007 DX=0000 SP=FFEE BP=0000 SI=0000
DI=0137
DS=0B82 ES=0B82 SS=0B82 CS=0B82 IP=010A NV UP EI PL NZ AC
PO CY
0B82:010A 0476 ADD AL,76
```

Probemos con otra cadena, que se colocará a partir de la posición 150. Obsérvese la mezcla de cadenas y bytes en formato hexadecimal que se hace. El valor 20 corresponde al espacio.

```
-e 150 'segunda' 20 20 'prueba' 00
-d 150
0B82:0150 73 65 67 75 6E 64 61 20-20 70 72 75 65 62 61 00
segunda prueba.
```

Y se inicializa el registro DI para que apunte a la dirección de comienzo de la nueva cadena, y se cambia el valor del puntero de instrucción otra vez al principio del programa.

```
-r ip
IP 010A
```

```
:100
-r di
DI 0137
:150
```

También, esta vez para ir más rápido, se utilizará el comando g (ejecutar), que recibe dos parámetros : el primero es la dirección desde la que comenzará la ejecución del programa y el segundo donde debe detenerse. Si se omite la dirección de inicio se toma, por defecto, la que contiene el registro IP.

```
-g ,10a
AX=0000 BX=0000 CX=0010 DX=0000 SP=FFEE BP=0000
SI=0000 DI=0160
DS=0B82 ES=0B82 SS=0B82 CS=0B82 IP=010A NV UP EI PL NZ
NA PO CY
0B82:010A 0476 ADD AL,76
```

CX contiene la longitud de esta nueva cadena : 10 hex, es decir, 16 caracteres.

Tal y como está escrito este miniprograma de ejemplo, puede considerarse como una subrutina que recibe un parámetro de entrada en el registro DI (la dirección de la cadena) y devuelve un resultado en el registro CX (la longitud). De esta forma, puede transformarse en una subrutina simplemente añadiendo la instrucción RET al final.

```
-a 10a
0B82:010A ret
0B82:010B
```

Ahora se ensamblará un pequeño programa que haga uso de esta rutina para calcular la suma de las longitudes de ambas cadenas

```
-a 110
0B82:0110 mov di,130
0B82:0113 call 100
0B82:0116 mov bx,cx
0B82:0118 mov di,150
0B82:011B call 100
0B82:011E add bx,cx
0B82:0120
```

Obsérvese cómo se utiliza el registro BX para guardar el resultado de la primera longitud, de forma que se le puede sumar luego la segunda.

La elección de este registro se ha hecho teniendo en cuenta que no es modificado por la subrutina. Por ejemplo, elegir el registro AX no sería adecuado, pues la subrutina modifica este registro. También, puede verse que la dirección que se proporciona al CALL es una dirección absoluta que el propio debug se encarga de convertir en relativa de forma automática.

Para comprobar el programa, se coloca el puntero de instrucción en la dirección de comienzo y se hace una ejecución paso a paso :

```
-r ip
IP 010A
:110
-t
AX=0000 BX=0000 CX=0010 DX=0000 SP=FFEE BP=0000 SI=0000
DI=0130
DS=0B82 ES=0B82 SS=0B82 CS=0B82 IP=0113 NV UP EI PL NZ NA
PO CY
0B82:0113 E8EAFF CALL 0100
```

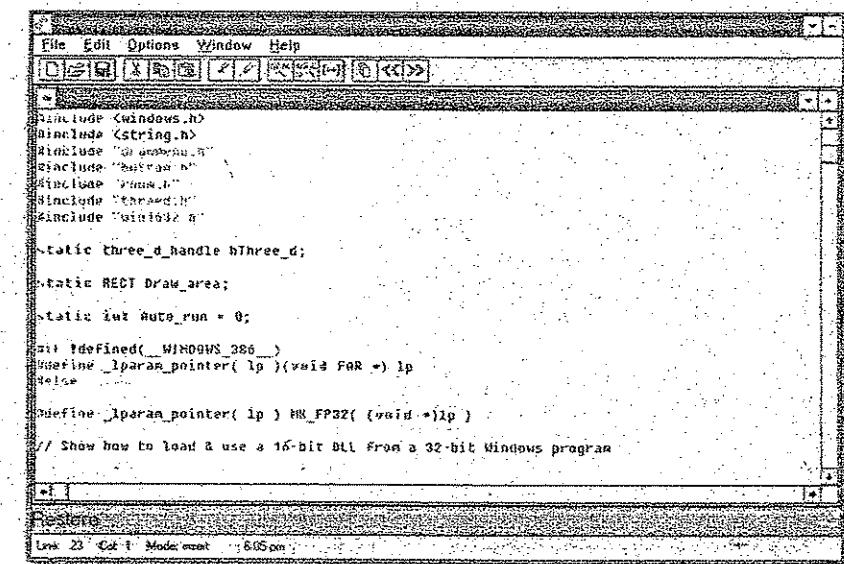
Si se utilizase ahora el comando t, la siguiente instrucción que se ejecutaría sería la situada en la posición 100. Pero como ya sabemos que la subrutina funciona bien, es posible evitar todas sus instrucciones y ejecutarla como si fuese una instrucción simple utilizando el comando p

```
-p
AX=0000 BX=0000 CX=0007 DX=0000 SP=FFEE BP=0000 SI=0000
DI=0137
DS=0B82 ES=0B82 SS=0B82 CS=0B82 IP=0116 NV UP EI PL NZ AC
PO CY
0B82:0116 89CB MOV BX,CX
Se continúa utilizando t y p hasta llegar al final, punto en el que BX contendrá la suma de ambas longitudes
AX=0000 BX=0017 CX=0010 DX=0000 SP=FFEE BP=0000 SI=0000
DI=0160
DS=0B82 ES=0B82 SS=0B82 CS=0B82 IP=0120 NV UP EI PL NZ NA
PO NC
0B82:0120 8BF7 MOV SI,DI
```

Por último, para abandonar debug se utiliza el comando q

```
-q
```

Aunque debug es muy simple, resulta extremadamente útil para experimentar cómo funcionan pequeños programas.



A screenshot of a Windows Notepad window showing assembly code. The code includes includes for windows.h, string.h, stdio.h, stdlib.h, and win32.h, defines for WJN0WS_326, and a main loop with a switch statement. The code is annotated with comments explaining how to load and use a 16-bit DLL from a 32-bit Windows program.

```
File Edit Options Window Help
[Windows Notepad window title bar]

#include <windows.h>
#include <string.h>
#include <stdio.h>
#include <stdlib.h>
#include <win32.h>
#include <thread.h>
#include <win32.h>

static three_d_handle hThree_d;
static RECT Draw_area;
static int Auto_Run = 0;

#define _WJN0WS_326_
#define _lparam_pointer( lp ) (void FAR * ) lp
#define _lparam( lp ) (void FAR * ) lp

// Show how to load & use a 16-bit DLL from a 32-bit Windows program

// Main program entry point
int WINAPI WinMain( HINSTANCE hInstance, HINSTANCE hPrevInstance, LPVOID lpCmdLine, int nCmdShow )
{
    // Initialize the application
    if( !Init() )
        return 0;

    // Start the main loop
    while( !Auto_Run )
    {
        // Check for messages
        if( GetMessage( &msg, NULL, 0, 0 ) )
        {
            // Process message
            TranslateMessage( &msg );
            DispatchMessage( &msg );
        }
        else
        {
            // No messages, do something else
        }
    }

    // Application exit
    return 0;
}

// Initialization function
BOOL Init()
{
    // Load the 16-bit DLL
    hThree_d = LoadLibrary( "3D.dll" );
    if( !hThree_d )
        return FALSE;

    // Get the address of the 16-bit DLL's entry point
    hThree_d = GetProcAddress( hThree_d, "Three_D" );
    if( !hThree_d )
        return FALSE;

    // Call the 16-bit DLL's entry point
    if( !CallWindowProc( hThree_d, hMainWnd, WM_PAINT, 0, 0 ) )
        return FALSE;

    return TRUE;
}
```

Programación en Ensamblador



La creación de programas directamente en código máquina con Debug es aceptable sólo para pequeños ejemplos, pues el extenso uso que hace de direcciones de memoria dificulta tanto el proceso de programación como las modificaciones, en caso de que se quiera realizar alguna modificación al programa.

Para solventar estas deficiencias existe lo que se denomina lenguaje ensamblador, que proporciona un nivel superior de programación permitiendo la utilización de nombres simbólicos para referirse a direcciones de memoria. Una vez creado el programa, éste es convertido a código máquina mediante un programa especial denominado programa ensamblador o, simplemente, ensamblador, siendo en este paso donde se le asigna una dirección de memoria concreta a cada una de las direcciones simbólicas. Además, el ensamblador también ofrece una serie de ayudas destinadas a facilitar la programación, como son la posibilidad de crear programas complejos por partes o módulos, la creación de nuevas instrucciones a partir de las existentes (macro), el control de algunos errores o la posibilidad de definir tipos complejos de datos.

Al contrario que sucede con los lenguajes de alto nivel, que disponen de funciones complejas como la escritura de un texto en pantalla o el cálculo de una función trigonométrica, el lenguaje ensamblador únicamente es capaz de realizar las mismas acciones que son posibles desde el código máquina. Esto asegura que existe una equivalencia directa entre lenguaje ensamblador y código máquina, por lo que la eficiencia y calidad del código depende únicamente de nosotros mismos y no de la persona que ha creado el programa ensamblador.

Una diferencia importante entre código máquina y ensamblador es que el código máquina es fijo y definido por el fabricante del procesador, mientras que el lenguaje ensamblador cambia de unos programas a otros, definiendo cada uno de ellos la forma de expresar las instrucciones, los modos de direccionamiento, la definición de direcciones simbólicas y el resto de ayudas a la programación. Por poner un símil, la palabra "mesa" se escribe de diferente forma según el idioma ("table" en inglés, "mesa" en español...), pero en cualquier idioma hace siempre referencia al mismo concepto: un tipo de mueble.

Además del juego de instrucciones expuesto en el capítulo anterior, el ensamblador cuenta con una serie de instrucciones especiales conocidas como directivas, que no generan código en el programa resultante, pero que sirven para indicarle al programa ensamblador cómo debe realizar su trabajo.

Programas necesarios para la programación en ensamblador

Para la programación de un programa ensamblador se necesita, en primer lugar, un editor de texto. La única condición que debe incluir obligatoriamente este editor es la posibilidad de leer y escribir en formato ASCII. El sistema operativo MS-DOS incluye EDIT, un editor que sirve perfectamente para este propósito. En el CD-ROM que acompaña a este libro encontrará, además, algún otro programa de este tipo, alguno de ellos especialmente diseñado para este propósito.

Lo siguiente que se necesita es, obviamente, el programa ensamblador. El primero de este tipo que apareció para PC, y que marcó el estándar, fue el Macro Assembler de Microsoft. Otro ensamblador muy extendido es Turbo Debugger de Borland, que además de ser totalmente compatible con el producto de Microsoft, ofrece la posibilidad de utilizar una nueva sintaxis de ensamblador mucho más comprensible. Además de estos dos productos, es posible encontrar también otros ensambladores shareware y freeware, cada uno de los cuales utiliza su propia sintaxis y método de programación.

Explicar el lenguaje ensamblador de todos estos productos sería demasiado extenso a la vez que, sin duda, crearía un gran número de confusiones. Puesto que la sintaxis del Macro Assembler es la más extendida, será la que se trate de forma exclusiva en este libro. Sin embargo, recuerde que las diferencias de unos ensambladores a otros son mínimas y que, conociendo los conceptos de un ensamblador, es posible adaptarse rápidamente a la sintaxis de otro diferente.

Otro programa que será de gran ayuda es un depurador o *debugger*. Se trata de un programa con el cual es posible seguir la ejecución de un programa una vez finalizado, con el propósito de comprobar su funcionamiento o detectar algún error en el mismo. Debug, por tanto, es un programa de depuración muy simple que viene con el MS-DOS y que puede utilizarse para este propósito. Además, los fabricantes de ensambladores suelen incluir su propio depurador, mucho más completos y con un aspecto más agradable gracias a la utilización de menús, ratón y ventanas. Además, estos depuradores avanzados permiten seguir tanto el código máquina como el código en lenguaje ensamblador, lo que permite conocer fácilmente en qué punto del programa se encuentra en cada momento.

Construcción de un programa en ensamblador

La construcción de un programa ensamblador comienza por la escritura en el editor de texto del programa en ensamblador. Por convenio, a los

ficheros que contienen código en lenguaje ensamblador se les añade la extensión .ASM.

Una vez finalizado se llama al programa ensamblador, pasándole como parámetro el nombre del fichero de texto que contiene el programa que se va a ensamblar. Tras unos instantes, el programa ensamblador convertirá el contenido del fichero de texto en el código máquina correspondiente, dejándolo almacenado en un archivo en formato de "código objeto". Sólo en el caso de que el programa contenga algún error de tipo sintáctico el ensamblador emitirá un mensaje de error indicando la línea en que se detectó el fallo, y no se generará el fichero objeto.

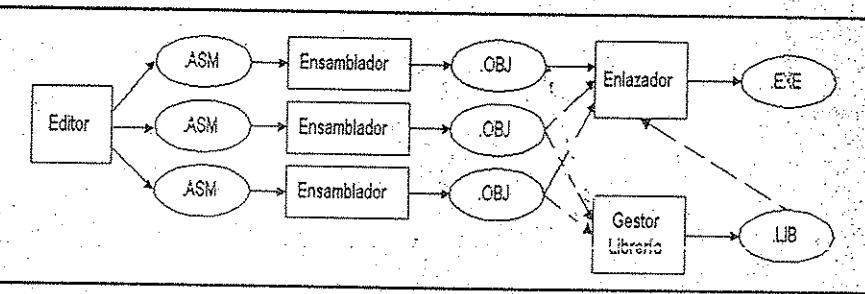
El formato de "código objeto" es utilizado por el ensamblador para permitir la programación por módulos, es decir, mantener varios ficheros para crear un programa ensamblador, cada uno de los cuales realiza una función determinada. Por ejemplo, en la realización de un juego, se podría tener un módulo con todas las rutinas de dibujo en pantalla, otro para la lectura de un joystick, otro para guardar y cargar partidas y otro que gestionase el desarrollo del juego. Esta forma de programar no sólo tiene la ventaja de trabajar con ficheros de código más pequeños, sino que facilita la utilización de rutinas en varios proyectos (el módulo de dibujo en pantalla podría utilizarse para otro juego), reduciendo el tiempo de ensamblado, pues en el caso que se realicen cambios en un módulo sólo haría falta reensamblar dicho módulo, sin necesidad de tocar los módulos restantes. Puesto que unos módulos pueden necesitar llamar a rutinas contenidas en otros, los ficheros de código objeto contienen tanto el código máquina del módulo como todos los símbolos que necesita utilizar de otros módulos, así como los símbolos que posee y que pueden ser utilizados por otros módulos.

Lá unión de todos los módulos objeto que forman un programa y su conversión en el programa definitivo la lleva a cabo una utilidad denominada enlazador o linker. Este programa recibe como parámetros los nombres de los módulos que compondrán el programa, observa qué elementos necesita cada módulo y establece los enlaces necesarios para que el programa funcione.

En caso de que el programa esté compuesto por un gran número de módulos objeto, es posible agruparlos dentro de un único fichero conocido como "librería". A la hora de realizar el enlace bastará con dar el nombre de la librería para que el programa enlazador obtenga de ella los módulos que necesite, evitando así el tener que especificar de forma explícita cada uno de los necesarios. Estas librerías son creadas y mantenidas por un programa especial que cuenta con opciones de

mostrar los módulos contenidos en una librería, añadir nuevos módulos o eliminar alguno de los que contiene.

El siguiente esquema muestra el proceso completo de creación de un programa en ensamblador cuyo código se encuentra repartido en tres archivos separados. Se muestra tanto el proceso simple en el que se enlazan los tres archivos objeto como la posibilidad de crear una librería con algunos de ellos (flechas discontinuas).



Utilización de los programas

Tanto el programa ensamblador como el enlazador y el gestor de librerías son utilidades de línea de comandos, es decir, no cuentan con un interfaz de usuario y todas las opciones deben indicarse escribiéndolas a continuación del nombre del programa.

La conversión de los ficheros contenido el código en lenguaje ensamblador (.ASM) a ficheros objeto (.OBJ) se lleva a cabo a través del programa ensamblador. En el caso de Microsoft, este programa se llama MASM, mientras que el de Borland es TASM. Tras él aparece siempre el nombre del fichero que contiene el programa en ensamblador, teniendo en cuenta que si tiene la extensión .ASM no hace falta escribirlo.

Siguiendo al nombre del fichero fuente pueden colocarse hasta tres parámetros opcionales, separados por comas, y que representan respectivamente el nombre que tendrá el fichero objeto resultante, un nombre para crear un fichero de listado que muestra el código máquina generado tras el ensamblado y, por último, el nombre de un fichero de referencias cruzadas, el cual contiene todos los símbolos que aparecen en el programa, así como todos los números de líneas que hacen referencia a cada uno de ellos.

Existen también una serie de opciones de ensamblado compuestas por el símbolo slash (/) seguido de una letra, y que modifican algún aspecto del

ensamblado del programa. Generalmente no es necesario indicar ninguna, pero en caso de que sea necesario utilizar alguna de ellas, hay que colocarlas entre el nombre del programa ensamblador y el del fichero fuente.

Por tanto, el formato general de llamada al programa ensamblador es el siguiente:

MASM [opciones] fuente [,objeto] [,listado] [,refcruzada]

En caso de que no se especifique un nombre de módulo objeto, se tomará el mismo nombre del fichero origen, pero con la extensión .OBJ.

El enlazador también tiene sus propios parámetros. Tras su nombre pueden utilizarse una serie de modificadores compuestos por / seguido de una letra. A continuación se colocan hasta cuatro parámetros más separados por comas: los módulos objeto que componen el programa, los nombres de las librerías que contienen módulos adicionales, el nombre del fichero resultante y el nombre de un fichero de mapa. En caso de que haya varios módulos objeto o varias librerías, éstos se separarán unos de otros mediante el signo más (+).

El fichero de mapa es un archivo de texto que contiene información acerca de los módulos que se han unido y qué posición dentro del programa se ha asignado a cada uno de ellos.

El formato de llamada al programa enlazador es como sigue:

LINK [opciones] objeto [,librería] [,ejecutable] [,mapa]

Para que quede más claro, veamos cómo sería el proceso necesario para crear un programa compuesto por dos módulos (m1.ASM y m2.ASM) en ensamblador:

1. Ensamblar el primer módulo para crear m1.OBJ

MASM m1

2. Ensamblar el segundo módulo para generar m2.OBJ, pero se desea obtener un fichero de listado m2.LST

MASM m2,m2,m2

3. Enlazar los dos archivos para crear el programa PROG.EXE

LINK m1+m2,,PROG.EXE

Como no hace falta ninguna librería, se omite el segundo parámetro, pero no la coma correspondiente.

La gestión de librerías se lleva a cabo mediante el programa LIB (TLIB para Borland), el cual va seguido del nombre de la librería y de la secuencia de operaciones que debe llevar a cabo sobre ella. La forma de expresar estas acciones es mediante signos matemáticos con el siguiente significado:

Operación	Acción
+ módulo	Añadir módulo a la librería
- módulo	Eliminar módulo de la librería
± módulo	Sustituir módulo en la Librería
*	Extraer módulo de la librería
**	Extraer y eliminar módulo

Formatos de ficheros ejecutables en MS-DOS

El sistema operativo MS-DOS dispone de dos tipos de ficheros ejecutables, cada uno de los cuales requiere poseer ciertas propiedades y con un proceso de creación ligeramente diferente.

El formato EXE es el más utilizado, y es el que se genera por defecto cuando se sigue el proceso indicado anteriormente. Las propiedades de un programa EXE son las siguientes:

- Puede tener cualquier tamaño, hasta el máximo de memoria convencional del sistema.
- Puede especificar una zona de memoria que se utilizará como pila.
- La ejecución del programa puede comenzar en cualquier punto del código, no necesariamente el principio.

Por contra, los programas de tipo COM están mucho más limitados. Las condiciones que deben cumplir son las siguientes:

- Deben ocupar un único segmento, por lo que su tamaño no puede superar los 64 Kilobytes.
- No deben especificar una posición para la pila, pues el sistema operativo les asigna como segmento de pila el mismo que el del código e inicializa el puntero de pila al final de dicho segmento. Por tanto, cuanto mayor sea el código de un programa COM, menos sitio quedará para la pila.
- Deben reservar los primeros 256 bytes de su segmento (0 al 255) para que el MS-DOS guarde sus datos.

- La ejecución de estos programas comienza con la primera instrucción situada en el segmento, es decir, en la posición 100h (256).
- Además, el proceso de creación de este tipo de programas es más complejo; variando según el ensamblador que se utilice:
- Para el MASM se generará un programa .EXE que dará un mensaje de error, pues no se ha especificado la pila. A continuación se utiliza el programa EXE2BIN para convertir este programa EXE al formato COM.
- Para el TASM basta indicar la opción /t al enlazador: TASM /t modulo.OBJ

Inicio y finalización de un programa ensamblador

Los programas hasta ahora vistos eran pequeñas secuencias de instrucciones que funcionaban de forma dependiente del programa

TABLA.1

Desplazamiento	Tamaño	Contenido
0h	2	Instrucción INT 20h
2h	2	Tamaño de la memoria disponible en bloques de 16 bytes (párrafos)
4h	1	Reservado, normalmente vale 0
5h	5	Llamada a la función despachadora de trabajos
Ah	4	Dirección de salto al finalizar
Eh	4	Dirección de salto al pulsar Ctrl-Break
12h	4	Dirección de salto si se produce un error crítico
16h	22	Utilizado por MS-DOS
2Ch	2	Segmento de las variables de entorno
2Eh	34	Área de trabajo del MS-DOS
50h	3	Instrucciones INT 33, RETF
53h	2	Reservado para MS-DOS
55h	7	Extensión del primer FCB
5Ch	9	Primer FCB
65h	7	Extensión del segundo FCB
6Ch	20	Segundo FCB
80h	1	Longitud de los parámetros
81h	127	Parámetros

DEBUG, permitiéndonos controlar su comienzo y final. Sin embargo, a la hora de escribir programas autónomos hay que tener en cuenta que éstos son puestos en marcha por el sistema operativo MS-DOS. El usuario interactúa con este sistema operativo a través del programa COMMAND.COM, el cual muestra una línea de comandos que recibe las órdenes que debe llevar a cabo. Cuando se ejecuta un programa escribiendo su nombre en la línea de comandos del COMMAND.COM, lo primero que éste hace es reservar la memoria RAM necesaria para ejecutar dicho programa; crea una estructura de datos especial conocida con el nombre de PSP (Prefijo Segmento de Programa) que provee información sobre los parámetros que se pasaron al programa o la forma de acceder a las variables de entorno. La siguiente tabla muestra los distintos datos contenidos en el PSP, así como su significado: [ver tabla.1]

De estos campos, los más utilizados son el segmento de cadenas de entorno y los parámetros.

El segmento de cadenas de entorno apunta a un segmento que contiene las variables definidas mediante la instrucción SET de MS-DOS. Todas estas variables están almacenadas de forma consecutiva, separando unas de otras mediante un byte con el valor 0. El final de las cadenas se marca con dos bytes con valor cero.

Los parámetros indicados en la línea de comandos tras el nombre del programa se encuentran situados a partir de la dirección 81h del PSP. Además, el byte de la posición 80h indica la longitud de la línea de parámetros.

Una vez creado el PSP, se carga el programa y se inicializan los registros DS y CS para que apunten hacia el PSP recién creado. En el caso de programas COM, DS y ES tendrán el mismo valor que CS, y el PSP ocupará los primeros 256 bytes de este segmento. Los programas EXE guardan el PSP en una zona de memoria separada, por lo que deberán guardar el valor del registro DS al comienzo del programa para poder acceder más tarde a los datos del PSP.

Una vez que el programa finalice, debe indicar al sistema operativo este hecho. Para ello tiene dos posibilidades. La forma más general consiste en colocar el valor 4Ch en el registro AH y un valor de retorno en AL; llamando a continuación a la interrupción 21h. Este valor de retorno que se deposita en AL puede ser chequeado por otros programas o mediante la instrucción ERRORLEVEL dentro de un fichero de proceso por lotes (.BAT). Por ejemplo, si se desease finalizar un programa devolviendo un valor de retorno de 4, se utilizaría el siguiente código:

```
MOV AH,4Ch ;Estas dos instrucciones pueden sustituirse por
MOV AL,4 ; la instrucción simple: MOV AX,4C04h
INT 21h
```

La forma alternativa de finalizar un programa es mediante una interrupción 20h. Este sistema no permite devolver un valor de retorno, y además tiene el inconveniente de que el registro CS debe encontrarse apuntando al segmento del PSP.

Valores numéricos y símbolos en ensamblador

Antes de profundizar en otros aspectos del lenguaje ensamblador, es interesante detenerse un momento a comprender cómo utiliza éste los valores numéricos y simbólicos que aparecen en el programa y cómo deben indicarse los mismos.

En principio, el ensamblador utiliza como base por defecto la decimal, siendo posible utilizar tanto cifras hexadecimales como octales y binarias, siempre y cuando se les añada al final la letra terminadora que indica su base (q u o para octal, h para hexadecimal y b para binario).

Sin embargo, y debido a ciertos motivos de diseño del lenguaje ensamblador, es obligatorio que los números comiencen siempre por un dígito. Esto es cierto para todos los sistemas anteriores excepto el hexadecimal, pues al utilizar seis letras como dígitos (A a F) es posible que un número comience por alguna de estas letras. Para solventar esta situación lo que se hace es colocar un cero adicional a la izquierda de las cifras hexadecimales que presenten este problema.

0111101b	;valor binario
10	;valor decimal
10h	;valor hexadecimal
377q	;valor octal
A73h	;hexadecimal que produce error, pues comienza por la letra A
0A73h	;la misma cifra de antes, pero que no da error.

Los símbolos, por su parte, son nombres descriptivos que se utilizan para designar variables, direcciones de memoria, subrutinas, macros, segmentos y valores constantes. Estos nombres se componen de una combinación tanto de caracteres alfabéticos como numéricos, pero con la salvedad de que el primer carácter no puede ser de tipo numérico. Por caracteres alfabéticos se entienden las letras de la a a la z, tanto en mayúsculas como minúsculas, así como los caracteres de subrayado (_), interrogación (?), dólar (\$) y arroba (@). Hay que tener en cuenta también que no pueden utilizarse como nombres de símbolo nombres que ya se encuentran definidos por el ensamblador, como es el caso de los nombres de las instrucciones, de los registros o de las directivas del ensamblador.

Existen dos parámetros de los símbolos que varían de unos ensambladores a otros, que son la longitud máxima en caracteres y el tratamiento de mayúsculas y minúsculas. Por norma general, la longitud máxima de un símbolo se sitúa en torno a los treinta y dos caracteres y se tratan igual las mayúsculas que las minúsculas, por lo que nombres como "abc12", "ABC12" y "aBc12" representan al mismo símbolo.

Ejemplos de símbolos válidos son: Dato, _Dato, Dato12, Dato_primer, Dato\$12.

Ejemplos no válidos son los siguientes: 12Dato (comienza por número), Call (es el nombre de una instrucción), Assume (es el nombre de una directiva).

Estructura de un programa en ensamblador

La mejor forma de ver la estructura de un programa en ensamblador es a través de un ejemplo sencillo, que además servirá para recordar los conceptos del código máquina del 8086.

El ejemplo escogido es un programa que muestra en pantalla la cadena pasada como parámetro, de la misma forma que hace el comando ECHO de MS-DOS. Para ello, se utiliza la función del sistema operativo que muestra en pantalla una cadena cuyo final se marca por el carácter \$. Según se verá más adelante, esta función se llama a través de la interrupción software 21h, pasando en el registro AH el valor 9, mientras que la cadena a imprimir se encuentra en el segmento de datos y el desplazamiento indicado por el registro DX, es decir, comienza en DS:[DX].

Puesto que se trata de una aplicación muy simple, se utilizará el formato de archivo ejecutable COM.

```
Código SEGMENT
ASSUME CS:Código, DS:Código
ORG 100h
inicio: MOV BL,80h      ;Lee la longitud de la línea de
                           ;comandos al registro BL
                           ;XOR BH,BH
                           ;MOV [BX+81h],'$' ;Coloca el signo $ como final de cadena
                           ;MOV DX,81h
                           ;MOV AH,9
                           ;INT 21h
                           ;INT 20h
                           ;Código ENDS
                           ;END inicio
```

Comencemos con la línea de la primera instrucción:

inicio: MOV BL,80h ;Lee la longitud de la línea de comandos al registro BL

En ella aparece en primer lugar una etiqueta simbólica, la cual hace referencia a la posición de memoria en la que se ubicará la instrucción que va a continuación (MOV BL,80h). Tras esta instrucción aparece un comentario, cuyo comienzo se marca por un punto y coma, y que ocupa el resto de la línea. Estos comentarios son información descriptiva para que las personas que lean el programa puedan comprenderlo más fácilmente y no tienen significado alguno para el programa ensamblador, que los ignora. Los comentarios son especialmente útiles en los programas realizados en ensamblador, pues el bajo nivel de este lenguaje hace que los programas creados con él sean complicados de entender qué es lo que realmente están haciendo en un momento determinado. Por tanto, es mejor excederse en comentarios que quedarse corto.

Otra característica del ensamblador es que pueden utilizarse caracteres como si fuesen bytes colocándolos entre comillas simples. Esto es lo que sucede en la instrucción MOV [BX+81h],'\$. Durante el ensamblado se realizará la conversión necesaria de carácter a byte, convirtiéndose la instrucción anterior en MOV [BX+81h],24h, pues 24h es el código ASCII del carácter '\$'.

Todo el programa se encuentra contenido en un bloque, cuyo principio lo marca la linea **codigo SEGMENT**, y su final **codigo ENDS**.

Estas dos directivas delimitan lo que se conoce como un "segmento", es decir, un bloque de código y/o datos que forma una unidad indivisible y que se encuentra contenido enteramente dentro de un segmento. El nombre que se le da al segmento aparece tanto al definir su principio como su final, y representa el valor del segmento físico de memoria en la que se encuentra el bloque. Según esto, para conseguir que el registro DS apuntase al segmento definido no habría más que hacer:

MOV AX,codigo	;No se puede hacer de forma directa.
MOV DS,AX	;por lo que se recurre al registro AX de forma temporal

De todas formas, la carga de segmentos sólo es posible en ficheros de tipo EXE, pues los de tipo COM, al constar de un único segmento, ya apuntan siempre a él.

La tercera y cuarta línea del programa vienen determinadas por el hecho de que se trate de un programa de tipo COM. ASSUME es una

directiva que permite informar al ensamblador hacia qué segmentos apunta cada uno de los registros de segmento. Debe tenerse en cuenta que el ensamblador NO hace que los registros de segmento apunten a los segmentos indicados por la directiva ASSUME, sino que debe ser el programador el que lleve a cabo este proceso. La finalidad de ASSUME es, simplemente, que el ensamblador coloque de forma automática los prefijos de segmento adecuados a cada modo de direccionamiento. Por el momento no entraremos en más detalles, solamente se dará la idea que en este caso ASSUME está informando al ensamblador que el MS-DOS ha inicializado DS y ES para que apunten al mismo segmento de código, el llamado "codigo".

La otra directiva, ORG , indica al ensamblador que debe comenzar a ensamblar a partir del desplazamiento 100h. Esto es debido a que antes de esta dirección se encuentra el PSP, el cual no debe ser modificado.

La última línea, END, marca el final del programa ensamblador e indica la dirección de la primera instrucción del mismo. Puesto que se trata de un programa COM, la dirección será la situada en el desplazamiento 100h, es decir, la primera.

Una vez escrito el programa, se procede a guardarlo en el disco duro con el nombre "ECO.ASM". Tras ello se realiza el proceso de ensamblado:

Microsoft : MASM ECO.ASM
Borland : TASM ECO.ASM

Y a continuación el enlazado:

Microfoft:	LINK ECO.OBJ
	EXE2BIN ECO.EXE
	REN ECO.BIN ECO.COM
	DEL ECO.EXE
Borland:	TLINK /t ECO.OBJ

Tras estos pasos, el programa quedará en el disco con el nombre ECO.COM. Pruebe a ejecutarlo pasando varias cadenas como parámetro:

ECO prueba
ECO una cadena más larga

Etiquetas

Una de las facilidades que aporta el lenguaje ensamblador es la posibilidad de trabajar con direcciones simbólicas; más fáciles de recordar que los números hexadecimales de que se compónen las direcciones físicas.

El lenguaje ensamblador representa estas direcciones si, si como etiquetas, las cuales son una cadena de caracteres alfanuméricos que identifican la posición de memoria en la que comienza una instrucción del programa o alguno de sus datos.

Cualquier etiqueta que aparezca en un programa debe encontrarse definida en un único lugar de éste. La forma general de crearlas es a través de la directiva LABEL, la cual va precedida por el nombre de la etiqueta y le siguen una serie de atributos que designan si se trata de una etiqueta de código o de datos así como su tipo. La dirección de memoria que se le asigna a la etiqueta es la que tendrá la siguiente instrucción o dato que encuentre el ensamblador.

Para las etiquetas de código sólo existen dos valores posibles de atributo y que tienen relación con la distancia de salto de las instrucciones que hagan referencia a ellas. Estos atributos son FAR y NEAR. El ensamblador utiliza la información de la directiva LABEL para que, cuando encuentra una instrucción de transferencia de control (CALL o JMP) a dicha posición sepa qué tipo de salto debe utilizar (FAR o NEAR).

El siguiente fragmento de programa, aunque no tiene utilidad, ilustra la definición de las etiquetas de código :

```
Parte1: LABEL NEAR
        MOV AX,10
        JMP Parte1
Parte2: LABEL FAR
        JMP Parte2
```

En este pequeño fragmento de código pueden verse dos etiquetas que son de código, pues la siguiente línea del programa es una instrucción de código. La etiqueta Parte1 designa la dirección de memoria de la instrucción MOV AX,10 de la misma forma que Parte2 designa la posición de memoria de la instrucción JMP Parte2. Puesto que Parte1 es una etiqueta NEAR el salto de la tercera línea (JMP Parte1) será un salto de tipo NEAR mientras que JMP Parte2 será un salto de tipo FAR que al tener por destino del salto a sí mismo, provoca un bucle infinito en el programa.

Puesto que las etiquetas de código de tipo NEAR son muy utilizadas, existe una forma abreviada de definirlas, consistente en colocar simplemente el nombre de la etiqueta finalizado por dos puntos (:). Según esto, el siguiente código :

```
Parte1: LABEL NEAR
        MOV ax,10
```

puede escribirse como :

```
Parte1:
        MOV ax,10
```

e incluso las líneas pueden agruparse en una sola, dando lugar a la forma más conocida de etiqueta :

```
Parte1: MOV AX,10
```

Es posible colocar varias directivas LABEL consecutivas con diferentes nombres y atributos para referencias así a una misma posición de memoria de diferentes formas. Esto resulta útil para poder saltar a una misma posición tanto mediante salto de tipo FAR como NEAR. Por ejemplo :

```
Parte1: LABEL FAR
Parte2: MOV AX,10
        JMP Parte2
        JMP Parte1
```

Las dos instrucciones de salto provocan una transferencia del control del programa a la misma posición de memoria, pero el salto a Parte1 será de tipo FAR mientras que el de Parte1 será NEAR.

Un detalle importante es que la etiqueta se define exactamente en la instrucción o dato al que referencia, pero puede utilizarse en cualquier parte del programa, incluso en partes del código anteriores a la propia definición de la instrucción. Precisamente este caso especial, en el que se utiliza una etiqueta que se define más adelante del programa, es el causante de que los ensambladores deban realizar dos pasadas al código para poder resolver de forma adecuada estas "referencias adelantadas"; (aunque actualmente ya hay ensambladores que realizan una única pasada). En caso que el ensamblador no sea capaz de averiguar el tipo de salto que debe realizar, es posible indicárselo mediante los modificadores FAR PTR o NEAR PTR, los cuales se sitúan entre la instrucción de salto (JMP o CALL) y la etiqueta de código :

```
JMP FAR PTR Parte1
        CALL NEAR PTR Parte2
```

Las etiquetas de referencia a datos por su parte hacen referencia a una posición de memoria en la que guardar un dato de un tipo determinado y

pueden utilizarse en los modos de direccionamiento sustituyendo los valores constantes de los direccionamientos indirectos, relativos e indexados. Sin embargo la definición de la etiqueta no reserva el espacio de memoria necesario, por lo que este tipo de etiquetas se utiliza en combinación con las directivas de definición de datos para acceder a los datos utilizando un tipo diferente a aquel con el que fueron declarados.

Para las etiquetas de datos los atributos posibles son BYTE, WORD y DWORD para indicar un tamaño de dato de 1, 2 y 4 bytes respectivamente.

Dato LABEL BYTE

NOP

MOV [Dato],AL

Este fragmento de código muestra el uso de una etiqueta de datos para modificar el código de un programa. En él se define una etiqueta de código que referencia a la posición de la instrucción NOP. Por tanto, la siguiente instrucción MOV Dato,AL guardará el valor contenido en el registro AL en la posición de memoria ocupada por la instrucción NOP modificándola.

Definición de datos

Un programa no sólo utiliza la memoria para contener las instrucciones en código máquina que lo forman, sino que habitualmente también se utiliza parte de ésta para almacenar datos que el programa necesitará durante su ejecución, como puede ser un texto que se muestra en pantalla para informar al usuario acerca de algún acontecimiento. Además, los programas suelen necesitar memoria para almacenar variables o valores intermedios generados durante el programa.

El lenguaje ensamblador define para este propósito varias directivas, mediante las cuales es posible reservar espacio de memoria para almacenar datos y, opcionalmente, darles un valor inicial que tendrán al comienzo del programa. En total, son cinco las directivas que existen, cada una de las cuales permite definir un tipo concreto de datos.

Para definir un elemento de datos, el único requisito que hay que cumplir es especificar su tamaño mediante alguna de las directivas de definición de datos. Estas directivas son unas instrucciones especiales que permiten reservar una o varias variables del mismo tamaño y opcionalmente asignarles un valor inicial. Las cinco directivas definidas para este propósito se resumen en esta tabla:

Directiva	Nombre	Tipo de datos	Longitud
DB	Define Byte	Byte	1
DW	Define Word	Word	2
DD	Define Doubleword	Doble word	4
DQ	Define Quadword	Cu-duple word	8
DT	Define Ten bytes	Diez bytes	10

La forma de utilizarlas es colocar la directiva de definición de datos como si fuese una instrucción de ensamblador (precedida, si se desea, de una etiqueta). A continuación se deja un espacio en blanco y se continúa colocando los valores iniciales de las posiciones de memoria, separados por comas. En caso de que se desee simplemente reservar la posición, pero no dar un valor inicial, se colocará un signo de interrogación en lugar de un valor. Por ejemplo:

DB 12h,45,?,17h

;Define cuatro bytes, el tercero sin valor definido. El primer y cuarto valor son hexadecimales (terminados en h), mientras que el segundo es decimal.

DW 1234h, ? ;Define dos words, el segundo sin valor definido.
 DB 12h,1234h,45h ;El segundo valor da error, porque
 1234h no cabe en un byte.

En el caso que se trate de una definición de bytes con DB, es posible indicar caracteres como valores. Como ya se indicó anteriormente, esto hace que el ensamblador inicialice dicha posición de memoria con el código ASCII de dicho carácter. Si lo que se indica es una secuencia de caracteres contenida entre comillas, entonces el ensamblador reservará un byte para cada carácter.

Ejemplo:

DB 'H','o','l','a'
 DB 'Hola'
 ;Reserva 4 bytes con los códigos ASCII
 de la cadena "Hola"
 ;Igual que el anterior, pero más sencillo

Es posible mezclar tanto datos numéricos como de carácter y de cadena en una misma definición, separándolos por comas:

DB 12h,'Hola,45,'H'

Una situación muy frecuente es el caso en que varios datos consecutivos almacenan un mismo valor. Para esto, existe una directiva de repetición de datos denominada DUP, la cual va precedida por un valor numérico que indica el número de datos seguidos al mismo valor, mientras que el dato que se repite va a continuación entre paréntesis:

DB 2 dup (12h) ; 3 dup ('a') ; equivale a DB
 12h,12h,'a','a','a'
 DW 2 dup(1234h), 3 dup (?) ; equivale a DW
 1234h,1234h, ?, ?, ?

Dentro de la directiva dup pueden colocarse varios datos e incluso otras directivas dup, siempre y cuando todos estos datos sean del mismo tipo:

DB 2 dup (12h,'a') ;equivale a DB
 12h,'a',12h,'a'
 DB 2 dup (3 dup (12h), ?) ;equivale a DB 12h,12h,12h,
 ?,12h,12h,12h, ?

Modos de direccionamiento en lenguaje ensamblador

Puesto que el ensamblador utiliza las etiquetas como si fuesen posiciones de memoria, es posible utilizar éstas en los lugares donde se

debería utilizar una posición de memoria. El ensamblador, al procesar el programa, sustituirá estas etiquetas por la dirección que corresponda. Según esto, suponiendo que Datos es una etiqueta válida, es posible utilizarla como desplazamiento dentro de los siguientes modos de direccionamiento:

MOV AX,[Datos]	;Directo
MOV AX,[Datos+BX]	;Relativo a base
MOV AX,[Datos+SI]	;Indexado
MOV AX,[Datos+BX+SI]	;indexado con base

Puesto que los tres últimos direccionamientos se utilizan para acceder a posiciones consecutivas de memoria, como por ejemplo a matrices, es posible escribirlos de otra forma alternativa, más comprensiva, sacando la referencia a la etiqueta fuera del direccionamiento:

MOV AX,[Datos+BX] -> MOV AX,Datos[BX]
MOV AX,[Datos+SI] -> MOV AX,Datos[SI]
MOV AX,[Datos+BX+SI] -> MOV AX,Datos [BX+SI]

Este último modo, el indexado con base, permite separar también los registros base e índice, para poner aún más de manifiesto su utilización para acceder a matrices bidimensionales:

MOV AX,[Datos+BX+SI] -> MOV AX,Datos[BX][SI]
--

Puesto que el modo de direccionamiento directo es uno de los más utilizados, el ensamblador permite omitir los corchetes, como si se tratase de un direccionamiento inmediato:

MOV AX,[Datos] -> MOV AX,Datos

Aunque esta segunda forma de expresión pueda parecer más simple, hay que tener presente que representa un direccionamiento directo y no uno inmediato, es decir, MOV AX,Datos carga en AX el valor contenido en la posición Datos, no el número de dicha posición, como sería de suponer.

En caso de que se desee cargar en un registro la posición de memoria a la que hace referencia una etiqueta, será preciso recurrir bien a la instrucción LEA o a la instrucción MOV, cuyo segundo operando será un valor inmediato obtenido a través de OFFSET. Por ejemplo, las siguientes líneas muestran las dos formas de cargar en el registro BX la dirección donde se encuentra la variable Datos:

LEA BX,Datos
MOV BX,OFFSET Datos

Otra característica interesante del ensamblador es que, dado que conoce el tamaño de los datos definidos en memoria a través de las directivas de definición de los mismos, puede realizar comprobaciones de tipo y elegir el formato de la instrucción adecuado en cada caso. Ya se vio anteriormente que, cuando una instrucción sólo referenciaba a una posición de memoria, era preciso especificar el tamaño mediante byte ptr o word ptr. Pues bien, en ensamblador, si la referencia se hace a una etiqueta, es posible omitir ésto, pues el ensamblador ya conoce el tamaño y lo colocará de forma automática:

```
datos db 12h
INC byte ptr datos ;Ambas instrucciones son
                     ;equivalentes
INC datos ;Como el ensamblador sabe que
           ;datos es de tipo byte (db), añade
           ;el byte ptr de forma automática.
```

El conocimiento de los tipos, además, evita errores en los que se intenta asignar un dato word en un byte o viceversa:

```
MOV datos,AX ;Da un error porque AX es word y
              ;Datos es byte (db)
```

A continuación se presenta un ejemplo que utiliza los conceptos explicados para sumar una serie de números consecutivos en memoria, cuyo final se marca por un byte al valor 0 y almacena su resultado en una variable definida en memoria a tal propósito.

```
codigo SEGMENT
ASSUME DS:código, CS:código
ORG 100h
inicio: JMP programa

Datos DB 1,10h,15,7,25,0E6h,0
suma DW ?

programa: MOV BX,OFFSET Datos
           XOR CX,CX
           XOR AH,AH
bucle:   MOV AL,[BX]
           ;Datos a sumar
           ;Lugar donde depositar el
           ;resultado
           ;Carga la dirección del
           ;primer dato en BX
           ;Inicializa la suma a cero
           ;Coloca a 0 la parte alta de
           ;AX porque los datos son
           ;byte y la suma es de word.
           ;Carga el dato
           ;apuntado por BX
```

```
CMP AL,0
JE fin
ADD CX,AX

INC BX
JMP bucle
fin:  MOV suma,CX

INT 20h
codigo ENDS
END inicio
```

En este ejemplo puede verse, además, una nueva forma de distribuir el código y los datos. Puesto que tanto las líneas de código como las definiciones de datos pueden mezclarse libremente, es conveniente colocar los datos antes del código que los utiliza. Si se sigue esta norma, el ensamblador conocerá cuál es el tipo de cada dato cuando lo encuentre en el código, lo cual le ayudará a generar un código más eficiente. Puesto que el programa que se está creando es de tipo .COM, el cual comienza la ejecución a partir del desplazamiento 100h, es preciso colocar en dicha posición una instrucción de salto condicional que realice la transferencia de control al comienzo real del programa. Si se tratase de un programa de tipo .EXE no existiría este inconveniente, pues en ellos puede colocarse el comienzo del programa en cualquier punto libremente.

Constantes y variables en ensamblador

Una de las posibilidades que ofrece el lenguaje ensamblador es la de asignar un nombre a un valor numérico, de forma que sea posible utilizar dicho nombre en cualquier punto donde se pudiese utilizar el valor que representa. A estos nombres simbólicos se les denomina "constantes de ensamblado", y tienen una doble finalidad. Por un lado, al asignar un nombre descriptivo a un número se facilita la comprensión del programa. Por otra parte, si alguna vez es necesario cambiar dicho valor sólo hará falta realizarlo en la definición del nombre simbólico para hacerlo en todo el programa, sin necesidad de buscar todos los lugares donde se utiliza.

La forma de definir una constante es a través de la directiva EQU, la cual necesita que se preceda del nombre de la constante y vaya seguida del valor que se desea asignar. Por ejemplo, si se desease crear una constante con el carácter que representa el salto de línea en MS-DOS se podría hacer de la siguiente manera:

retorno EQU 0Dh

Más adelante, en el programa bastaría con utilizar el nombre simbólico ancho en todos los lugares donde haga falta este dato. Puesto que además los nombres simbólicos no tienen tipo, es posible utilizarlos en cualquier tipo de instrucciones, como puede verse en los siguientes ejemplos:

Dato:	DB 'Ejemplo', retorno, 'Otra línea', retorno ;Equivale a DB 'Ejemplo',0Dh,'Otra línea',0Dh
MOV AX, retorno	;Equivale a MOV AX,000Dh
MOV AL, retorno	;Equivale a MOV AL,0Dh

Una de las características de las constantes de ensamblado es que su valor no se puede cambiar, y es el mismo durante todo el proceso de ensamblado. Si lo que se desea es utilizar una variable para contener un valor que cambia a lo largo del ensamblado, entonces hay que utilizar las denominadas "variables de ensamblado".

Las variables de ensamblado son similares en concepto a las constantes, pero al contrario que éstas, su valor puede cambiar durante el proceso de ensamblado. La forma de definirlas es colocando su nombre seguido del signo igual y el valor que se desea almacenar en ellas. En cualquier punto del programa es posible cambiar su valor simplemente asignándole cualquier otro.

```
V=1
MOV AX,V ;Equivale a MOV AX,1
V=10
MOV AL,V ;Equivale a MOV AL,10
V=V+1
MOV AH,V ;Equivale a MOV AH,11
```

Hay que tener en cuenta que tanto constantes como variables de ensamblado existen únicamente durante el proceso de ensamblado, y no durante la ejecución del programa resultante.

Expresiones constantes

Si se realiza una serie de operaciones matemáticas sobre una serie de valores constantes, el resultado de todas ellas será a su vez un valor constante. Este simple razonamiento es la base de lo que se conoce como "expresiones constantes", es decir, que en cualquier punto del programa donde puede utilizarse un valor numérico puede utilizarse cualquier

expresión matemática cuyos operandos sean números o constantes o variables de ensamblado.

A tal efecto, el ensamblador dispone de una serie de operadores matemáticos que se resumen en la siguiente tabla: [Tabla.2, página siguiente]

Operadores de variables y etiquetas

Además de los operadores numéricos existen otros adicionales, que toman como argumento el nombre de una variable o etiqueta y devuelven un valor numérico que puede utilizarse en expresiones constantes. Estos operadores son los siguientes:

SEG: Devuelve el segmento de la etiqueta o variable que aparece a continuación. Ejemplo : MOV AX, SEG Datos

OFFSET: Devuelve el desplazamiento de la etiqueta o variable que aparece a continuación.

Ejemplo: MOV AX, OFFSET Datos

LENGTH: Se aplica únicamente a variables y devuelve el valor de repetición indicado con la directiva DUP en la definición de la variable que se indica como argumento. Sólo mira el primer dato de la definición.

Ejemplo :

Datos	DW 1234h,4444h
Datos1	DB 'Prueba', 2 dup (12)
Datos2	DW 10 dup (12)
MOV AX,LENGTH Datos	; Mueve a AX el valor 1 porque LENGTH sólo toma el valor del primer elemento
MOV AX,LENGTH Datos1	;Mueve a AX el valor 1. El dup no afecta porque no es el primer elemento
MOV AX,LENGTH Datos2	;Mueve a AX el valor 10

SIZE : Devuelve el tamaño en bytes del elemento al que referencia su único operando. Si se trata de una referencia a variable, devuelve el número de bytes de cada elemento de datos, y si es una etiqueta devuelve -1 para etiquetas NEAR y -2 para FAR. Para variables de tipo estructura, unión y registro devuelve el número de bytes que necesita un elemento de ellas.

Ejemplo :

etiqueta : MOV AX,SIZE Datos ;AX=2
MOV AX,SIZE Datos1 ;AX=1
MOV AX,SIZE Etiqueta ;AX=-1

WIDTH : Retorna el número de bits de un campo de una variable de tipo registro. Se verá su utilización cuando se hable de los registros.

MASK : Devuelve la máscara necesaria para acceder a un campo de bits de un registro. Al igual que con **WIDTH**, su utilización detallada se verá más adelante.

TYPE : Devuelve un valor numérico indicando el tipo de la variable o etiqueta que va a continuación. Los valores devueltos son los mismos que los del operando **SIZE**.

Procedimientos y subrutinas

Uno de los elementos más importantes dentro de la programación son las subrutinas. Por tanto, el lenguaje ensamblador no podía dejar de tenerlas en cuenta, y nos ofrece una forma de definir rutinas que pueden ser luego llamadas mediante la instrucción CALL.

TABLA.2

Operador	Uso
+	op1 + op2
-	op1 - op2
*	op1 * op2
/	op1 / op2
MOD	op1 MOD op2
NOT	NOT op1
AND	op1 AND op2
OR	op1 OR op2
XOR	op1 XOR op2
SHL	op1 SHL op2
SHR	op1 SHR op2
EQ	op1 EQ op2
EN	op1 NE op2
GT	op1 GT op2
GE	op1 GE op2
LT	op1 LT op2
LE	op1 LE op2
\$	\$
HIGH	HIGH op1
LOW	LOW op1

La definición de subrutinas se lleva a cabo mediante la pareja de directivas PROC /ENDP, las cuales marcan respectivamente el principio y el final de la subrutina. Ambas directivas deben ir precedidas por una etiqueta que indica el nombre que recibe la subrutina, y que es el parámetro que debe especificarse a la instrucción CALL a la hora de realizar la llamada. Esta etiqueta que da el nombre a la subrutina no se finaliza en dos puntos, pero no por ello es de tipo FAR, sino que el tipo de la subrutina se indica a continuación de la directiva PROC mediante las palabras reservadas NEAR o FAR. En caso de omitirse, por defecto se supone NEAR.

Cabría suponer que, puesto que la directiva ENDP marca el final de la subrutina, es el propio ensamblador el que se encarga de colocar de forma automática la instrucción de retorno de la subrutina (RET). Sin embargo, esto no es así, pues una de las características de las directivas es que no generan código, quedando por tanto a cargo del usuario el colocar la instrucción RET para marcar el final de la subrutina. Lo que sí realiza el ensamblador de forma automática es la colocación del tipo NEAR o FAR a la instrucción RET según convenga, así como a las instrucciones CALL que hagan referencia a ella.

El siguiente programa define una subrutina que suma las cantidades apuntadas por el registro SI, cuyo final se marca por una cifra con el valor 0 y devuelve la suma en el registro CX :

Funció	n
Suma dos valores	
Resta el segundo operando del primero	
Multiplica ambos operandos	
Cociente de la división entera de op1 entre op2	
Resto de la división entera de op1 entre op2	
Invierte los bits de op1	
Deja a 1 los bits que son 1 en ambos operandos, y el resto a 0	
Deja a 0 los bits que son 0 en ambos operandos, y el resto a 1	
Deja a 0 los bits con igual valor en ambos operandos y a 1 los que difieren	
Desplaza op1 tantos bits a la izquierda como indica op2	
Desplaza op1 tantos bits a la derecha como indica op2	
Devuelve -1 si op1 es igual que op2. Si no, devuelve 0	
Devuelve -1 si op1 es distinto de op2. Si no, devuelve 0	
Devuelve -1 si op1 es mayor que op2. Si no, devuelve 0	
Devuelve -1 si op1 es mayor o igual que op2. Si no, devuelve 0	
Devuelve -1 si op1 es menor que op2. Si no, devuelve 0	
Devuelve -1 si op1 es menor o igual que op2. Si no, devuelve 0	
Se sustituye por la dirección en la que se está ensamblando la instrucción actual	
Devuelve el byte alto del operando de tipo word op1	
Devuelve el byte bajo del operando de tipo word op1	

```

codigo SEGMENT
ASSUME CS :codigo,DS :codigo
ORG 100h
inicio : JMP programa

Suma PROC NEAR
    XOR CX,CX ;Inicializa la suma a cero
    XOR AH,AH ;Coloca a 0 la parte alta de AH
    bucle: MOV AL,[SI] ;Carga el dato apuntado por SI
            CMP AL,0 ;Lo compara con 0
            JE fin ;Y si es igual, deja de sumar
            ADD CX,AX ;Si no es cero, suma el byte leído a la suma parcial
            INC SI ;Apunta al siguiente dato
            JMP bucle ;Vuelve por otro dato

fin : RET
Suma ENDP

Datos1 DB 12h,15h,17,25,0EFh,0
Datos2 DB 45,17,1,2,0
Suma1 DW ?
Suma2 DW ?

programa:
    MOV SI,OFFSET Datos1
    CALL Suma
    MOV suma1,CX
    MOV SI,OFFSET Datos2
    CALL Suma
    MOV suma2,CX
    INT 20h

codigo ENDS
END inicio

```

Al igual que se hizo anteriormente con los datos, también es conveniente que la definición de la subrutina se encuentre antes de su inicialización, pues de esa forma el ensamblador conocerá su tipo (FAR o NEAR) lo antes posible y podrá generar el tipo de instrucción CALL necesario en cada caso.

PASO DE PARAMETROS A SUBRUTINAS

Una forma de ver una subrutina es como un bloque que recibe unos datos de entrada denominados parámetros, los procesa y devuelve uno o varios valores de resultado. En general, existen tres formas de llevar a cabo este traspase de datos entre el programa y la subrutina: a través de registros, mediante variables globales y a través de la pila.

El método de paso de parámetros a través de registros es el más rápido y el más utilizado por los programas en ensamblador. Consiste simplemente en colocar cada uno de los datos que precisa la subrutina en un registro del procesador establecido de antemano. La rutina, una vez que finalice su trabajo, devolverá sus resultados también en los registros.

Por ejemplo, el siguiente código muestra una rutina que recibe dos números de 16 bits, uno en el registro AX y otro en el BX, y devuelve como resultado la suma de ambas cantidades en el registro AX, y su diferencia en el BX.

Sumaresta	PROC		
	MOV CX,AX	;Guarda de forma temporal la primera cantidad	
	ADD AX,BX	;Realiza la suma	
	SUB CX,BX	;Realiza la resta	
	MOV BX,CX	;Coloca el resultado en el registro de resultado	
	RET	;Retorna	
Sumaresta	ENDP		
programa :	MOV	AX,0A784h	;Coloca primer parámetro en AX
	MOV	BX,1273h	;Segundo parámetro en BX
	CALL	Sumaresta	;Llamada
	ADD	AX,BX	;AX y BX contienen los resultados

Aunque rápido y eficiente, el método de paso de parámetros por registro también tiene sus inconvenientes, los cuales se originan por el hecho de que el procesador 8086 tiene un número muy limitado de registros. Esta carencia provoca que en muchas ocasiones haya que enviar el contenido de un registro a la pila para poder utilizarlo en la llamada a una subrutina para, una vez retorne, volver a extraerlo de la pila. Por otra parte, puede ser que se necesite pasar más parámetros que registros hay, en cuyo caso habría que utilizar alguno de los otros dos métodos.

El sistema de variables globales utiliza para su comunicación con las subrutinas una serie de variables en las que se dejan los parámetros antes de

llamar a la subrutina. Una vez que finaliza la rutina, ésta deja sus resultados en otra serie de variables reservadas a tal efecto. Reescribiendo el ejemplo anterior para que haga uso de este sistema, quedaría como sigue:

```

param1    DW  ?
param2    DW  ?
Resul1   DW  ?
Resul2   DW  ?

SumaResta PROC
    MOV AX,param1
    ADD AX,param2
    MOV Resul1,AX
    MOV AX,param1
    SUB AX,param2
    MOV Resul2,AX

    RET
SumaResta ENDP

programa : MOV Param1,0A784h
            MOV Param2,1273h
            CALL SumaResta
            MOV AX,Resul1
            ADD AX,Resul2

```

Este programa realiza la misma función que el anterior, y además hace uso de un único registro, el AX. Sin embargo, su ejecución será más lenta, pues hace un número más alto de accesos a la memoria. Además, las variables definidas para el paso de parámetros se encuentran presentes durante todo el programa, aún cuando la subrutina sólo se utilice en una única ocasión, con el consiguiente desperdicio de memoria. Quizás se podría pensar, como solución a este problema, en utilizar siempre las mismas variables para el paso de parámetros a todas las subrutinas, un método válido pero muy propenso a los errores, puesto que si se llamase a una subrutina desde dentro de otra habría que buscar algún método para guardar los parámetros de la subrutina actual antes de sobreescribirlos con los de la nueva.

Por último, el método de paso de parámetros por la pila es el más utilizado por los lenguajes de programación. Cuenta con las ventajas del sistema de variables globales, pero no tiene tantos inconvenientes. Su funcionamiento es el siguiente:

El programa guarda los parámetros de la subrutina en la pila mediante instrucciones PUSH.

- A continuación llama a la subrutina.
- La subrutina accede a los parámetros situados en la pila.
- La subrutina coloca sus resultados en la pila, sobre los parámetros de entrada y retorna.
- El programa principal recoge los resultados de la pila mediante instrucciones POP.

Los dos primeros pasos son simples y no precisan explicación adicional. El problema aparece en el tercer paso, donde la subrutina debe acceder a los parámetros que se han situado en la pila. Para ello no puede realizar un POP, puesto que en la cima de la pila se encuentra la dirección de retorno depositada por la instrucción CALL. Además, el extraer los datos de la pila a registros convertiría a este sistema en el mismo que el paso por registros, pero con el inútil trabajo de guardar los datos en la pila para luego sacarlos.

La forma correcta de acceder a los parámetros es, por tanto, utilizar el modo de direccionamiento relativo a base utilizando para ello el registro BP. Puesto que dicho modo de direccionamiento utiliza como segmento por defecto el de pila, hace innecesaria la utilización de prefijos de segmento para acceder a los parámetros. Por tanto, lo primero que debe hacer la subrutina es colocar el registro BP apuntando a los parámetros situados en la pila. El código que realiza esta operación es idéntico en todos los lenguajes de programación, recibiendo el nombre de "prefijo de subrutina", y se compone de dos instrucciones :

```
PUSH BP
MOV BP,SP
```

La primera de ellas, PUSH BP, guarda el contenido del registro BP para restaurar su valor original antes de retornar al programa que realizó la llamada. La razón de esto es permitir que una subrutina pueda llamar a su vez a otras subrutinas sin peligro. Si una rutina modificase el registro BP, al retornar la rutina que la llamó no sería capaz de acceder a sus parámetros, provocando el consiguiente fallo en el programa.

La siguiente instrucción, MOV BP,SP, coloca el registro BP apuntando a la cima de la pila. Como SP apunta siempre al último elemento introducido en la pila, queda claro que [BP+0] apunta al valor del registro BP recién almacenado en la misma. De la misma forma, y suponiendo que la rutina es de tipo NEAR, [BP+2] apuntará a la dirección de retorno almacenada por la instrucción CALL y a partir de [BP+4] se encontrarán los parámetros

guardados en la pila. Estos parámetros aparecen en orden inverso al que se introdujeron, es decir [BP+4] es el último parámetro, [BP+6] el penúltimo y así sucesivamente. En caso de que la rutina sea de tipo FAR, el contenido de la pila es algo diferente, pues un CALL de tipo FAR almacena en la pila tanto la dirección de retorno como el segmento. Por tanto, [BP+4] contendrá el segmento de retorno y los parámetros comenzarán a partir de [BP+6].

El ejemplo anterior, con este sistema, quedaría:

SumaResta	PROC		Prefijo de subrutina
	PUSH BP		
	MOV BP,SP		
	MOV AX,[BP+4]		;Lee parámetro 1
	ADD AX,[BP+6]		;Suma al segundo
	MOV BX,[BP+4]		;Lee parámetro 2
	SUB BX,[BP+6]		;Resta el segundo
	MOV [BP+6],AX		;Deja resultado sobre parámetro 1
	MOV [BP+4],BX		;Deja resultado 2 sobre parámetro 2
	POP BP		
	RET		;Retorna
	ENDP		
programa:	MOV AX,0A784h		
	PUSH AX		;Guarda primer parámetro en la pila
	MOV AX,1273h		
	PUSH AX		;Guarda el segundo
	CALL SumaResta		;Llamada
	POP BX		;Recupera resultado2 (diferencia)
	POP AX		;Recupera resultado1 (suma)

En este ejemplo se da la coincidencia que coincide el número de parámetros con el de resultados, lo que ha simplificado el proceso de retorno de resultados. Puesto que tras la llamada a la subrutina la pila debe quedar tal y como estaba antes de la llamada, en caso de que no coincida el número de parámetros de entrada con el de resultados habrá que ajustar de alguna forma la pila eliminando los datos sobrantes. Para ello se pueden utilizar dos sistemas:

1- La subrutina coloca sus resultados sobre los primeros parámetros de llamada y a continuación realiza una instrucción POP BP seguida de un

RET 2^*n , siendo n el número de parámetros de entrada menos el de resultados. Tras esto, la subrutina retornará y bastará con sacar de la pila mediante POP los resultados.

2- La subrutina coloca sus resultados sobre los primeros parámetros de llamada, realiza un POP BP y retorna. El programa principal se encarga de dejar la pila como estaba con la instrucción ADD SP, 2^*n , siendo n el número de parámetros de entrada menos el número de resultados. A continuación recupera los resultados mediante instrucciones POP.

Simplificando el ejemplo anterior para que sólo devuelva la suma de ambos parámetros, el ejemplo quedaría así para el primer método:

SumaResta	PROC		Prefijo de subrutina
	PUSH BP		
	MOV BP,SP		
	MOV AX,[BP+4]		;Lee parámetro 1
	ADD AX,[BP+6]		;Suma al segundo
	MOV [BP+6],AX		;Deja resultado sobre parámetro 1
	POP BP		
	RET 2		;Retorna descartando un parámetro
SumaResta	ENDP		
programa:	MOV AX,0A784h		
	PUSH AX		;Guarda primer parámetro en la pila
	MOV AX,1273h		
	PUSH AX		;Guarda el segundo
	CALL SumaResta		;Llamada
	POP AX		;Recupera resultado (suma)

Y por el segundo método quedaría de la siguiente forma:

SumaResta	PROC		Prefijo de subrutina
	PUSH BP		
	MOV BP,SP		
	MOV AX,[BP+4]		;Lee parámetro 1
	ADD AX,[BP+6]		;Suma al segundo
	MOV [BP+6],AX		;Deja resultado sobre parámetro 1
	POP BP		
	RET		;Retorna sin descartar parámetros

SumaResta	ENDP	
programa :	MOV AX,0A784h PUSH AX MOV AX,1273h PUSH AX CALL SumaResta ADD SP,2 POP AX	;Guarda primer parámetro en la pila ;Guarda el segundo ;Llamada ;Descarta un parámetro ;Recupera resultado (suma)

Estos dos sistemas tienen sus ventajas e inconvenientes. El primero es más rápido, pues la misma instrucción que retorna limpia la pila, pero la segunda es más flexible, porque permite subrutinas con un número variable de parámetros y la posibilidad de ignorar los resultados.

El paso de parámetros variable se realiza guardando en la pila el número de parámetros deseado, y a continuación se guarda en número de parámetros. La subrutina puede conocer el número de parámetros que se le han pasado mediante el parámetro en [BP+4] (BP+6 en rutinas FAR) y acceder a ellos a partir de [BP+6]. El problema ocurre a la hora de retornar, pues el parámetro de la instrucción RET es un valor constante. Este problema se soluciona mediante el segundo sistema, pues la instrucción ADD SP es particular para cada llamada a la subrutina, y puede variar de unas a otras.

La segunda ventaja, ignorar el resultado, se realiza con el segundo método realizando tras la llamada a la subrutina una instrucción ADD SP,2*n, siendo n esta vez el número de parámetros que se pasaron. De esta forma se eliminan de la pila tanto los parámetros como los resultados.

Además de los tres métodos de paso por parámetros expuestos, es posible utilizar combinaciones de éstos. Por ejemplo, se podrían pasar los parámetros de entrada en la pila, pero devolver los resultados en registros o en posiciones fijas de memoria. Quedará pues a elección del programador el utilizar el método que crea más conveniente.

VARIABLES LOCALES

En algunas ocasiones, una subrutina necesita algún lugar donde almacenar un resultado intermedio de los cálculos que realiza. Aunque habitualmente se utiliza para esto un registro del procesador o la pila, en ocasiones sería conveniente disponer de una variable temporal que existiese únicamente durante la ejecución de la subrutina y que desapareciese al finalizar ésta.

A este tipo de variables se les denomina "variables locales", y son muy importantes en programación, pues permiten implementar mecanismos como la recursividad (una rutina que se llama a sí misma para hacer parte de su trabajo) y la reentrada, (permitir que varios procesos simultáneos llamen a una misma rutina a la vez).

El sistema habitual de variables locales utiliza la pila para almacenarlas, por lo que suele ir ligado también al paso de parámetros en la pila. El proceso de utilización de variables locales en una subrutina sigue tres fases:

1- Creación de espacio para las variables al principio de la subrutina. Esto se lleva a cabo restándole al puntero de pila SP tantos bytes como hagan falta para almacenar las variables locales. Según esto, el comienzo de la subrutina quedaría:

```
PUSH BP  
MOV BP,SP  
SUB SP,n ;n es el espacio necesario para las variables locales
```

2- Acceso a las variables locales. Se aprovecha que las variables locales se han creado a partir de la dirección apuntada por BP, por lo que puede accederse a ellas mediante direccionamientos del tipo [BP-n], estando situada la primera variable local en [BP-2].

3- Al finalizar la subrutina se destruye el espacio reservado para estas variables mediante las siguientes instrucciones:

```
ADD SP,n ;Deja Sp como estaba  
POP BP ;Restaura BP  
RET ;Retorna
```

Segmentos de ensamblador

Tanto el código como los datos se agrupan en bloques conocidos como segmentos. A pesar de que tienen el mismo nombre, los segmentos de un programa ensamblador no son equivalentes a los segmentos en que divide la memoria el procesador 8086. Una diferencia notable es el hecho de que un segmento ensamblador puede comenzar en cualquier posición de memoria mientras que, como ya se vio, los segmentos de memoria comienzan siempre en posiciones múltiplo de 16.

Aunque no son equivalentes, segmentos ensamblador y de memoria tienen varias relaciones. La primera de ellas es que un segmento no puede ocupar, entre código y datos, más de 64 Kilobytes de memoria. La segunda similitud es que el nombre de un segmento puede utilizarse como valor

simbólico del segmento de memoria en que se encuentra éste.

Código: SEGMENT
MOV AX,Código

;Coloca en AX el valor del segmento donde se encuentra Código.

Código: ENDS

Sin embargo, hay que tener en cuenta que esta forma de utilizar el nombre del segmento de código sólo es posible en programas de tipo .EXE. En programas .COM esto no es un problema, pues cuentan con un único segmento que se puede cargar mediante MOV AX,CS.

La forma de definir un segmento es mediante una pareja de directivas que marcan su comienzo y su final. Estas directivas son SEGMENT para el principio y ENDS para el final. Ambas van precedidas por el nombre que se le da al segmento y, en el caso de la primera, va seguida de una serie de palabras que indican algunas características del segmento.

Las características o atributos que se aplican a la directiva SEGMENT son la alineación, la combinación y la clase, las cuales pueden darse en cualquier orden. En caso de que se omitan algunos de estos cuatro tipos de atributos, el ensamblador tomará para ellos sus valores por defecto.

El primero de estos atributos, el de alineación, se utiliza para que el segmento comience en una dirección de memoria que sea múltiplo de un valor determinado. Sus posibles valores son los siguientes:

Alineación	Múltiplo de
BYTE	1
WORD	2
DWORD	4
PARA	16
PAGE	256
MEMPAGE	4096

El valor de alineación por defecto es PARA, el cual es el utilizado en la mayoría de las ocasiones, puesto que al utilizar una dirección múltiplo de 16 como comienzo hace que el segmento ensamblador coincida exactamente sobre un segmento de memoria, formándose así una equivalencia entre ambos.

El atributo de combinación, por su parte, especifica cómo debe combinarse el segmento con otros que tengan el mismo nombre y que se encuentren en otros módulos del programa. En caso de que no se especifique, se toma el valor PRIVATE por defecto.

Combinación	Forma de combinación
PRIVATE	No se combina con otros segmentos del mismo nombre
PUBLIC	Se combina con los segmentos del mismo nombre en un único segmento
STACK	El segmento se combina con otros del mismo nombre para formar la pila del programa
COMMON	Coloca el segmento en las mismas posiciones de memoria que otros con el mismo nombre
AT xxxx	El segmento hace referencia al segmento físico indicado por el valor numérico xxxx. Este tipo de segmentos se utiliza para acceder a memoria fuera del programa, como vectores de interrupción o datos de la BIOS
MEMORY	El segmento se colocará al final del programa, permitiendo utilizar la memoria que hay más allá del mismo. Sólo puede haber un segmento de este tipo, y si hay varios, el resto se tratan como COMMON.
VIRTUAL	Define un tipo de segmento común a varios módulos, pero que se define dentro de otro.

Por último, el atributo de clase especifica un nombre de clase entre comillas para el segmento que el enlazador utilizará a la hora de ordenar los segmentos. Esta ordenación provocará que segmentos con el mismo atributo de clase se coloquen agrupados unos tras otros. Esta característica se utiliza para conseguir que todos los segmentos de un mismo tipo (datos, código, pila) se coloquen juntos en memoria cuando se cargue el programa.

Ejemplos:

datos SEGMENT PARA PUBLIC ;Datos ;Define un segmento que se combinará con otros de clase "Datos"

pila SEGMENT PARA STACK "STACK" ;Define la pila del programa

video SEGMENT AT 0B800h ;Define un segmento para direccionar la memoria de video en modo texto.

Para acceder a los datos de un segmento es preciso colocar algún registro de segmento apuntando hacia él. Además, es necesario informar al ensamblador de a qué segmentos apunta cada uno de los registros de segmento, de forma que éste pueda crear los prefijos de segmento adecuados.

La directiva ASSUME puede ir seguida de uno o varios nombres de registros de segmento, seguidos de dos puntos y el nombre del segmento hacia el que están apuntando. Un ejemplo sería:

```
ASSUME CS:Código, DS :Datos, SS :Pila
```

En caso de que se desee anular la asociación entre un registro de segmento y un segmento, se puede utilizar el segmento NOTHING para indicar este hecho:

```
ASSUME DS :Datos
```

:Informa que DS apunta al segmento Datos:

```
ASSUME DS :NOTHING
```

:Informa que DS ya no apunta a Datos

Un detalle importante es que ASSUME es una directiva y, como tal, no genera código que cargue en los registros de segmento el valor que indica. Es, por tanto, labor del programador el iniciar los registros de segmento de forma adecuada e informar al ensamblador mediante ASSUME de los cambios que haga en ellos. Una típica situación errónea es la siguiente:

```
Datos SEGMENT  
va DW ?  
Datos ENDS
```

```
Código SEGMENT  
MOV AX,va  
Código ENDS
```

Este ejemplo da un error de ensamblado, puesto que el ensamblador no conoce qué registro de segmento está apuntando al segmento Datos. Una simple línea arregla este error:

```
Código SEGMENT  
ASSUME DS :Datos  
MOV AX,VA  
Código ENDS
```

Efectivamente, este cambio evita el error de ensamblado, pero el programa sigue sin funcionar. La razón es que el registro DS no apunta al segmento Datos, pues no se han incluido las instrucciones para ello. El siguiente programa sería el correcto:

```
Código :SEGMENT
```

```
MOV AX,Datos
```

```
MOV DS,AX
```

```
ASSUME DS :Datos
```

```
MOV AX,va
```

```
Código ENDS
```

También es imprescindible colocar al principio de todos los segmentos del código la directiva ASSUME, asociando el registro CS al segmento actual:

```
Código SEGMENT
```

```
ASSUME CS :Código
```

Y en el caso de programas EXE se añadirá una directiva ASSUME asociando el registro SS al segmento de pila. Hay que indicar, sin embargo, que en los programas EXE es el propio MS-DOS el que se encarga de inicializar los registros SS y SP al segmento de pila definido por el programa, por lo que no habrá que preocuparse de cambiarlos.

Los segmentos, a su vez, se pueden agrupar entre sí para formar segmentos mayores. Esta operación se realiza mediante la directiva GROUP, que va precedida por el nombre del nuevo segmento formado por la combinación de los segmentos cuyos nombres se indican separados por comas a continuación de la directiva GROUP. El tamaño conjunto de los segmentos de un grupo no debe superar los 64Kb. El nombre de este grupo puede utilizarse como si fuese un nombre de segmento.

```
Datos SEGMENT  
va dw ?  
Datos ENDS
```

```
Datos1 SEGMENT  
vb dw ?  
Datos1 SEGMENT
```

```
grupo GROUP Datos,Datos1
```

```
Código SEGMENT
```

```
ASSUME CS :código, DS :grupo
```

```
MOV AX,grupo
```

```
MOV DS,AX
```

A la hora de utilizar grupos hay una irregularidad que es preciso tener en cuenta. Mientras que las referencias a las variables del grupo funcionan de forma correcta, la función OFFSET no lo hace así, y devuelve el desplazamiento de cada variable del grupo respecto a sus segmentos originales, no al grupo. Por ejemplo, en la definición anterior de segmentos:

```
MOV va,AX  
MOV AX,vb
```

funcionan de forma correcta, pero:

```
MOV AX,OFFSET va  
MOV AX,OFFSET vb
```

devuelven ambas el mismo valor, es decir, un desplazamiento 0, puesto que OFFSET ya lo ha calculado respecto al segmento Datos; y OFFSET vb respecto al segmento Datos1, en lugar de hacerlo relativo al grupo.

Para solventar esta situación, lo que se hace es preceder el nombre de la variable con el nombre del grupo, obteniéndose así los desplazamientos correctos:

```
MOV AX,OFFSET grupo :va  
MOV AX,OFFSET grupo :vb
```

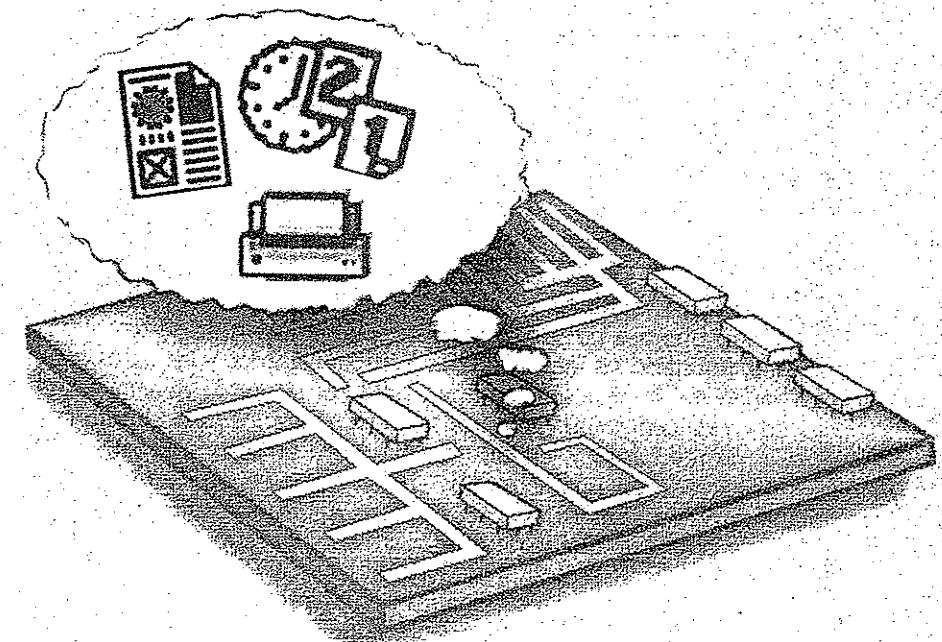
Estructura de un programa tipo .EXE

Hasta ahora, los programas completos que han aparecido eran de tipo COM y no se había entrado en detalles de la estructura de un programa EXE. Es por ello que los ejemplos que muestran la utilización de los segmentos muestran únicamente fragmentos y no un programa completo, puesto que el manejo de los segmentos es exclusivo de los programas de tipo EXE.

El siguiente listado muestra una estructura de programa de tipo EXE; y constituye un buen punto de partida para realizar un programa o para comprobar los ejemplos de este libro.

Salvo en raras ocasiones, el orden de los segmentos no es importante. Se ha escogido, pues, una distribución que suele ser la habitual, pero que no tiene ninguna razón especial:

```
;Define un segmento de pila de 200 words.  
Pila SEGMENT PARA STACK "STACK"  
DW 200 dup (0)  
Pila ENDS  
  
Datos SEGMENT  
;Aquí se colocarían las variables que necesita el programa  
Datos ENDS  
  
Codigo SEGMENT  
ASSUME CS :Codigo, DS :Datos, SS :pila  
  
;A continuación vendrían las subrutinas.  
rutina PROC  
  
rutina ENDP  
  
;Y por último el código del programa  
inicio : MOV AX, Datos ;Al principio del programa ajusta  
DS para que coincida con lo especificado por ASSUME  
MOV DS,AX  
;Resto de instrucciones del programa  
MOV AX,4c00h ;Finaliza el programa  
INT 21h  
codigo ENDS  
END inicio
```



Funciones de la BIOS y el DOS



os programas no sólo se limitan a realizar cálculos con datos contenidos en memoria, sino que también reciben información de otras fuentes, como puede ser un teclado o un disco duro, y envían también información a dispositivos como una pantalla, un puerto serie o una impresora. Aunque el código máquina del 8086 provee de mecanismos para manejar periféricos, a menudo el control de éstos de forma directa es muy complejo y laborioso. Es por ello que en el IBM PC existen ya una serie de rutinas precargadas que se encargan de realizar todo este proceso, liberando al programador de esta tarea y simplificando enormemente la creación de programas.

Las dos fuentes de rutinas son, en primer lugar, la ROM BIOS, en cuyo interior se encuentran toda una serie de rutinas básicas para el manejo de periféricos. La segunda de estas fuentes es el sistema operativo MS-DOS, el cual proporciona unos servicios más completos que los ofrecidos por la BIOS. Por ejemplo, mientras que la BIOS tiene funciones para leer y escribir datos en un disco duro de forma directa, sector a sector, el MS-DOS posee rutinas más complejas que permiten trabajar con ficheros y directorios. Por tanto la técnica habitual será utilizar las funciones del MS-DOS, debiendo recurrir a la BIOS únicamente para aquellas situaciones en que el MS-DOS no disponga de una función similar o cuando se desee controlar los periféricos de una forma más directa.

La forma de acceder a estas rutinas es a través de interrupciones software. Para ello, se colocan en unos registros determinados los parámetros que necesita la subrutina y se llama a la interrupción correspondiente. Una vez completado el proceso, la interrupción retornará devolviendo el resultado en registros. Los registros utilizados y los datos que contienen varían de una función a otras, pero como norma general estas funciones no modifican ningún registro, salvo los utilizados para el paso de parámetros y de resultados y, en caso de que haya un error, devuelven la bandera de acarreo activada. Además, en los casos en que una misma interrupción ofrece varias rutinas, el registro AH contiene un número identificativo de la función que se desea utilizar.

Tratar todos los servicios proporcionados por las interrupciones sería muy extenso, por lo que se tratarán únicamente aquellos servicios que tengan una utilidad práctica y no se hayan quedado anticuados.

La ROM BIOS

El primer gran conjunto de rutinas para el manejo de periféricos se encuentra contenido en la ROM BIOS, la misma que se encarga de chequear el ordenador y de iniciar la carga del sistema operativo. Para su trabajo, la BIOS utiliza las interrupciones desde la 10h hasta la 1Fh. Algunas de estas

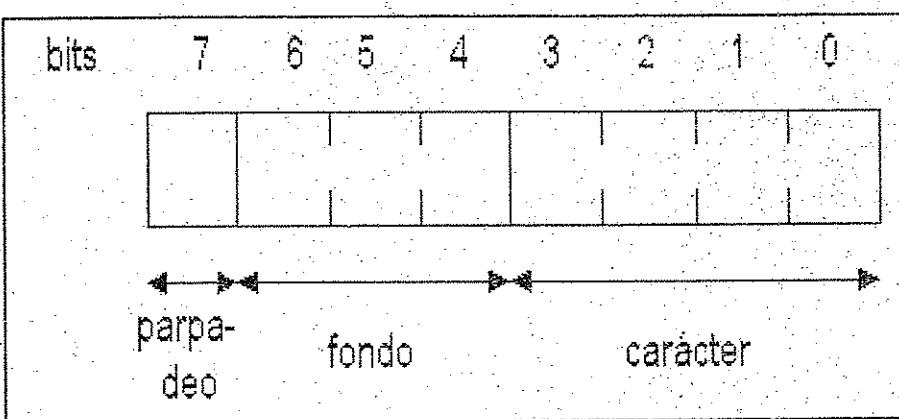
interrupciones realizan funciones de un dispositivo concreto, como la pantalla o la unidad de disco, mientras que otras proveen información acerca del sistema o apuntan a tablas con datos.

INT 10h: PANTALLA

La interrupción 10h contiene una serie de rutinas para el manejo de la pantalla tanto en modo de texto como en modo gráfico. Aunque originariamente la ROM BIOS sólo incluía soporte para las tarjetas de vídeo CGA y MGA, las modernas EGA, VGA y SuperVGA incluyen una ROM propia que añade funciones a esta interrupción para manejar los nuevos modos gráficos con mayor resolución y número de colores.

En modo texto, la pantalla se comporta como una matriz que contiene en cada una de sus posiciones una letra. Cada una de estas posiciones lleva asociado además un atributo, que indica tanto el color de la letra como el color de fondo. Cada una de estas posiciones de pantalla ocupa, pues, dos bytes, uno para almacenar el código ASCII de la letra y otro para almacenar el atributo de color.

El atributo, por su parte, se divide en tres partes, una que contiene el color del carácter (4 bits menos significativos) y otra con el de fondo (3 siguientes) y, por último, el bit más alto controla el parpadeo del carácter, de forma que si se coloca a 1 la letra aparecerá y desaparecerá intermitentemente.



Además de los caracteres, el modo texto cuenta con el cursor, un elemento que parpadea y que muestra una posición en pantalla. La BIOS también ofrece funciones para posicionar el cursor, leer dónde se encuentra o variar su forma.

En modo gráfico, la pantalla se comporta también como una matriz de elementos, cada uno de los cuales corresponde a un punto de pantalla. Cada una de estas posiciones contiene el color del punto, un valor que depende del

modo de vídeo que se esté utilizando y que puede variar desde dos colores hasta más de 16 millones, según la tarjeta gráfica que se utilice. Sin embargo, las funciones de la BIOS sólo soportan de forma estándar hasta la tarjeta gráfica VGA con un máximo de 256 colores, por lo que las resoluciones con un mayor número de tonalidades deben controlarse bien a través de nuevas funciones aportadas por la BIOS de la tarjeta gráfica o, más habitualmente, manejando directamente la memoria de vídeo de la tarjeta.

En algunos modos, la tarjeta gráfica dispone de varias páginas, es decir, varias pantallas en las que es posible escribir. Como cabría suponer, en un momento dado sólo puede estar visible una de estas páginas, pero es posible cambiar la que se muestra en el monitor con una simple llamada a una de los servicios de vídeo de la BIOS. Las múltiples páginas permiten, por ejemplo, preparar una página sin que el usuario vea el proceso de forma que, una vez construida, al hacerla visible da la impresión de que aparece de forma instantánea.

Para utilizar las funciones de vídeo de la BIOS se coloca en primer lugar el número de función en el registro AH. En caso de que se necesiten parámetros adicionales, éstos se situarán en otros registros. Si se produjera un error durante la ejecución de la función, la bandera de acarreo se devolverá activada para indicarlo.

A continuación se tratan las funciones más habituales de la BIOS para el manejo de la pantalla, así como los parámetros que necesitan:

TABLA.3

Modo (Valor AL)	Tipo	Resolución	Colores	Páginas	Tarjeta gráfica
0h	Texto	40x25	16 (Grises)	8	CGA
1h	Texto	40x25	16	8	CGA
2h	Texto	80x25	16 (Grises)	8	CGA
3h	Texto	80x25	16	8	CGA
4h	Gráfico	320x200	4	1	CGA
5h	Gráfico	320x200	4 (Grises)	1	CGA
6h	Gráfico	640x200	2	1	CGA
7h	Texto	80x25	2	8	Monocromo
0Dh	Gráfico	320x200	16	8	EGA
0Eh	Gráfico	640x200	16	4	EGA
0Fh	Gráfico	640x350	2	2	EGA
10h	Gráfico	640x350	16	2	EGA
11h	Gráfico	640x480	2	1	VGA
12h	Gráfico	640x480	16	1	VGA
13h	Gráfico	320x200	256	1	VGA

- Función 0: Cambia el modo de vídeo.

Mediante este servicio se puede cambiar entre diversas configuraciones de pantalla en modo texto o en modo gráfico. En AL se coloca el modo de vídeo al que se desea cambiar, que debe ser uno de los especificados en la siguiente tabla: [Tabla.3]

- Función 1h: Cambiar la forma del cursor.

La forma del cursor se define indicando la línea del carácter en que empieza el cursor en CH y la línea en que termina en CL. El rango de valores va de 0 a 31, pero en la tarjeta CGA sólo se utiliza de 0 a 7, y en la EGA y VGA de 0 a 13. Si se colocan ambas líneas al valor 31, el efecto es que el cursor desaparece.

- Función 2h: Posicionar el cursor.

La posición del cursor viene determinada por la página en la que se encuentra, así como la línea y la columna del carácter sobre el que se sitúa. Estos parámetros se le proporcionan a la función en los registros BH (página), DH (fila) y DL (columna). Existe un cursor independiente para cada una de las páginas.

- Función 3h: Lee posición del cursor.

La función 3 devuelve tanto la posición del cursor (fila, columna) como su forma actual. No requiere parámetros de entrada y, al retorno de la interrupción 10h, los registros DH y DL contendrán la fila y la columna del cursor, respectivamente, mientras que CH y CL contienen la línea inicial y final que definen la forma del cursor, utilizando el mismo formato que el usado por el servicio 1.

- Función 5h: Seleccionar la página activa.

Selecciona cuál de las páginas disponibles se visualiza en el monitor. El único parámetro que necesita es el número de página en el registro AL, valor que varía desde 0 hasta un valor una unidad inferior al número de páginas del modo de vídeo actual.

- Función 6h: Desplazamiento de texto hacia arriba.

Esta función desplaza un número determinado de líneas hacia arriba un rectángulo definido dentro de la pantalla de texto. Los parámetros que necesita son los siguientes:

AL: Número de líneas a subir. Si vale 0 se borra el recuadro.

CH,CL: Fila y columna de la esquina superior izquierda del rectángulo a desplazar.

DH,DL: Fila y columna de la esquina inferior izquierda del rectángulo a desplazar.

BH: Atributo a utilizar en las líneas inferiores que quedan en blanco.

- Función 7h: Desplazamiento de texto hacia abajo.

Se trata de una función similar a la anterior y que recibe los mismos parámetros, sólo que en este caso el desplazamiento del texto se realiza hacia abajo.

- Función 8h: Leer carácter y atributo.

Mediante esta función es posible leer el carácter y el atributo de la posición donde se encuentra situado el cursor. Para ello necesita conocer únicamente la página de texto a la que se refiere, y tras la llamada a la interrupción se recibe en AL el carácter y en AH el atributo.

- Función 9h: Escribir carácter y atributo.

Para escribir un carácter indicando también el atributo debe llamarse a esta función, colocando en BH el número de página, en AL el carácter a escribir, en BL el atributo y en CX el número de veces qué hay que escribir dicho carácter.

- Función 0Ah: Escribir carácter.

Esta función escribe un carácter en la posición actual del cursor, pero conservando el atributo que ya contiene. Requiere en BH el número de la página y en AL el carácter a escribir.

- Función 0Bh: Cambiar paleta de color.

Esta función se utiliza en los modos gráficos 4 y 5 para cambiar entre las dos posibles paletas de color de la CGA. Para ello hay que colocar en BH el número de la paleta de color a cambiar, y en BL el color que se asigna. Para BH sólo hay dos posibilidades: BH=0, que cambia el color de fondo y el del marco al indicado en BL, y BH=1, que permite cambiar entre las dos paletas de cuatro colores de la CGA (BL=0 y BL=1).

- Función 0Ch: Escribir un pixel.

La primera de las funciones para modo gráfico permite colocar un pixel con un color determinado, especificando para ello la página en la que se

encuentra en BH, la coordenada X en CX y la coordenada Y en DX. El color se indica en AL.

- Función 0Dh: Leer un pixel.

Esta función es la inversa de la anterior. Lee el color que tiene un pixel determinado por el número de página en que se encuentra (BH), su coordenada X (CX) y su coordenada Y (DX). El valor del color se devuelve en el registro AL.

- Función 0Eh: Escribir un carácter como TTY.

Esta función se utiliza por programas que únicamente quieren sacar texto por pantalla y que no necesitan el color. Cuando se escribe una letra a través de esta función, el cursor avanza de forma automática al siguiente carácter, cambiando de línea si es preciso e incluso haciendo un scroll de la pantalla cuando se llega a la parte inferior derecha de ésta. Hay, además, cuatro caracteres especiales: el 7h, que genera un pitido; el 8h, que borra el último carácter; el 0Ah, que salta una línea y el 0Dh que vuelve al principio de la línea. Para utilizarlo, se coloca en AL el carácter que se desea escribir y en BL el color del carácter (sólo en modo gráfico).

- Función 0Fh: Obtener el modo de vídeo.

El servicio 0Fh de vídeo no necesita parámetros, y devuelve información acerca del modo de vídeo en que se encuentra la pantalla. Los datos devueltos son:

AL: Modo de vídeo actual.

AH: Número de caracteres por línea.

BH: Número de página que se está visualizando.

INT 11h Y 12h: CHEQUEO DE LA CONFIGURACIÓN Y MEMORIA

Estos dos servicios devuelven información acerca de los dispositivos conectados en el sistema y del total de la memoria disponible. Dado que los datos que devuelve no han variado desde el primer PC que salió al mercado, algunos de ellos carecerán de sentido actualmente.

La interrupción 11h devuelve en AX la configuración del sistema, utilizando los bits como se muestra en la siguiente tabla:

Bits	Contenido
0	1: Hay unidad de disco, 0: No hay
2-3	Memoria RAM en la placa base del sistema. 00:16Kb, 01:32Kb, 10:48Kb, 11:64Kb
4-5	Modo de vídeo inicial de la pantalla. 01: 40x25, 10: 80x25 color, 11: 80x25 mono.
6-7	número de unidades de disco. 00: 1, 01: 2, 10: 3, 11: 4
9-11	número de puertos serie
12	Puerto de Joystick (0: No, 1: Sí)
14-15	número de puertos paralelo

La interrupción 12h, por su parte, devuelve en AX el número de Kb de memoria convencional instalados en el sistema. Puesto que hoy en día todos los ordenadores superan los 640Kb máximos del primer PC, esta función siempre devuelve el valor 640.

INT 13h: MANEJO DE DISCOS

La BIOS ofrece a través de la interrupción 13h una serie de rutinas que se utilizan para controlar las unidades de disquete y de disco duro a bajo nivel. Al igual que sucedía con las funciones de pantalla, todos los servicios de disco necesitan que se especifique el número de la función que se desea realizar en el registro AH. Además, todas estas funciones utilizan la bandera de acarreo para indicar que se ha producido un error durante el transcurso de la operación, y en el registro AH se encuentra un valor numérico indicativo del error que ha sucedido. Sin embargo, hay que tener en cuenta que a veces los errores se ocasionan debido a que la unidad de disco estaba apagada y el disco ha tardado más tiempo del normal en alcanzar la velocidad normal de giro. Por ello es conveniente, en caso de que se produzca un error al acceder al disco, reintentar la operación un número determinado de veces (tres o cuatro normalmente) antes de dar por fallida la operación.

La siguiente tabla resume los códigos de error devueltos por la interrupción 13h en el registro AH:

AH	Error
0h	No hubo error
2h	Marca de identificación de sector errónea o inexistente
3h	Disco protegido contra escritura
4h	Sector no encontrado
8h	Error de DMA
9h	Error de límite de 64Kb en DMA
10h	Error de CRC (de datos)
20h	Error en el controlador de disco
40h	Error en el posicionamiento sobre la pista
80h	La unidad de disco no responde

Algunas de las funciones de manejo de disco requieren que se les proporcione el número de la unidad sobre la que trabajarán. Para unidades de disco estos números son: 0 para la A y 1 para la B. Los discos duros, por su parte, comienzan a partir de 80h para la primera unidad de disco duro (C:), 81h para el segundo disco duro, etc... Tenga en cuenta que el acceso se realiza a nivel de disco duro, no de particiones, por lo que si tiene varias particiones en una misma unidad no se considerarán como unidades de disco diferentes.

- Función 0: Reinicialización del disco.

Esta función reinicializa tanto el controlador del disco como la unidad. Debe realizarse cuando se produzca un error en un acceso a disco antes de volver a intentar la operación.

- Función 1: Obtener el estado del disco.

Esta función no realiza acción ninguna. Simplemente comprueba el estado del controlador y la unidad de disco y, en caso de detectar un error, devuelve en el registro AH uno de los códigos indicados en la tabla anterior.

- Función 2: Leer sectores.

La lectura de datos desde el disco se lleva a cabo a través de esta función, la cual puede leer uno o varios de los sectores contenidos dentro de una pista determinada de la unidad. Para ello necesita que se le proporcionen los siguientes datos:

Número de la unidad en DL.

Número de cara en DH.

Número de pista en CH.

Primer sector a leer en CL.

Número de sectores a leer en AL.

Dirección en la que guardar los datos en ES:BX.

Tras la operación, si no hubo error, AL contendrá el número de sectores leídos.

- Función 3: Grabar sectores.

Esta función recibe los mismos parámetros que la anterior, sólo que en este caso ES:BX apunta hacia la zona de memoria en la que se encuentran los datos a guardar. Tras la llamada, si no hubo error, AL contiene el número de sectores grabados.

- Función 4: Verificar sectores.

La verificación de sectores comprueba que éstos se encuentran grabados correctamente. Resulta útil para, tras una grabación, comprobar que los datos se han almacenado correctamente.

Esta función recibe los mismos parámetros que la función 2, pero no es necesario especificar una dirección donde almacenar los datos, pues éstos realmente no se leen.

- Función 5: Formatear una pista.

Esta función trabaja de forma similar a la de lectura, pero en este caso la acción se lleva siempre sobre una pista completa; por lo que el valor del sector inicial contenido en CL no se utiliza. ES:BX, por su parte, apunta a una zona de datos que contiene información acerca de cómo se formateará la pista.

Esta especificación de formato consta de cuatro bytes para cada uno de los sectores que contendrá la pista (valor que se indica en el registro AL). Estos cuatro bytes contienen, en este orden, el número de pista, el

número de cabeza, el número del sector y el tamaño del sector. Bajo MS-DOS, el tamaño de sector tiene siempre el valor 2.

INT 14h: PUERTO SERIE

Los puertos serie se utilizan para la conexión de dispositivos como modems, impresoras o ratones. A través de ellos es posible enviar y recibir bytes, pero al contrario de qué sucede con los discos, donde se transfieren bloques de datos, la transmisión y recepción se realiza byte a byte.

En total, la interrupción 14h ofrece cuatro funciones para el manejo de los puertos serie, que son las siguientes:

- Función 0: Inicializar puerto serie.

Esta función ajusta los parámetros de velocidad, paridad, bits de parada y tamaño de los datos que se transmiten. Antes de enviar y recibir datos por un puerto serie es necesario llamar a esta función, de forma que los parámetros coincidan con los del aparato situado al otro extremo del cable que sale del puerto serie.

Estos cuatro datos se colocan en el registro AH, utilizando los bits siguientes:

Bits	Parámetro	Valores
0-1	Longitud del carácter	10: 7 bits, 11: 8 bits
2	Bits de parada	0: uno, 1: dos
3-4	Tipo de paridad	00: ninguna, 01: impar, 10: ninguna, 11: par
5-7	Velocidad en baudios	000: 110, 001: 150, 010: 300, 011: 600, 100: 1200, 101: 2400, 110: 4800, 111: 9600

- Función 1: Enviar un carácter.

Esta función intenta transmitir por el puerto serie, cuyo número se indica en DX, el valor contenido en AL. Si se envía el carácter de forma correcta, entonces AH retorna con el valor 0. En caso de fallo se devuelve un código de error en AH, cuyo valor es el indicado para este registro en la tabla de la función 3 de esta misma interrupción.

- Función 2: Recibir un carácter.

Se trata de la función inversa a la anterior e intenta recibir un carácter del puerto serie, indicado en el registro DX. En caso de éxito, AH retorna con el valor 0 y AL contiene el carácter recibido. Los códigos de error son los indicados en la tabla del servicio 3.

- Función 3: Obtener estado del puerto serie.

Recibe como único parámetro el número del puerto serie en DX, y devuelve por una parte en AH un código que indica el estado de la

línea, y en AL otro de estado del módem. Las siguientes tablas resumen el significado de cada uno de los bits retornados en estos registros:

Registro AH (estado de la línea)

Bit	Significado cuando está a 1
0	Datos preparados
1	Desbordamiento en recepción
2	Error de paridad
3	Error de formato de tramas
4	Error de detección de portadora
5	Buffer de transmisión vacío
6	Buffer de desplazamiento vacío
7	Tiempo máximo excedido (timeout)

Registro AL (estado del módem)

Bit	Significado cuando está a 1
0	* Listo para enviar (señal CTS)
1	* Datos preparados para enviar (señal DSR)
2	* Llamada detectada (señal RI)
3	* Portadora detectada (señal DCD)
4	Listo para enviar (señal CTS)
5	Datos preparados para enviar (señal DSR)
6	Llamada detectada (señal RI)
7	Portadora detectada (señal DCD)

Los bits del 4 al 7 muestran los parámetros actuales del módem, mientras que los del 0 al 3 son los mismos, pero toman el valor 1 si su valor ha cambiado desde la última llamada.

INT 16h: TECLADO

El teclado es el principal dispositivo de entrada, por lo que también posee su propia interrupción, a pesar de que dispone de muy pocas funciones para su manejo. Únicamente hay tres funciones (numeradas de 0 a 2) que se seleccionan, como es habitual, colocando su número en el registro AH.

- Función 0: Leer carácter.

Esta función espera hasta que se pulse una tecla y devuelve en AH y AL un par de valores que indican la tecla que se ha pulsado. AL contiene el código ASCII de la tecla pulsada, o bien 0 para teclas especiales como F1 o Inicio, que no poseen un código ASCII. Por su parte, AH contiene el código de exploración de la tecla que se ha pulsado.

- Función 1: Comprobar si hay caracteres en el *buffer* de teclado.
A través de esta función es posible averiguar si hay alguna pulsación esperando para ser procesada. En caso de que haya algún carácter esperando a ser recogido se devuelve la bandera de cero (ZF) activada, debiéndose llamar a continuación a la función 0 para recoger la tecla.

- Función 2: Lee el estado de las teclas de *Shift*.

Las teclas de *Shift* son aquellas que cuando se pulsan en combinación con otras modifican su significado, como por ejemplo la de mayúsculas o la de Control. Estas teclas toman el valor 1 si dicha tecla se encuentra pulsada o 0 si está levantada. La siguiente tabla indica qué bit corresponde a cada una de ellas:

Bit	Tecla especial
0	Mayúsculas derecha
1	Mayúsculas izquierda
2	Control
3	Alt
4	Bloq. Despl
5	Bloq. Num
6	Bloq. Mayús
7	Insert

INT 17h: IMPRESORA

Los servicios de impresora se utilizan para enviar datos a un dispositivo de impresión conectado al primer puerto paralelo. Las funciones disponibles para este propósito a través de la interrupción 17h son las siguientes:

- Función 0: Enviar un byte a la impresora.

El byte a enviar se coloca en el registro AL. Tras la llamada a la interrupción AH devuelve el estado de la impresora, según se indica en la tabla de la función 3 de esta misma interrupción.

- Función 1: Reinicializar la impresora.

Envía un código de reinicio a la impresora, de forma que ésta se recupere de una situación de error.

- Función 2: Obtener el estado de la impresora.

Devuelve en AH una serie de bits, que indican cómo se encuentra la impresora o si ha sucedido algún tipo de error con ella. Los valores posibles para cada bit son:

Bit	Significado si vale 1
0	No se pudo enviar el dato (timeout)
1-2	No utilizados
3	Error de entrada/salida

4	Impresora activa (on line)
5	Impresora sin papel
6	Señal de reconocimiento de la impresora
7	Impresora libre (0: impresora ocupada)

OTRAS INTERRUPCIONES DE LA BIOS

Además de las vistas, la BIOS dispone de otras interrupciones que realizan funciones específicas. Éstas son las que se enumeran a continuación:

- INT 5: Imprimir pantalla.

Imprime la pantalla en modo texto. Puede imprimir pantallas gráficas si se carga el programa GRAPHICS de MS-DOS.

- INT 19h: Reiniciar el equipo.

Una llamada a esta interrupción provoca que se vuelva a cargar el sistema operativo.

- INT 1Ah: Servicios de hora.

Permiten leer y cambiar el valor del temporizador interno del ordenador. Este temporizador es un valor de 32 bits que se actualiza 18.2 veces por segundo, es decir, aproximadamente una vez cada dos centésimas de segundo. Existen dos funciones asociadas a esta interrupción, que son las siguientes:

- Función 0: Leer el temporizador.

Devuelve los siguientes resultados:

Word alto del temporizador en CX.

Word bajo del temporizador en DX.

AL es distinto de 0 si el temporizador sobrepasó un día de tiempo.

- Función 1: Ajustar el temporizador.

Ajusta el temporizador al valor indicado por los registros CX (word alto) y DX (word bajo).

- INT 1Ch: Pulso del temporizador.

Esta interrupción se llama cada vez que se incrementa el temporizador, es decir, 18.2 veces por segundo. Es posible hacer que una rutina se ejecute cada cierto tiempo si se conecta a esta interrupción, siguiendo un proceso que se explicará al hablar de los programas residentes.

El MS-DOS (INT 21h)

El segundo proveedor de servicios a los programas es el sistema operativo MS-DOS, el cual ofrece un paso más de funcionalidad sobre las funciones ofrecidas por la BIOS. A lo largo de la explicación de los servicios del DOS se verá que, en ocasiones, algunas de las funciones que éste ofrece son análogas a otra que ofrece la BIOS. En estas situaciones deberá utilizarse siempre la que

ofrece el MS-DOS, puesto que seguramente ofrezca alguna funcionalidad adicional de importancia. Por ejemplo, a la hora de enviar un carácter a la pantalla, tanto el MS-DOS como la BIOS disponen de funciones para realizarlo, pero es mejor utilizar la función del MS-DOS, puesto que ofrece la ventaja adicional de que la salida en pantalla puede redirigirse a un archivo en el disco. Además, habitualmente el MS-DOS ofrece soporte a un mayor número de dispositivos que la BIOS no es capaz de manejar, como es el caso de unidades de CD-ROM o discos magneto-ópticos.

La mayoría de las funciones del MS-DOS se agrupan en una única interrupción, la 21h, a la que se llama indicando en el registro AH el número de servicio que se desea utilizar, y el resto de los parámetros que necesita en el resto de los registros del procesador. Tras su ejecución, las funciones del DOS devuelven el acarreo activo si se produjo un error durante su realización. Además, muchas de estas funciones, en caso de error, devuelven en el registro AH un código de error, indicando qué fue lo que ocurrió:

Código de error	Significado
0h	No hubo error
1h	Número de función inexistente
2h	No se encontró el fichero
3h	No se encontró el directorio
4h	No hay más handles disponibles
5h	Acceso denegado a un fichero
6h	Handle de fichero no válido
7h	Bloques de control de memoria no válidos
8h	Memoria agotada
9h	Bloque de memoria no válido
0Ah	Variables de entorno no válidas
0Bh	Formato no válido
0Ch	Código de acceso erróneo
0Dh	Dato erróneo
0Fh	Unidad de disco no válida
10h	Intento de borrar el directorio activo
11h	Dispositivo diferente
12h	No hay más ficheros

ENTRADA/SALIDA ESTÁNDAR (TECLADO/PANTALLA)

Habitualmente, durante la utilización de un programa, éste va mostrando datos en la pantalla mientras que el usuario le envía la información que necesita a través del teclado. El MS-DOS ofrece este modo de funcionamiento por defecto, pero además ofrece lo que se denomina redirección, es decir, la posibilidad de coger la salida de un programa que aparecería en pantalla y enviarla a la impresora, a un

puerto serie o a un fichero en el disco. Lo mismo puede hacerse con la entrada y utilizar los datos de un archivo como si su contenido se fuese escribiendo a través del teclado.

El sistema de redirección es utilizado principalmente por los programas denominados "filtros" (pequeñas utilidades que transforman datos), siendo este tipo de programas, debido a su pequeño tamaño y al gran volumen de datos que habitualmente tienen que tratar, los principales candidatos a realizarse en lenguaje ensamblador. Dentro del sistema operativo MS-DOS encontrará un gran número de programas de este tipo, como FIND, SORT o MORE.

Por tanto, el MS-DOS no tiene un dispositivo "teclado" y otro "pantalla", sino que los denomina "dispositivo de entrada estándar" y "dispositivo de salida estándar" para indicar el hecho de que pueden redirigirse hacia otro dispositivo o fichero en disco.

Las funciones de este tipo ofrecidas por la interrupción 21h son:

- Función 2h: Escribir un carácter en pantalla.
Simplemente se coloca en DL el carácter que se desea que aparezca en pantalla. No permite especificar ni el color ni el atributo.
- Función 9h: Escribir cadena de caracteres terminada en '\$'.
Esta función imprime la cadena de caracteres apuntada por DS:DX, cuyo final se marca por un signo \$, reconociendo además los caracteres especiales del servicio 0Eh de la BIOS, como puede verse en el siguiente fragmento de código:

Cadena	db	'Prueba',13,10,'Otra línea','\$'
inicio :	MOV AX,CS	
	MOV DS,AX	
	MOV DX,OFFSET Cadena	
	MOV AH,9h	
	INT 21h	

- Función 1h: Leer un carácter del teclado con eco y chequeo de Break.
Espera a que el usuario pulse una tecla, la muestra en pantalla y la devuelve en el registro AL. Si el usuario pulsa la combinación de teclas CTRL-Inter durante la espera, el programa finaliza.
- Función 6h: Leer un carácter del teclado sin espera.
Si el usuario ha pulsado alguna tecla, ésta se devuelve en el registro AL. En caso de que no pulsase ninguna la función retorna, devolviendo el valor 0 en AL. No se chequea la pulsación de CTRL-Inter.
- Función 7h: Leer un carácter del teclado sin eco y sin chequeo de Break.

```

CALL  Copia
MOV   AL,0      ;Llama a la rutina
              ;Carga código de retorno si no
              ;hay error en AL
JNC   NoError
INC   AL        ;Si hubo un error
              ;Le suma 1 al valor de retorno,
              ;devolviendo 1
              ;Retorna al DOS
NoError:MOV AH,4Ch
         INT 21h
Código ENDS
END   Inicio

```

OTRAS OPERACIONES CON FICHEROS

Además de leer y escribir, sobre un fichero pueden realizarse otra serie de operaciones como borrarlo, cambiar su nombre o sus atributos. Estas operaciones se llevan a cabo mediante los siguientes servicios de la interrupción 21h:

- Función 41h: Borrar un fichero.

La eliminación de un fichero se lleva a cabo mediante esta función, la cual únicamente necesita que se le proporcione la dirección donde se encuentra el nombre del fichero mediante los registros DS:DX. Si el fichero es de sólo lectura se devuelve un error, por lo que para eliminar un fichero de este tipo habrá que cambiar sus atributos previamente mediante la función 43h.

- Función 43h: Lee/Cambia los atributos de un fichero.

Esta función requiere que el nombre del fichero se encuentre apuntado por la pareja de registros DS:DX. Si se desean obtener los atributos del fichero se colocará en AL el valor 0, y a la vuelta de la función el registro CL los contendrá en el mismo formato que se explicó para la función de crear un fichero. Si lo que se desea es cambiar los atributos del fichero, entonces se colocará el valor 1 en AL y en CL los atributos del fichero.

- Función 56h: Renombrar un fichero.

El objetivo de esta función es el de cambiar el nombre de un fichero ya existente, del cual se especifica su nombre en la zona de memoria apuntada por DS:DX, mientras que el nuevo nombre se encuentra apuntado por ES:DI. Una característica interesante es que, si se especifica un directorio en el nombre del fichero de destino, entonces el fichero se moverá desde el directorio en el que se encuentra al especificado en el nuevo nombre (siempre y cuando ambos directorios se encuentren en la misma unidad de disco).

- Función 57h: Lee/Cambia la fecha y hora de un fichero.

Todos los ficheros tienen asociada una fecha y una hora que marca cuándo fue la última vez que se modificaron, encargándose el MS-DOS de llevar a cabo esta modificación de forma automática cada vez

que se cierra un fichero abierto para escritura. Sin embargo, a través de esta función puede realizarse tanto la lectura de la fecha y hora (AL=0) como la modificación (AL=1) del fichero abierto, cuyo *handle* se encuentra en BX. En lectura (AL=0) la hora se devuelve en el registro CX y la fecha en DX, mientras que en escritura debe situarse la hora en CX y la fecha en DX antes de llamar a la función. Tanto la fecha como la hora se encuentran en un formato propio del que se pueden convertir a través de las siguientes fórmulas:

$$CX = 2048 * \text{HORA} + 32 * \text{MINUTO} + \text{SEGUNDO} / 2$$

$$DX = 512 * (\text{AÑO}-1980) + 64 * \text{MES} + \text{DIA}$$

MANEJO DE DIRECTORIOS

El sistema operativo MS-DOS permite organizar los ficheros dentro de una estructura jerárquica de carpetas conocida como "Árbol de directorios". Sobre estos directorios se pueden realizar las siguientes funciones:

- Función 0Eh: Cambiar la unidad de disco.

Esta función permite seleccionar la unidad de disco activa, a la vez que informa acerca del número de unidades instaladas en el sistema. Debe indicarse la unidad a la que se desea cambiar en el registro DL, utilizando el valor 0 para la unidad A; 1 para la B; y así sucesivamente. Tras llamar a la interrupción 21h, el registro AL contiene el número total de unidades instaladas en el sistema.

Función 19h: Obtener unidad de disco activa.

Se trata de la función complementaria de la anterior, que devuelve en el registro AL el número de la unidad de disco que se encuentra activa actualmente (A=0, B=1, C=2...).

- Función 39h: Crear un directorio.

Esta función es equivalente al comando MD de MS-DOS, y requiere únicamente que se le proporcione la dirección donde se encuentra el nombre del directorio a crear en la pareja de registros DS:DX. Si se produce un error durante la creación, éste se devolverá de la forma habitual mediante la bandera de acarreo y el registro AH.

- Función 3Ah: Borrar un directorio.

Equivale al comando RD de MS-DOS, y borra el directorio cuyo nombre está contenido a partir de la posición de memoria apuntada por los registros DS:DX. El directorio no debe contener ningún archivo para que pueda eliminarse, o de lo contrario se producirá un error.

- Función 3Bh: Cambiar directorio.

El nuevo directorio al que se desea cambiar se encuentra contenido a partir de la dirección de memoria apuntada por DS:DX.

PUERTO SERIE Y DE IMPRESORA

Igual que sucedía con la BIOS, el MS-DOS también dispone de funciones específicas para el manejo de la impresora y el puerto serie, sólo que en este

El siguiente ejemplo resume los conceptos de manejo de ficheros realizando un pequeño programa que copia un archivo llamado Prueba1.Dat, situado en el directorio actual, al directorio raíz de la unidad C:, llamándole Prueba2.Dat. Obsérvese sobre todo el control de errores que se ha hecho, el manejo de dos archivos de forma simultánea, la utilización de una constante simbólica y la forma en que se han pasado los parámetros a la subrutina de copia, indicando el nombre del fichero origen en la dirección apuntada por DS:DX y el del de destino en DS:CX. A la salida, la función devuelve la bandera de acarreo a 1 si ha sucedido algún error durante la copia o a 0 para indicar un final correcto. Esta bandera se utiliza para devolver un valor de retorno al MS-DOS, que puede ser controlado mediante una condición de *ERRORLEVEL*.

```

TAMBUFFER EQU 4000
Pila SEGMENT PARA STACK "STACK"
DW 200 dup(?)
Pila ENDS
Datos SEGMENT
nombre1 DB 'Prueba1.dat',0
nombre2 DB 'C:\Prueba2.dat',0
buffer DB TAMBUFFER dup(?)
Datos ENDS
Codigo SEGMENT
ASSUME CS:Codigo,DS:Datos,SS:Pila
Copia PROC
MOV AX,3D00h ;Abrir fichero origen sólo lectura (AH=3Dh, AL=0h)
INT 21h ;Si hay error, retorna de la subrutina
JC Error ;Guarda handle del fichero origen en SI
MOV DX,CX ;Coloca DS:DX apuntando al nombre del fichero destino
XOR CX,CX ;Atributo normal
MOV AH,3Ch ;Crea el fichero INT 21h JC
ErrorC1 ;Si hay error, sale pero cerrando fichero origen
MOV DI,AX ;Si no hay error, guarda handle en DI
Bucle: MOV CX,TAMBUFFER ;Leer en bloques de TAMBUFFER bytes

```

MOV BX,SI		;Carga el handle del fichero origen en BX
MOV AH,3Fh	INT 21h	;Leer TAMBUFFER bytes
JC ErrorC2		;Si hay error, sale cerrando ambos ficheros
OR AX,AX		;Si se leyeron 0 bytes es que se llegó al final
JZ final	MOV CX,AX	;y se acaba la copia correctamente
MOV BX,DI		;Escribir tantos bytes como se leyeron
MOV AH,40h	INT 21h	;Carga el handle del fichero destino en BX
JC ErrorC2		;Escribir en fichero destino
CMP AX,CX		;Si hay error, cierra los dos ficheros abiertos
JNE ErrorC2		;Si se llenó el disco da un error
JMP Bucle		;y cierra los dos ficheros
final : MOV BX,DI		;Copiar siguiente bloque del fichero
MOV AH,3Eh	INT 21h	;Se cierra el fichero destino
MOV BX,SI	INT 21h	;Y luego el origen
CLC		;CF=0 para indicar que no hubo error
RET		;Error: Cierra fichero destino
ErrorC2 :MOV BX,DI		
MOV AH,3Eh	INT 21h	
ErrorC1 :MOV BX,SI		
MOV AH,3Eh	INT 21h	
Error : STC	RET	
Copia ENDP		
inicio : MOV AX,Datos		
MOV DS,AX		
MOV DX,OFFSET nombre1		
MOV CX,OFFSET nombre2		
		Ajusta el segmento
		Carga primer nombre en DS:DX
		Carga nombre fichero destino en DS:CX

Bit	Significado
0	Sólo lectura
1	Oculto
2	De sistema
3	Etiqueta de volumen
4	Directorio
5	Archivo

En caso de que el fichero ya exista, su contenido se borra. Tras su ejecución, esta función devuelve en el registro AX el *handle* del nuevo fichero.

- Función 3Dh: Abrir fichero ya existente.

Esta función se utiliza para leer o escribir datos en un fichero que ya existe. Los parámetros son los mismos que en la función anterior, indicando el nombre del fichero mediante los registros DS y DX y los atributos en CL. Además, en AL debe indicarse el modo de apertura del fichero, que puede ser uno de los siguientes:

Valor AL	Significado
0	Acceso sólo lectura
1	Acceso sólo escritura
2	Acceso lectura y escritura

Tras su ejecución, esta función devuelve en el registro AX el *handle* del fichero.

LECTURA Y ESCRITURA EN EL FICHERO

Tras la apertura ya será posible efectuar operaciones de lectura y de escritura. No siempre será posible llevar a cabo estas dos funciones sobre un fichero, y dependerá del modo en el cual haya sido abierto éste. Así pues, si se ha abierto con la función de Crear o se abrió sólo para escritura no será posible leer de él, de la misma forma que no se podrá escribir si se abrió sólo para lectura. La única ocasión en que pueden llevarse a cabo ambas operaciones es si se abrió el fichero para lectura y escritura.

Además de estas dos operaciones, también es posible colocarse en cualquier punto del fichero mediante la función de posicionamiento, permitiéndose así leer o escribir datos en cualquier punto del archivo.

Estas tres funciones (leer, escribir y posicionar) reciben siempre en el registro BX el *handle* del fichero sobre el que realizarán la operación.

- Función 3Fh: Leer de un fichero.

Esta función lee del fichero, cuyo *handle* se encuentra en BX, el

número de bytes indicado en el registro CX. Los datos leídos se almacenan en la dirección cuyo segmento y desplazamiento se indica en los registros DS y DX. Tras la operación, en AX se devuelve el número de bytes leídos, que puede ser menor que el pedido en caso de que se llegue al final del fichero. En caso de que suceda un error (CF=1), AH devolverá un código de error en lugar de los bytes leídos.

- Función 40h: Escribir en un fichero.

La operación de escritura es similar a la de lectura, sólo que en este caso los registros DS y BX contienen el segmento y desplazamiento de la zona de memoria que contiene los datos a grabar. El resto de los parámetros son los mismos: el *handle* del fichero en BX y el número de bytes a escribir en CX. Tras la llamada a la interrupción, el registro AX contendrá el número de bytes realmente escritos, que será menor que el que se indicó en CX en caso de que no haya suficiente espacio en el disco duro. Si ocurre algún error, se devuelve la bandera de acarreo a 1 y el registro AH contiene el código del error.

- Función 42h: Posicionarse en un punto de un fichero.

En ocasiones el acceso secuencial a un fichero no es suficiente y se necesita realizar un acceso aleatorio, es decir, poder leer o escribir porciones del fichero en cualquier orden. Mediante esta función es posible colocarse sobre cualquier punto del archivo, de forma que la siguiente operación de lectura o escritura se realizará desde dicha posición.

Para realizar el posicionamiento se coloca, en primer lugar, en BX el *handle* del fichero, y la posición se indica como un entero de 32 bits que irá contenido en los registros CX (16 bits superiores) y DX (16 bits inferiores). Además, en AL se indica un valor que especifica respecto a qué punto del fichero se realiza el posicionamiento, según la siguiente tabla:

Valor de AL	Posicionarse respecto a
0	Principio del fichero
1	Posición actual
2	Final de fichero

CIERRE DE FICHERO

Una vez finalizado el trabajo con él, el fichero debe cerrarse, de manera que el sistema operativo pueda utilizar su *handle* para abrir posteriormente otros ficheros. Se trata de una operación muy simple que se lleva a cabo mediante el siguiente servicio:

- Función 3Eh: Cerrar fichero.

Esta función recibe como único parámetro el *handle* del fichero a cerrar en el registro BX. Si hay un error, éste es indicado por la función colocando la bandera de acarreo a 1 y en el registro AH el código de error.

Espera a que el usuario pulse una tecla y la devuelva en AL. En caso de que éste pulse la combinación CTRL-Inter, es ignorada.

- Función 8h: Leer un carácter del teclado sin eco y con chequeo de Break. Funciona del mismo modo que la anterior, pero en caso de que éste pulse la combinación CTRL-Inter, el programa finaliza.
- Función 0Ah: Leer una serie de caracteres del teclado a una zona de memoria.

Esta función espera a que el usuario teclee una serie de caracteres finalizados por la tecla ENTER y los almacena en una zona de memoria. Los registros DS y DX indican, respectivamente, el segmento y el desplazamiento de una zona de memoria cuyo primer byte indica el número máximo de caracteres que caben, incluido el retorno de carro. Tras llamar a la función, el segundo byte de este bloque de memoria especifica el número de caracteres que el usuario ha tecleado, los cuales se encuentran contenidos a partir del tercer byte.

- Función 0Bh: Chequea si hay algún carácter disponible. Comprueba si el usuario ha tecleado algún carácter y devuelve el valor 0FFh en el registro AL en caso que haya alguno esperando, o 0 si el buffer de teclado está vacío.

- Función 0Ch: Borrar el buffer de teclado y llamar a función de teclado.

En caso que haya caracteres pendientes en el buffer de teclado, éstos se borran y a continuación se llama a la función de teclado, cuyo número se indica en el registro AL. Los números posibles son: 1h, 6h, 7h, 8h y Ah.

ACceso A FICHEROS

A la hora de guardar y recuperar información de un disco, las funciones del MS-DOS tienen una gran ventaja sobre las de la BIOS, y es la posibilidad que ofrecen de manejar ficheros de datos. Al contrario que sucedía con las funciones de la BIOS, para las que había que conocer la dirección física de la información que se quería leer o escribir, el MS-DOS maneja toda esa información, de forma que únicamente hay que indicarle el nombre del fichero al que se van a escribir o leer los datos y él se encarga de buscar el lugar donde se encuentra.

Para trabajar con ficheros, el MS-DOS ofrece dos métodos: a través de bloques de control de ficheros (FCB) o a través de handles o descriptores de ficheros. El primero de estos sistemas, el de los FCB, se utilizó únicamente en la versión 1 del MS-DOS y apenas se utiliza hoy en día, manteniéndose únicamente por cuestiones de compatibilidad.

Es por ello que sólo se va a tratar el manejo de ficheros a través de handles, pues se trata de un sistema más simple y potente que el antiguo de FCB.

A la hora de trabajar con un fichero, lo primero que hay que hacer es abrirlo. A través de esta operación el sistema operativo le asigna al fichero un handle o descriptor de fichero, que no es más que un valor de 16 bits (tipo word) que identifica a dicho fichero. De esta forma, cada vez que se quiera leer o escribir en el fichero sólo hará falta indicar su handle, en lugar de dar de nuevo el nombre completo. Puesto que el número de handles es un valor limitado, una vez que se finaliza el trabajo con un fichero hay que cerrarlo. Así, se le indica al MS-DOS que se ha terminado de utilizar dicho archivo y puede utilizar su handle para cualquier otro que se abra en el futuro.

Al principio de cualquier programa ya hay cinco handles creados (que van desde el número 0 al 5), y que corresponden, en este orden, a la entrada estándar, la salida estándar, la salida de errores estándar, el dispositivo auxiliar de salida (AUX) y la salida estándar de impresora. Estos handles son destruidos por el MS-DOS automáticamente cuando el programa finaliza. Además, dado que la entrada y salida se consideran ficheros, es posible utilizar las funciones de manejo de ficheros para enviar datos a la pantalla o a la impresora, o para leerlos del teclado.

Las funciones de manejo de ficheros utilizan la bandera de acarreo para indicar si ha sucedido algún error, devolviendo además en el registro AH un código de error de los indicados en la tabla anterior. Si una función requiere un nombre de fichero, éste debe darse como una cadena de caracteres terminada por un byte al valor 0. Este nombre de fichero puede contener no sólo un nombre de fichero, sino también una especificación de unidad y de directorio, como puede verse en los siguientes ejemplos:

nombre1	DB	'Fichero1.Txt',0
nombre2	DB	'C :Fichero1.Txt',0
nombre3	DB	'D :\Ejemplos\Prueba\Fichero1.Txt',0

ABRIR EL FICHERO

Antes de utilizar un fichero, éste debe abrirse con alguna de las dos funciones siguientes, las cuales devuelven el handle con el que se referirá al archivo en posteriores operaciones:

- Función 3Ch: Crear un fichero para escritura. Esta función recibe en los registros DS y DX el segmento y desplazamiento de una cadena de caracteres con el nombre y directorio del archivo que se va a crear. El final de esta cadena se marca por un byte con el valor 0. Los diferentes bits del registro CL indican los atributos que tendrá el fichero según la siguiente tabla:

HILOAD. HiLoad. Ejemplo de cómo cargar programas en la memoria superior.

IBMTOKEN. IBM Token. Controlador de paquetes para una red de tipo Token Ring.

MB. Mido Box. Se trata de un secuenciador MIDI realizado completamente en ensamblador.

NNANSI3. Nnansi 3.0. Controlador ANSI de pantalla que soporta tarjetas gráficas VGA.

OSKDL10. OSK DL10. Núcleo de un sistema operativo para 386, con controladores incluidos.

PASSWORD. Password. Driver que se coloca en el CONFIG.SYS para pedir una clave de acceso al ordenador.

PRIAC. Priac. Programa que lee el contenido de la zona de memoria de comunicación entre aplicaciones de la BIOS.

SETENV. SetEnv. Varios programas para crear y modificar variables de entorno.

STDERRF. STDERR f. Programas que redirigen la salida de error estándar.

TCRND. TCRND. Rutinas de cálculo de números aleatorios.

TDSK22SR. TDSK 2.2. Disco RAM con tamaño modificable que puede utilizar memoria EMS y XMS.

TRACE. Trace. Programa que controla todas las llamadas que haga un programa a las interrupciones.

TSWORKS. TSR-Works. Varios programas para el manejo de programas residentes.

UNLS8616. Unload86 1.6. Convierte ficheros binarios a formato hexadecimal Intel o BNP.

VARIOS. Varios. Multitud de programas con fuentes en ensamblador.

Directorio ENSAMBLA: Ensambladores.

A86V402. A86 4.02. Ensamblador shareware muy bueno, aunque su sintaxis difiere de la que utilizan los ensambladores comerciales contenidos en este libro.

ASSEMBLE. Assemble. Pequeño ensamblador muy adecuado para realizar programas muy simples, pues genera ficheros ejecutables directamente, sin necesidad de enlazado.

CHEAP401. Cheap Assembler 4.10. Ensamblador freeware para crear tanto programas ejecutables como rutinas en código máquina que pueden llamarse desde lenguajes como Basic y Pascal.

MAGIC110. Magic Assembler 1.10. Este ensamblador puede generar tanto ejecutables de tipo COM como sectores de arranque para discetes.

MAX31. Max 3.1. Simple ensamblador que crea programas de tipo COM y BLD sin necesidad de utilizar enlazador o la utilidad EXE2BIN.

NASM090. Netwide Assembler 0.90. Ensamblador muy completo que además puede generar un gran número de formatos de fichero ejecutable para otros sistemas operativos como Linux o Windows NT.

VALARROW. Arrowsoft Assembler 1.0D. Conjunto compuesto de un ensamblador, un editor y un enlazador equivalentes a la versión 3.0 de MASM.

WASM202. Wolfware Assembler 2.02. Un ensamblador muy bueno, y que incluye una completa documentación acerca de su funcionamiento.

Directorio HEXEDIT: Editores de fichero en formato hexadecimal.

CEDITS6. Cheat Editor 5.6. Editor de ficheros binarios que cuenta además con un modo de desensamblado de código.

HEX51. Hex 5.1. Editor hexadecimal con opciones de búsqueda y conversor entre números en diferentes bases de numeración.

HXEDIT21. Hex Edit 2.1. Completo editor hexadecimal que cuenta con la interesante capacidad de deshacer los cambios que se hayan hecho a un archivo (incluso los realizados en anteriores ejecuciones del programa).

XED11B. XED 1.1b. Editor hexadecimal que a sus avanzadas capacidades de edición añade un desensamblador, edición de bits y generación de ficheros de datos en lenguaje C.

ZAPIT307. Zap-It 3.07. Con este editor es posible editar tanto ficheros binarios como sectores de discos. Incluye una sencilla calculadora de conversión entre bases.

Directorio UTIL: Utilidades para programar en ensamblador.

ABSADR. Absolute Addresses. Convierte direcciones en formato segmento:desplazamiento a direcciones lineales de memoria.

AFIX. A86 Fix. Programa que convierte un gran número de utilidades freeware en ensamblador para que puedan ser ensambladas con la utilidad A86.

ASMLINT. ASM Lint. Utilidad que permite detectar conflictos entre los tipos de datos utilizados en diferentes módulos del programa ensamblador.

BIN. Bin. Utilidad equivalente al EXE2BIN, pero de dominio público.

INCBIN. Include Binary. Esta utilidad permite insertar ficheros de datos dentro de programas de tipo EXE realizados en ensamblador.

LINK. Val Linker. Enlazador experimental del cual se incluye tanto el ejecutable como su código fuente en C.

MEMORY43. Memory 4.3. Programa residente que permite en cualquier momento comprobar la ocupación de memoria que hacen los programas, las llamadas a interrupciones que realizan e incluso desensamblarlos.

POKE10. Poke 1.0. Programa que permite cambiar el valor de posiciones de memoria del PC. Puede utilizarse como un programa normal o en el CONFIG.SYS durante el arranque del ordenador.

PX. Procedure Cross Referencer. Utilidad que coge un programa en ensamblador y genera un listado de todas sus subrutinas y las llamadas entre ellas.

IMPORTANTE: En el directorio LIBRO1 se ha incluido parte del software que aparecía de regalo en el primer número de esta colección, titulado "Cómo Programar tus propios Juegos". Para aquel que no hubiera adquirido este primer número, será de mucha utilidad, pues en él se incluyen librerías, juegos y editores de texto, gráficos y sonido. Aquel que ya lo haya adquirido, debe instalar el software desde este CD, pues debido a un fallo en la duplicación del software del número anterior, algunas de las utilidades que se incluían podrían dar problemas.

DIRECTORIO DOC: Documentos e información.

ASMMAG. ASM Magazine. Varios números de una revista electrónica especializada en programación en ensamblador.

ASMTUTOR. Assembler Tutor. Dos archivos de texto que contienen una pequeña introducción a la programación en lenguaje ensamblador.

CURSOASM. Curso de ensamblador. Pequeña introducción a la programación en ensamblador escrita en castellano.

FAQ. Frequently Asked Questions. Varias preguntas comunes relacionadas con el lenguaje ensamblador y sus respuestas.

INTER51. Interrupt List 5.1. Una gran fuente de referencia acerca de interrupciones, funciones de la BIOS y MS-DOS, instrucciones, puertos de entrada/salida, etc...

PCASMTUT. PC Assembly Tutor. Tutorial de programación en ensamblador con multitud de ejemplos.

PMTUT. Protected Mode Tutor. Breve explicación acerca de cómo crear programas que funcionen en modo protegido.

PVI. PVI. Información acerca de las interrupciones virtuales en modo protegido del procesador Pentium.

UNDOC. Varios. Información sobre características no documentadas de los procesadores Intel.

VARIOS. Varios. Información sobre diversos temas.

DIRECTORIO EDITOR: Editores de texto.

ALABV10. Assembler Laboratory 1.0. Editor de texto para programación en ensamblador que cuenta con coloreado de sintaxis, marcado de errores, gestor de proyectos y definición de macros.

ASMED182. Asm Edit 1.82. Entorno de programación en ensamblador, que cuenta además con una completa ayuda de instrucciones de todos los procesadores hasta el Pentium, interrupciones y tablas de MS-DOS.

ASMENV19. ASM Environment 1.9. Pequeño editor que puede llamar al ensamblador y al enlazador sin necesidad de salir de él.

BE400. Bingo Text Editor 4.00. Editor de texto que acepta ficheros de cualquier tamaño y que cuenta con menús modificables, deshacer múltiples acciones y un lenguaje de programación de macros similar al C.

BOXER70B. Boxer 7.0b. Completo editor especialmente indicado a la programación en cualquier lenguaje. Cuenta también con una calculadora y una tabla ASCII.

CALVIN22. Calvin 2.2. Versión simplificada del editor VI del sistema operativo UNIX.

FEDSH20. FED 2.0. Editor para MS-DOS con funciones avanzadas de búsqueda y sustitución, edición de ficheros binarios, acceso a sistemas de ayuda externos y marcado de errores de compilación.

MELITE. Multi-Edit Lite. Este editor puede manejar hasta 25 archivos de cualquier tamaño de forma simultánea. Puede llamar a compiladores externos dejando prácticamente toda la memoria libre.

EN300. NE 3.00. Editor freeware diseñado para sustituir al EDIT que acompaña al

sistema operativo MS-DOS, contando además con funciones avanzadas para el procesador de textos.

PCEDIT32. PCEdit 32. A pesar de que este editor ha sido programado pensando en su portabilidad, resulta muy adecuado para realizar programas en cualquier lenguaje de programación, aunque se encuentra especialmente orientado al C.

SMED101. Smooth Editor 1.01. La característica más destacable de este editor es que realiza los cambios de pantalla mediante un scroll muy suave del texto.

TED. Ted. Editor para programas en ensamblador utilizando los programas de la compañía Borland (TASMX, TLINK y Turbo Debugger).

TSEJR4. TSE Jr 4. Editor de propósito general que ocupa muy poco espacio, pero que es uno de los que mayor número de funciones incorpora.

TSETD25. TSE Pro 2.5. Versión ampliada del editor TSE Jr que cuenta con emulación de varios editores muy extendidos y un completo sistema de macros. Está limitado a 30 minutos de uso por sesión.

VIM42P16. VIM 4.2 16 bits. Versión ampliada y mejorada del popular editor VI de Unix con ayuda, soporte de ratón, deshacer y muchas nuevas opciones.

VIM42P32. VIM 4.2 32 bits. Versión de 32 bits para procesadores 386 y superiores del anterior editor.

DIRECTORIO EJEMPLOS: Programas creados en ensamblador, con su código fuente.

800K. 800k. Programa que formatea discos de 5 _ y doble densidad a 800KB y un programa residente para que sean reconocidos por el MS-DOS.

ANSWR202. Answer 2.02. Utilidad para ficheros BAT que lee una línea de texto y la guarda en una variable de entorno.

ASSIGN. Assign. Utilidad como el ASSIGN de MS-DOS.

ASTRSYS. Varios. Cuatro pequeñas utilidades para utilizar en ficheros de proceso por lotes.

BIMODAL. BiModal. Ejemplo de programación de un gestor de interrupciones en modo real y protegido.

BIOS. BIOS. Una BIOS desensamblada y comentada.

BYEPC300. Bye PC 3.00. Controlador de módem residente con un interfaz para programas en MS-DOS.

CTRLALT. CTRLALT. Utilidad residente con seis funciones distintas que se activan mediante combinaciones de teclas.

CUBE. Cube. Programa que resuelve un rompecabezas tridimensional.

EDIT. E. Pequeño editor.

FREEMACS. Freemacs. Editor similar al EMACS, con fuentes en ensamblador.

FUNPACK. Fun Pack. Conjunto de varios minijuegos y programas curiosos realizados en ensamblador.

GUSMOD. GusMod. Reproductor de archivos de música en formato MOD para la tarjeta de sonido Gravis UltraSound.

HD. Hard Disk. Programa de diagnóstico del disco duro.

HELP. Help. Sistema para crear pantallas de ayuda residentes en memoria que se activan con una combinación de teclas.

caso simplemente abarcan la parte de envío y recepción de datos, debiéndose recurrir a la BIOS para efectuar operaciones más complejas, como la configuración del puerto serie u obtener su estado.

- Función 3h: Leer carácter del puerto serie.

Esta función recibe un carácter desde el dispositivo serie estándar AUX, que normalmente coincide con COM1. El carácter se recibe en AL.

- Función 4h: Escribir carácter en el puerto serie.

Envía el carácter contenido en el registro DL a través del dispositivo serie estándar AUX.

- Función 5h: Enviar carácter a la impresora.

Esta es la única función que ofrece el DOS para el manejo de la impresora. Su finalidad es la de enviar el byte almacenado en DL al dispositivo impresor estándar, conocido como PRN.

SERVICIOS DE FECHA Y HORA

El MS-DOS mantiene un reloj con la hora y la fecha actual, disponiendo además de un conjunto de funciones para leer y modificar sus valores. Éstas son:

- Función 2Ah: Leer la fecha.

Tras llamar a este servicio, los registros CX,DX y AL contienen la fecha actual en el siguiente formato:

DL contiene el día del mes (1 a 31).

DH contiene el número del mes (1 a 12).

CX contiene el año (1980 a 2099).

AL contiene el día de la semana, desde el 0 (domingo) hasta 6 (sábado).

- Función 2Bh: Cambiar la fecha.

Esta función modifica la fecha del sistema a la indicada en los registros DX y CX, dando los datos en el mismo formato que los devolvía la función anterior, es decir, el día en DL, el mes en DH y el año en CX.

- Función 2Ch: Leer la hora.

Devuelve la hora del día calculada a partir del temporizador de la BIOS en los registros CX y DX en el siguiente formato:

CH contiene la hora (0 a 23).

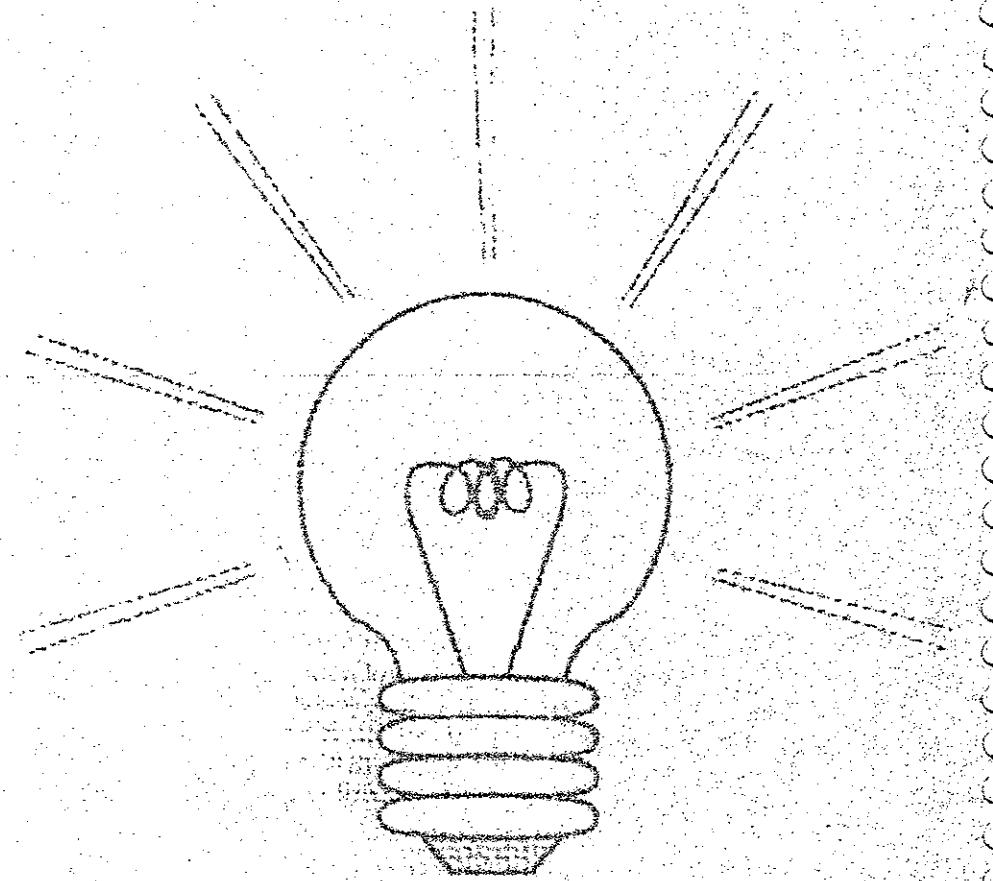
CL contiene los minutos (0 a 59).

DH contiene los segundos (0 a 59).

DL contiene las centésimas de segundo (0 a 99).

- Función 2Dh: Cambiar la hora.

Cambia la hora a la especificada por los registros CX y DX, según el formato indicado para la función anterior.



Características Avanzadas del Ensamblador



Hasta este punto se han tratado únicamente los aspectos fundamentales de la programación en el lenguaje ensamblador, con los que se pueden ya crear programas útiles de cualquier complejidad. Existen, sin embargo, toda una serie de posibilidades avanzadas que, si bien no son indispensables, sí que facilitan la tarea de programación en un gran número de ocasiones. Aunque al principio resulten algo complicadas de comprender y manejar, sobre todo en lo referente a las macros, en cuanto el lector comprenda su funcionamiento le ahorrarán una buena parte del tiempo de programación.

División en módulos

Una de las facilidades más utilizadas del ensamblador es la posibilidad de dividir el programa en varios módulos separados, cada uno de los cuales agrupa un número de funciones o rutas similares. Estos módulos se ensamblarán por separado, con el consiguiente ahorro de tiempo, pues sólo será necesario reensamblar los módulos que cambien para obtener los nuevos ficheros objeto, quedando los demás como están. Es a la hora de realizar el enlazado cuando se unirán los diferentes módulos para dar lugar al programa definitivo.

Para trabajar con módulos es necesario seguir los siguientes pasos:

En primer lugar, todas aquellas partes de un módulo que deseen utilizarse desde otro deben marcarse como de tipo PUBLIC. Esto puede hacerse con los segmentos y las subrutinas simplemente especificando el atributo PUBLIC en su declaración. El problema viene en cómo señalar los elementos de datos, etiquetas de código y constantes simbólicas que se desean utilizar desde otros módulos. La forma de hacerlo es muy simple, y consiste en utilizar la directiva PUBLIC seguida de los nombres de los símbolos que se desean hacer accesibles. El siguiente ejemplo muestra como hacerlo:

```
PRUEBA EQU 10
dato DW PRUEBA
codigo : JMP codigo
```

```
PUBLIC PRUEBA,dato,codigo
```

El segundo paso que hay que seguir es indicarle al ensamblador dentro de cada módulo qué elementos son los que coge de otros. Esto se hace colocando la directiva EXTRN seguida de los nombres de los símbolos de otros módulos que se utilizarán en el actual. Cada uno de los símbolos irá seguido por dos puntos y el tipo de símbolo de que se trata, que puede ser uno de los siguientes:

- Para variables: BYTE, WORD o DWORD.
- Para etiquetas y subrutinas: NEAR o FAR.
- Para constantes: ABS.

Por ejemplo, si un módulo desease utilizar los símbolos hechos públicos en el módulo del ejemplo anterior se utilizaría la siguiente línea:

```
EXTRN PRUEBA :ABS,codigo :FAR,dato :WORD
```

pudiéndose luego utilizar dichos símbolos en cualquier punto del programa:

```
MOV AX,PRUEBA
JMP Código.
MOV AX,dato
```

Si lo que se desea es, simplemente, mantener partes del programa en ficheros separados, pero no se necesita el ensamblado por separado, existe una alternativa más simple a este método, consistente en utilizar la directiva INCLUDE, la cual incluye el contenido del fichero que se indica como parámetro a partir del punto en que aparece la directiva INCLUDE. Por ejemplo, si en el programa se tiene un fichero denominado UTIL.ASM con algunas subrutinas necesarias para el programa, se utilizaría la siguiente línea:

codigo	SEGMENT	
INCLUDE UTIL.ASM		;Coloca en este punto del programa las rutinas
		;Aquí continua el programa

Macros

Las macros son otras de las funciones avanzadas del lenguaje ensamblador. Con ellas es posible crear pseudoinstrucciones, es decir, nuevas instrucciones del lenguaje ensamblador que en realidad no son más que una serie de instrucciones de las antiguas. Cuando se utiliza una de estas nuevas instrucciones dentro de un programa, el ensamblador la sustituye por la serie completa de instrucciones de que está compuesta, ensamblándolas a continuación. A este proceso se le denomina "expansión de la macro", y tiene lugar en todos los puntos del programa donde se utilice la macro. Dicho de otra forma, una macro es un tipo especial de subrutina que, en lugar de encontrarse su código en un único lugar, se coloca en todos los lugares en los que se utiliza. Esto hace evidente que una de las principales características que debe tener una macro es un tamaño pequeño. Además,

puesto que las macros no son más que subrutinas camufladas, hay que tener en cuenta si éstas modifican algún registro o bandera de estado, y tenerlo en cuenta en las instrucciones siguientes a la macro.

Las macros se definen mediante la directiva MACRO precedida del nombre de la macro, y seguida por los nombres de los operandos que necesita separados por comas. En líneas sucesivas se colocarán las instrucciones de la macro, pudiéndose colocar como operandos de ellas los parámetros de la misma. Por último, el final de la macro se señalará con la directiva ENDM. Veamos cómo crear una macro que realiza una operación de suma pero que acepta tres argumentos, los dos sumandos y el lugar donde depositar el resultado:

```
Suma MACRO destino,op1,op2
    MOV destino,op1
    ADD destino,op2
ENDM
```

Si, por ejemplo, dentro de un programa se utilizase esta macro:

```
MOV Dato,4444h
MOV AX,1234h
MOV BX,5678h
Suma CX,AX,BX
Suma DX,CX,Dato
```

se sustituiría por las siguientes instrucciones:

```
MOV dato,4444h
MOV AX,1234h
MOV BX,5678h
MOV CX,AX
;Expansion de la primera utilización
;de la macro
ADD CX,BX
MOV DX,CX
;Expansion de la primera utilización de
;la macro
ADD DX,Dato
```

A la hora de utilizar una macro, ésta sustituye ciegamente los parámetros que se le pasan en todos los operandos de la macro que los utilicen. Pueden, por tanto, construirse instrucciones ilegales que den lugar a errores de ensamblado, tal y como sucede si se llama a la macro del ejemplo anterior de la siguiente forma:

Suma Dato,Dato,AX
se expandiría como

```
MOV Dato,Dato
ADD Dato,AX
```

y la primera línea daría un error, puesto que ambos operandos acceden a un dato en memoria.

Puesto que las macros se expanden en todos los lugares en que se utilizan, no es posible utilizar en ellas etiquetas normales puesto que, si se expandiese la macro más de una vez, aparecería la misma etiqueta en varias partes del programa, produciéndose un error. La forma de solucionar este inconveniente es utilizando las etiquetas locales, las cuales tienen vigencia únicamente dentro de la expansión de la macro, no existiendo ya fuera de ellas y permitiendo, por tanto, su utilización repetida.

Las etiquetas locales se definen mediante la directiva LOCAL seguida de los nombres de las etiquetas separados por comas. El siguiente ejemplo define una macro denominada multiplica, la cual multiplica dos números por el método de sumas repetidas:

```
Multiplica MACRO op1,op2
LOCAL bucle
    MOV CX,op2
bucle : ADD op1,op2
        LOOP bucle
ENDM
```

Cuando se utilice esta macro debe tenerse en cuenta que modifica el registro CX, por lo que, si se desea preservar su contenido, habrá que guardarlo en la pila antes de llamar a la macro y recuperarlo a continuación:

```
PUSH CX
Multiplica AX,2
POP CX
```

Sin embargo, los parámetros de las macros no tienen por qué utilizarse únicamente como operandos de instrucciones, sino que es posible unirlos a otros elementos utilizando el operador &, que concatena un operando de la macro como si fuese un texto a cualquier otro elemento. Por ejemplo, la siguiente macro tiene dos parámetros: una condición y una instrucción. En caso de que se cumpla la condición se ejecutará la instrucción:

Si LOCAL	Macro Falso	Cond, Instr
	JN&Cond	Falso
	Instr	
Falso:		ENDM

Y un ejemplo de su utilización sería:

```
SI E,<INC AX>
SI GE,<MOV BX,CX>
```

que se expandirían como:

JNE	Falso1
INC	AX
Falso1:	
JNGE	Falso2
MOV	BX,CX
Falso2:	

En estos ejemplos puede verse la utilización de los símbolos <> para delimitar un argumento de una macro, cuyo uso es obligado cuando el parámetro de una macro contiene espacios, comas o tabuladores.

Otro operador utilizado en macros es el de cálculo, el cual convierte una expresión de texto en su valor numérico. Por ejemplo, sea la siguiente macro, que suma a su primer argumento el número indicado en el segundo más el indicado en el tercero (el segundo y el tercero deben ser números). Se haría así:

```
Sumados MACRO dest,n1,n2
    ADD dest,%n1+n2
ENDM
```

Si ahora se utilizase lo siguiente:

Sumados AX,3,2

se sustituiría por:

ADD AX,5

en lugar de ADD AX,3+2, que sería el resultado si no se hubiese utilizado el operador %

MACROS DE REPETICIÓN

Además de las macros normales existe un tipo especial, denominado "macros de repetición", que son utilizadas para repetir varias veces un número de instrucciones contenidas entre la directiva de la macro y el

ENDM. Al contrario que sucede con las macros normales, las de repetición no tienen nombre y se colocan únicamente en el lugar donde aparecen. Para conseguir su reutilización deben colocarse dentro de una macro.

La primera de estas directivas, IRP, toma dos parámetros: una variable y una lista de valores para dicha variable colocados entre <>, y separados por comas. La macro expande las instrucciones contenidas entre la directiva IRP y la ENDM una vez por cada uno de los valores indicados, tomando durante dicha expansión la variable el valor que se está tratando. Por ejemplo, si se desea crear una tabla de datos con la tabla del 7, se haría lo siguiente:

```
IRP numero,<0,1,2,3,4,5,6,7,8,9>
DB numero * numero
ENDM
```

lo que se sustituiría por el ensamblador como:

DB 0

DB 7

DB 14

y así sucesivamente hasta

DB 63

Otra directiva de este tipo es IRPC, muy similar a IRP, pero que utiliza como valores únicamente una cadena de caracteres, repitiéndose una vez por cada uno de los caracteres de la cadena, y tomando la variable el valor de dicho carácter. Con esta directiva, el ejemplo anterior quedaría:

```
IRPC numero,<0123456789>
DB numero * numero
ENDM
```

Su expansión sería igual a la de la macro anterior. Sin embargo, este sistema no serviría si, por ejemplo, se desease crear la tabla hasta el número 10.

Por último, la directiva REPT utiliza un único parámetro, que indica el número de veces que deben expandirse sus instrucciones. Si se desea realizar el mismo ejemplo de antes mediante esta nueva directiva, habrá que echar mano de una variable de ensamblado:

```
NUMERO = 0
REPT 10
    DB NUMERO * NUMERO
    NUMERO = NUMERO +1
ENDM
```

Variables de tipo registro (STRUC) y uniones (UNION)

Una estructura es una variable que agrupa en su interior a otras variables que se encuentran relacionadas entre sí. Por ejemplo, si se desease guardar el nombre, la edad y la estatura de una persona se podría crear una variable de tipo registro llamada Persona, la cual contiene en su interior tres variables con estos tres datos. Durante el programa sería posible crear variables de este tipo como si se tratase de cualquier otro.

La definición de una estructura de tipo registro se hace a través de la directiva STRUC, la cual va precedida del nombre identificativo que servirá más adelante para crear las variables de dicho tipo. En sucesivas líneas se definirán los diferentes campos de la estructura utilizando las directivas de definición de datos habituales. Por último, el fin de la estructura se marcará mediante la directiva ENDS precedida del nombre que se le dio a la misma. Esta definición de la estructura no reserva memoria alguna, sino que únicamente define una plantilla para que más adelante sea posible crear variables de este tipo.

```
Persona STRUC
    nombre      DB 30 dup( ?)
    edad       DB ?
    estatura   DW ?
Persona ENDS
```

Para crear variables de este tipo, se colocará el nombre de la variable seguido del nombre de la estructura y de una lista de datos de inicialización situados entre < y >, y separados por comas. Si se omite algún valor significará que se utilizará el valor por defecto indicado en la declaración de la estructura.

Por ejemplo, para definir la variable *Juan* y otra *Cliente* se utilizaría lo siguiente:

```
Juan Persona <'Juan García',20,170>
Cliente     Persona<,,,>
```

La variable *Juan* se inicializa con unos valores determinados, mientras que la *Cliente* se deja en blanco.

Es posible también declarar varias variables STRUC a la vez mediante la directiva dup:

```
Clientes Persona 10 dup <,,,>
```

De una estructura puede obtenerse su dirección (OFFSET), pero no puede utilizarse como una variable normal. Para acceder a los valores que contiene una estructura se utiliza el nombre de la variable de tipo estructura seguida de un punto y del nombre del campo al que se quiere acceder:

```
MOV AX,Juan.estatura
MOV Cliente.Edad,30
MOV BX,OFFSET Cliente.Nombre
MOV AL,Juan.Nombre[4]
```

Las uniones, por su parte, son un tipo especial de registros en los que todas las variables que contienen ocupan una misma zona de memoria. Se utilizan para acceder a variables tratándolas como diferentes tipos. Por ejemplo, si se desea tratar una variable de tipo doble word como si fuesen dos words consecutivas, se podría utilizar una variable de tipo union como en el siguiente ejemplo:

acceso	UNION	
	Dobleword	DD ?
	word	DW 2 dup(?)

acceso ENDS

Si ahora se declara una variable:

variable acceso<,,>

se podría acceder a ella como:

```
MOV variable.word,1234h
MOV variable.word[2],5678h
LES DI,variable.Dobleword
```

Campos de bits (RECORD)

Dado que en ensamblador es frecuente el trabajo con valores que se almacenan a nivel de bits, existe un tipo especial de estructura que almacena campos cuya longitud se expresa en bits. La longitud de estas estructuras no puede sobrepasar los 16 bits.

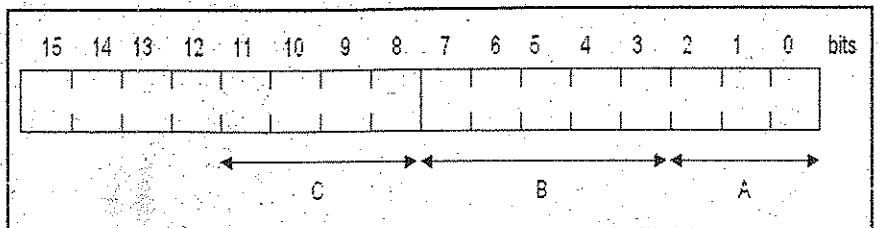
Para definir una estructura de este tipo se coloca la directiva RECORD, precedida por un nombre identificativo y seguida de las definiciones de los campos de bits separados por comas. La definición de un campo tiene la siguiente estructura:

nombre _campo:longitud_en_bits=valor_inicial

Por tanto, el siguiente ejemplo:

R RECORD A:3=2,B :5,C :4=10

define una estructura de registro con tres campos: A, de 3 bits y valor por defecto 2; B, de 5 bits sin valor por defecto y C, de 4 bits y valor 10 por defecto. La longitud total será, por tanto, de 12 bits, que se almacenarán en una variable de tipo word con la siguiente estructura:



Ahora se puede declarar una variable de este tipo:

dato R <1,8,>

que creará una variable con el valor 1 para el campo A, el valor 8 para el B y el 10 para el C (valor por defecto). Por tanto, la variable tendrá el valor:

1010 01000 001 = 0A41h.

Al contrario que sucedía con las variables de tipo registro, los campos de bits no pueden utilizarse de forma directa, debiéndose recurrir a las operaciones lógicas OR, AND, NOT y XOR para modificarlos. Sin embargo, las máscaras para realizar estas operaciones pueden obtenerse mediante el operador MASK, que genera una máscara dando el valor uno a los bits relativos al campo y 0 al resto. Además, el nombre del campo representa a su desplazamiento dentro de la variable.

Por tanto, para colocar el valor 7 en el campo B se realizaría lo siguiente:

MOV	AX,MASK B	;Carga la máscara en AX.
NOT	AX	;La invierte para que tenga todo 1 menos en el campo B que estará a 0.
AND	dato,AX	;Pone a 0 el campo B.

MOV	AX,7	;Coge el valor
MOV	CL,B	;Lo desplaza hasta la posición que debe tener
SHL	AX,CL	;dentro de la variable
OR	dato,AX	;Lo coloca mediante una instrucción OR

Glosario



ARGUMENTO:

Dato que necesita una subrutina para su funcionamiento. Cuando se llama a la subrutina deberá darse como parámetro un valor para cada uno de sus argumentos.

ASCII:

Juego de caracteres que define la ordenación y el valor numérico de cada una de las letras que utiliza el PC.

BASE DE NUMERACIÓN:

Los números se forman combinando una serie de dígitos entre sí. La base de numeración indica de cuántos de estos dígitos se dispone para formar las cifras.

BINARIO:

Sistema de numeración que cuenta únicamente con dos cifras: el 0 y el 1. Es el sistema de numeración utilizado por el procesador del PC.

BIOS:

(Basic Input-Output System): Pequeña memoria ROM situada en el ordenador que se encarga de arrancar el ordenador e iniciar la carga del sistema operativo. Además, incluye una serie de rutinas útiles para el manejo de periféricos.

BIT:

Cifra binaria que puede tomar el valor 0 o el 1.

BYTE:

Cantidad binaria de ocho bits que puede tomar un valor entre 0 y 255. En el PC representa, además, la mínima cantidad de memoria que se puede direccionar de forma individual.

CÓDIGO DESENSAMBLADO:

Forma de representar el código máquina, de forma que sea comprensible para las personas.

CÓDIGO MAQUINA:

Serie de instrucciones de un programa, en forma de valores binarios, en el formato que los entiende el procesador.

Compilador: Programa que convierte el código escrito en un lenguaje de alto nivel a código máquina.

DEPURADOR:

Programa que permite observar el funcionamiento de otro, de forma que se puedan localizar los errores de programación que contenga.

DESPLAZAMIENTO:

Posición dentro de un segmento en la que se encuentra una posición de memoria determinada.

DIRECCIÓN DE MEMORIA:

Valor numérico que identifica una posición de memoria. En el PC, las direcciones de memoria se componen de dos valores de 16 bits, denominados segmento y desplazamiento respectivamente.

DIRECCIONAMIENTO:

Modo de calcular la posición de memoria a la que hace referencia una instrucción de ensamblador.

DIRECTIVA:

Instrucción propia del programa ensamblador que no genera código, pero que indica la forma en que deben ensamblarse las instrucciones del programa.

DMA:

Sistema de transferencia de datos entre la memoria del ordenador y un periférico, de forma que el procesador puede realizar otras tareas mientras tanto.

ENLAZADOR:

Programa que realiza la unión de los diferentes módulos que componen un programa para dar lugar a un fichero ejecutable por el sistema operativo.

ENSAMBLADOR:

Lenguaje de programación de bajo nivel que cuenta con las instrucciones de código máquina, así como una serie de ayudas a la programación en este lenguaje. También se aplica este nombre al programa que realiza la conversión entre este lenguaje y el código máquina.

ETIQUETA:

Nombre simbólico que se le da a una posición de memoria para trabajar con ella, dejando que sea el ensamblador quien le asigne la dirección física de memoria que considere más conveniente.

HEXADECIMAL:

Sistema de numeración con 16 dígitos (los números del 0 al 9 y las letras de la A hasta la F). Se utiliza mucho en ensamblador, pues tiene una correspondencia directa con el sistema binario.

INTERRUPCIÓN:

Señal que envía un dispositivo al procesador para que éste detenga momentáneamente la ejecución del programa y le envíe o reciba algún dato de él.

LIBRERÍA:

Fichero que contiene varios módulos objeto para utilizar en programas.

LINKER:

Véase enlazador.

MACRO:

Ayuda que ofrece el lenguaje ensamblador para crear nuevas instrucciones a partir de fragmentos de programa que se utilicen de forma habitual.

MÁSCARA:

Serie de bits que se utilizan para aislar, borrar o cambiar parte de los bits contenidos dentro de un byte o word.

MNEMÓNICO:

Nombre que se le da a una instrucción de código máquina y que da idea de la acción que realiza.

MÓDULO OBJETO:

Formato de código máquina que contiene información adicional para que el enlazador pueda unirlo correctamente a otros del mismo tipo con el fin de formar el programa.

OCTAL:

Sistema de numeración que cuenta con ocho cifras diferentes (del 0 al 7).

OFFSET:

Véase Desplazamiento.

OPERANDO:

Valor concreto que toma el argumento de una instrucción o subrutina a la hora de llamarla.

PALABRA:

Tamaño de información para la que el procesador está más preparado para trabajar. El tamaño de palabra es de 16 bits para los procesadores anteriores al 386, y de 32 bits para éste y los siguientes.

PERIFÉRICO:

Dispositivo conectado al ordenador y que se comunica con él.

PUERTO:

Espacio de direcciones especial dentro de la memoria que sirve para la comunicación con otros componentes del ordenador y periféricos.

PROMPT:

Símbolo que muestra un programa invitando al usuario a que introduzca una orden o respuesta.

RESULTADO:

Valor que devuelve una subrutina.

SEGMENTO:

Bloque de memoria en el que se encuentra una posición de memoria determinada.

SIMBOLO:

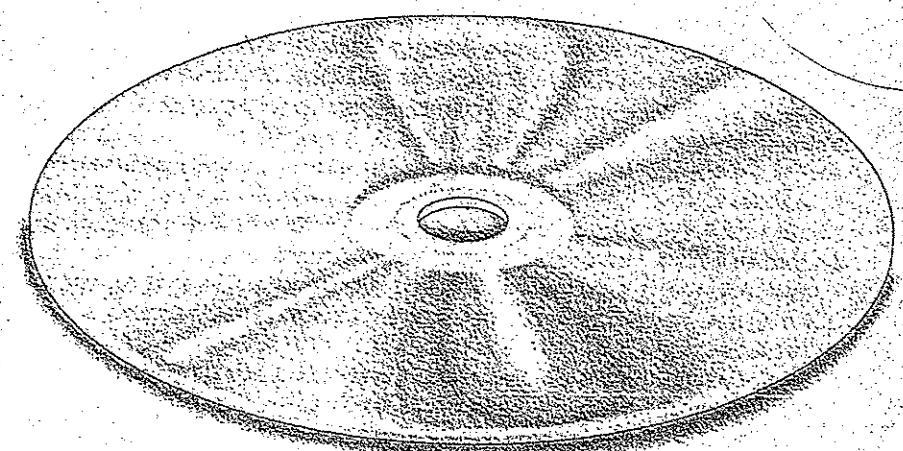
Nombre identificativo que se le da a algún elemento de un programa en ensamblador para trabajar con él. Ejemplos de símbolos son los nombres de segmento, los de subrutina, las constantes simbólicas y las etiquetas.

SUBRUTINA:

Pequeño fragmento de programa que recibe unos datos y devuelve uno o varios resultados, y que puede ser llamada desde puntos diferentes de un mismo programa.

WORD:

Cantidad de memoria equivalente a dos bytes consecutivos y que puede almacenar valores binarios de 16 bits.



Contenido del CD-ROM

En el CD-ROM que se incluye con este libro se incluyen un montón de utilidades de ayuda a la programación en ensamblador, divididos en distintas categorías (rutas, desensambladores, editores, documentación, ejemplos...) necesarias para cualquier programador. A continuación aparece una lista de este software:

DIRECTORIO CÓDIGO: Rutinas de ayuda a la programación en ensamblador.

386POWER. 386Power 2.00. Extensor del DOS para crear programas de 32 bits en modo protegido para procesadores 386 y superiores. Incluye rutinas para la creación de juegos.

3DVECT 3D. Vector Engine. Rutinas para dibujar escenas tridimensionales con mapeado de texturas, iluminación y sombreado Gouraud en modo protegido.

80X393. 80X393. Serie de textos y rutinas útiles para programadores en ensamblador.

AFLOAT. FLOAT. Librería para realizar cálculos en coma flotante desde programas realizados en ensamblador que no utilicen el coprocesador matemático.

AMACROS. A-MACROS. Conjunto de macros para lenguaje ensamblador que permiten utilizar construcciones de programación estructurada propias de los lenguajes de programación de alto nivel.

AMIS351. AMIS 3.5.1. Sistema para la creación de programas residentes en memoria.

ASM32. AMS32 3.0. Extensor de DOS para programación en 32 bits y rutinas para manejo de gráficos, menús, ventanas, disco, teclado e información del sistema.

ASMBASIC. Assembler BASIC. Rutinas en ensamblador para utilizar en programas en lenguaje BASIC. Incluyen manejo de memoria e interrupciones, carga de programas, scroll, impresora, etc...

ASMKIT. Assembler Kit. Conjunto de rutinas útiles para la programación en ensamblador.

ASMLIB40. ASMLIB 4.0. Las rutinas contenidas en ASM32, pero para programas realizados en 16 bits. Incluyen, además, acceso a memoria XMS y EMS.

ASMWIZ. Assembly Wizard's Library v1.6. Librería de rutinas para crear programas COM que abarcan gráficos, texto, ficheros, teclado, ratón, cadenas de caracteres, temporizadores, etc...

COREAID. CoreAids MASM Library Utilities. Conjunto de más de 50 rutinas para el manejo de memoria, entrada de datos, conversiones hexadecimales y ASCII, etc...

CPUTYPE. CPU Type. Rutina para averiguar el tipo de procesador instalado en el ordenador en que se está ejecutando el programa.

DOS32V12. Dos32 1.2. Extensor del DOS para crear programas de 32 bits en modo protegido.

FREELIB. FreeLib. 170 rutinas que contienen más de 17000 líneas de código de utilización libre en programas realizados en ensamblador.

MATH64. Math 64. Rutinas rápidas de operaciones aritméticas con cantidades de 64 bits para procesadores 386 y superiores.

MODEX. Mode X. Rutinas para el manejo de gráficos en modo X.

PSP. PSP Routines. Conjunto de funciones para el manejo del PSP.

SNIPPETS. Snippets. Multitud de rutinas para utilizar desde programas en ensamblador.

STDLIB. StdLib. Las rutinas de la librería estándar del lenguaje C para utilizarlas desde ensamblador.

TBONES. T-Bones 0.7. Base para la creación de programas residentes.

TSRTOO16. TSR Tool 1.6. Rutinas para la creación de programas residentes que necesitan llamar a funciones del MS-DOS, ratón o sonido.

WINDOWS. Windows. Rutinas para utilizar ventanas en modo texto desde ensamblador.

XLIB51. XLIB 5.1. Serie de rutinas para simplificar la programación de aplicaciones en modo protegido desde DOS. Soporta además funciones de depuración.

DIRECTORIO DESENSAM: Desensambladores y depuradores.

2ASM. 2ASM. Sencillo desensamblador de programas. Incluye el código fuente.

ASMGEN3. ASMGEN 3.0. Genera el código desensamblado de cualquier fichero de hasta 64 Kb. Además genera una lista de referencias cruzadas.

BUBBLE. The Bubble Chamber. Desensamblador que diferencia las zonas de datos de las de código del programa del que se desea obtener el código fuente.

BXD2. BrandX Symbolic Debugger 2.6 . Depurador de pantalla completa en modo texto que permite ejecutar programas sin que se pierdan los datos que envíen a la pantalla.

D86V402. D86 4.02. Depurador para los programas creados con el ensamblador shareware A86.

DMPPRG21. DumpProg 2.10. Desensambla un fichero ejecutable a partir de la información contenida en el archivo .MAP generado por el enlazador.

ID12. Intelligent Disassembler 1.2. Rápido desensamblador de programas que genera código ensamblador.

IDA35B. Interactive Disassembler 3.5. Completo desensamblador que soporta un gran número de procesadores y sistemas operativos. Incorpora un potente modo interactivo que permite guiar el desensamblado.

MD86. Masterful Disassembler 8086. Desensamblador que genera el código en ensamblador de cualquier programa.

OBJ2ASM. OBJ To ASM. Genera el código ensamblador a partir de un fichero de tipo OBJ.

RES86. RES 86 Disassembler. Desensamblador realizado en lenguaje ensamblador. Se incluye su código fuente.

SPYTRAK. Spy Trak. Desensamblador residente que realiza su trabajo siguiendo paso a paso la ejecución del programa y los valores que van tomando los registros del procesador.

SSD40. Servile Software Decoder 4.0. Depurador de programas que además permite desensamblar el código y proteger el disco contra escritura.

TRACE41. Trace 4.1. Desensamblador/depurador de programas que guarda en un fichero las instrucciones que va ejecutando.

ZD86. Zany Debugger versión 1.01. Depurador que soporta los tres ensambladores más extendidos: MASM, TASM y A86. Cuenta con posibilidad de desensamblado simbólico.