Maynard Kong

Lenguaje Ensamblador Macro Assembler

PONTIFICIA UNIVERSIDAD CATOLICA DEL PERU
FONDO EDITORIAL 1989

Maynard Kong. En 1964 ingresó a la Facultad de Ciencias Físicas y Matemáticas de la Universidad Nacional de Ingeniería. Egresó en 1968 y desde 1969 se ha desempeñado como profesor del Departamento de Ciencias de la Universidad Católica en cursos de Matemáticas de niveles y especialidades variados. Obtuvo el grado de doctor (PhD) en la Universidad de Chicago (Estados Unidos de América) en 1976. Fue profesor visitante en la Universidad de Stuttgart (República Federal de Alemania) en 1979, y al mismo tiempo becario de la Fundación von Humboldt en un programa de posdoctorado, y posteriormente, también en Venezuela, durante 4 años.

Ha publicado importantes trabajos de investigación y varios textos de consulta universitaria, entre los que se pueden mencionar: Teoría de conjuntos *(coautor con César Carranza),* Basic, Cálculo diferencial, Cálculo integral, Lenguaje de programación Pascal, *y* Lenguaje de programación C.

Ha participado en numerosos eventos de Matemáticas, promoción de las Ciencias Básicas e Informática tanto en el país como en el extranjero.

LENGUAJE ENSAMBLADOR MACRO ASSEMBLER

Maynard Kong

Lenguaje Ensamblador Macro Assembler

PONTIFICIA UNIVERSIDAD CATOLICA DEL PERU
FONDO EDITORIAL 1989

Primera edición, setiembre de 1989

Cubierta: Carlos González R.

Lenguaje ensamblador Macro Assembler

Impreso en el Perú - Printed in Peru

P R O L O G O

Este libro desarrolla los conceptos del lenguaje de programación ensamblador MACRO ASSEMBLER para los computadores que usan los procesadores de la familia 8086 con el sistema operativo MSDOS/IBM DOS.

La mayor parte de los ejemplos se exponen como programas completos e independientes -en algunos casos se repiten secciones de otro programas- y por lo tanto pueden ser utilizados directamente en la generación de sus correspondientes versiones ejecutables, para cuyo propósito el lector debe utilizar:

1) Un editor o procesador de textos a fin de obtener el programa texto fuente;

2) un programa compilador de este lenguaje ensamblador, por ejemplo: MASM.EXE, ASM.EXE, TASM.EXE, etc.

y 3) un programa enlazador o montador de módulos objetos tal como LINK.EXE.

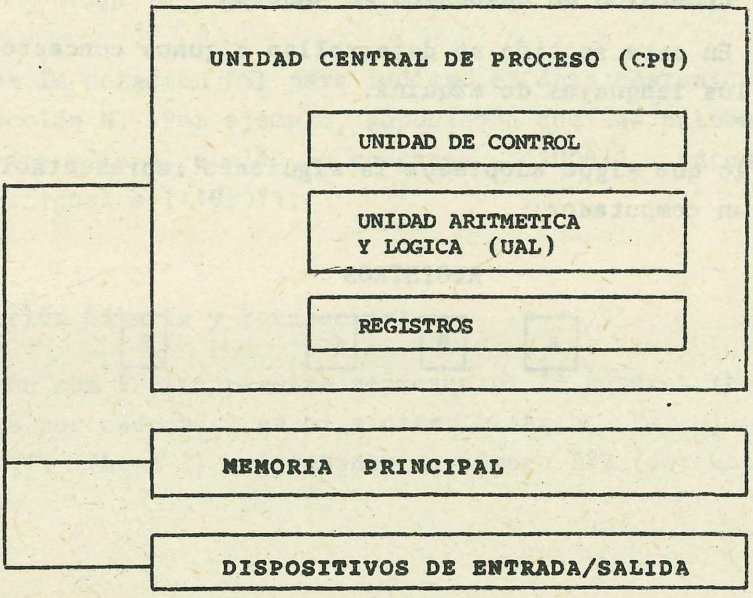
También es muy conveniente el poder disponer de algún programa "depurador", como DEBUG, SYMDEB, etc, para observar el proceso de ejecución de un programa ejecutable, examinar y modificar datos de registros o de memoria, facilitando así la tarea de búsqueda de las posibles fuentes de errores.

Maynard Kong

INTRODUCCION

1.1. ORGANIZACION DE UN COMPUTADOR

Desde un punto de vista lógico la organización de un computador admite el siguiente esquema:



Tanto los datos como las instrucciones de un programa son almacenados en la memoria principal.

La unidad central de proceso se compone de:

1) la unidad de control, que se encarga de traer instrucciones, descodificarlas y efectuar el control de las unidades del sistema;

2) la unidad aritmética y lógica, que realiza las operaciones aritméticas y lógicas. En ella se encuentra un conjunto de registros, que son áreas de memoria para almacenar datos intermedios o realizar funciones de control. Estos registros operan a altas velocidades;

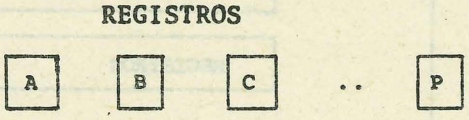
3) la memoria principal que sirve para almacenar tanto las instrucciones como los datos del programa;

4) dispositivos de entrada y salida, que son los medios con los cuales la unidad central de proceso hace la transferencia de datos.

1.2. ELEMENTOS DE LENGUAJES DE MAQUINA

En esta sección se desarrollan algunos conceptos básicos de los lenguajes de máquina.

En lo que sigue adoptamos la siguiente representación lógica de un computador:



MEMORIA

DIRECCION	CONTENIDO
0	101011 ... 010
1	010110 ... 110
.	.
.	.
.	.
C - 1	101010 ... 101

La memoria puede ser considerada como una sucesión de celdas contiguas o **palabras** cada una identificada por un número, llamado **dirección.** Cada palabra consiste de una cantidad fija L de bits (ceros o unos) que depende del sistema. L recibe el nombre de longitud de palabra del computador. Por ejemplo, si la longitud de palabra es 8, cada palabra podrá almacenar un byte u octeto:

bit	7	6	5	4	3	2	1	0
	1	1	1	0	0	0	1	1
valor	128	64	32	16	8	4	2	1

Emplearemos la notación [N] para indicar el dato contenido en la dirección N. Por ejemplo, suponiendo que las palabras son de 8 bits y la dirección 34 contiene 11100011, entonces [34] es igual a 11100011.

Representación binaria y hexadecimal

La expresión con 8 bits permite representar 2^8 números (hay dos valores por cada bit) en el sistema binario. Por ejemplo, 11100011 (base 2) representa al número 227 (decimal)

que se obtiene sumando aquellas potencias de 2 asociadas con cada dígito 1: 128 + 64 + 32 + 2 + 1 = 227.

Para abreviar la representación en binario, se suele emplear el sistema de numeración de base 16 o hexadecimal, en donde los dígitos son 0, 1, ... 9, A, B, C, D, E y F con A = 10, ..., y F = 15. Es muy fácil pasar de un sistema a otro. En efecto, para la conversión es suficiente observar que un dígito hexadecimal equivale a (un número binario formado por) cuatro dígitos binarios o bits. Por ejemplo:

1110	0011	(binario)
C	3	(hexadecimal)

Así, 11100011 binario = C3 hex.

Espacio de direccionamiento

La dirección es un número entre 0 y C-1, en donde C es la capacidad de la memoria, o el número total de palabras que componen la memoria. El conjunto de todas las direcciones posibles se denomina **espacio de direccionamiento**. Todas las direcciones se expresan utilizando una cantidad fija N de bits, de modo que C es igual a 2 elevado a la N. Por ejemplo, para el computador 8086 de Intel las direcciones emplean 20 bits y la unidad de memoria es de 8 bits, de suerte que es posible contar con

$$2^{20} = 1\ 048\ 576 = 1 \text{ Mega octeto},$$

esto es, más de un millón de direcciones con un byte.

Los registros son unidades de memoria rápida, con una determinada longitud, en donde se realizan la mayor parte de las operaciones intermedias, sean de cálculos aritméticos, lógicos o de control de programa. Existen registros denomina

dos de **propósito general** para realizar todas estas funciones y también hay otros registros reservados para funciones específicas como, por ejemplo, para indicar la próxima instrucción a ser ejecutada, señalar el estado de los indicadores (flags) de operaciones y de control, o indicar determinadas áreas de memoria, etc ...

Instrucciones de máquina

Una instrucción de máquina puede tener la siguiente forma:

en donde

COD es el código de la instrucción,
REGISTRO es el número del registro utilizado,
OPERANDO puede ser una dirección de la memoria,
un dato particular u otro registro.

Tanto COD como REGISTRO son identificados por números o códigos asignados por el fabricante del procesador.

Por ejemplo, en el computador 8086, una instrucción para mover datos es la siguiente (en binario):

1 0 1 1 w reg dato
(mover dato inmediato a registro)

en donde w=0 si dato es octeto y w=1, si dato es de 16 bits (2 octetos o word).

Entonces para mover el octeto 1 al registro AH, de código 100, se tendrá la siguiente instrucción de máquina:

1 0 1 1 0 1 0 0 0 0 0 0 0 0 0 1 (binario)

ó B401 = B 4 0 1 (hexadecimal).

Nótese que la instrucción emplea o requiere 4 octetos.

Esta instrucción puede ser escrita usando un lenguaje ensamblador, tal como MASM o ASM de Microsoft, empleando una palabra más fácil de recordar (palabra mnemónica) como MOV (mover):

MOV AH,1 (mover 1 inmediatamente al registro AH).

Por supuesto hay otras instrucciones de máquina que tienen formas distintas a la mostrada.

Modos de direccionamiento

Según el número de direcciones que una instrucción requiere para acceder a un dato se obtiene una clasificación de las instrucciones.

a) Direccionamiento directo o absoluto. En este caso la instrucción contiene la dirección del dato sobre el cual ha de hacerse la acción.

   Por ejemplo, MOV AX, [0150]

   se usa para mover al registro AX el contenido de la dirección 0150.

b) Direccionamiento inmediato. En este caso, el operando, llamado **operando inmediato**, es en realidad el dato mismo.

   Por ejemplo, la instrucción MOV AH,1

Hay otros modos de direccionamiento (llamados indirectos) en donde la instrucción requiere más de una dirección para acceder al dato.

1.3. BREVE DESCRIPCION DEL 8086

a) Registros

El procesador 8086 de Intel contiene 14 registros de 16 bits:

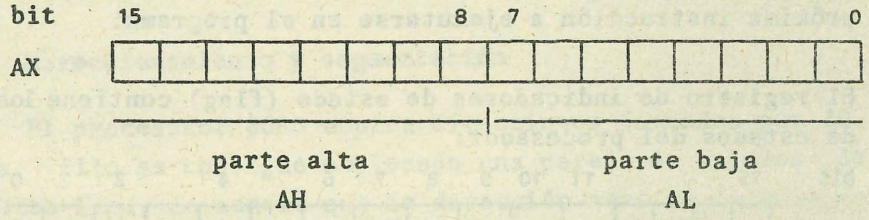
- de propósito general	AX	BX	CX	DX
- de segmentos	CS	DS	SS	ES
- punteros de pila	SP	BP		
- de índices	SI	DI		
- puntero de instrucciones	IP			

- un registro de indicadores de estado (flags), sin nombre lógico.

A su vez cada registro de propósito general se subdivide en dos registros de 8 bits que pueden ser accedidos separadamente:

registros altos (high)	AH	BH	CH	DH
registros bajos (low)	AL	BL	CL	DL

El registro AX se denomina acumulador de 16 bits y se compone de los registros de 8 bits AH y AL, (el acumulador de 8 bits):



Algunas operaciones usan exclusivamente estos registros.

De igual modo se descomponen los otros registros de propósito general.

En general se emplea la letra X para designar un registro extendido y las letras H y L para designar las partes alta y baja de tales registros, respectivamente.

El registro BX, llamado base de propósito general, se usa frecuentemente como registro de base para los direccionamientos.

El registro CX, contador, se emplea a menudo para realizar operaciones de conteo, iteraciones con las cadenas y rotaciones.

El registro DX, de datos, se usa algunas veces como una extensión del acumulador, por ejemplo para las multiplicaciones y divisiones, como veremos más adelante.

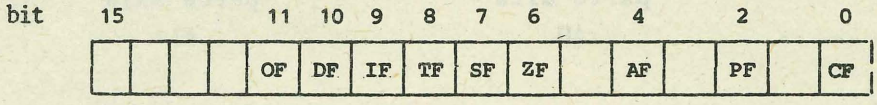
Los registros de segmento CS, DS, SS y ES, llamados de segmento de código, datos, pila y extra, sirven para indicar las direcciones de tales segmentos o áreas de memoria de trabajo.

Los registros SP y BP están asociados al segmento de pila.

Otros registros punteros de datos son los de índices: SI y DI.

El registro IP, puntero de instrucciones, apunta o señala la próxima instrucción a ejecutarse en el programa.

El registro de indicadores de estado (flag) contiene los bits de estados del procesador:



en donde:

bit			
0	CF =	indicador de acarreo	(carry flag)
2	PF =	indicador de paridad	(parity flag)
4	AF =	indicador auxiliar	(auxiliary flag)
6	ZF =	indicador de cero	(zero flag)
7	SF =	indicador de signo	(sign flag)
8	TF =	indicador de desvío	(trap flag)
9	IF =	indicador de interrupción	(interrupt-enable flag)
10	DF =	indicador de dirección	(direction flag)
11	OF =	indicador de desbordamiento	(overflow flag)

b) Palabras de Memoria

La longitud de palabra de memoria del 8086 es de 8 bits, de modo que cada dirección contiene un octeto o byte.

Cada octeto se puede expresar con dos dígitos hexadecimales, de manera que -usando el sistema hexadecimal- puede representar un número entre 0 y FF. Se suele emplear el término **palabra** (word) para referirse a un número formado por dos octetos, por lo que su representación requiere un máximo de 4 dígitos hexadecimales: 0 a FFFF (hexadecimal), esto es, de 0 a 65535 (64K -1, siendo K=1024).

c) Direccionamiento y segmentación

El procesador 8086 emplea direcciones formadas por 20 bits. Ello se consigue empleando una pareja de valores de 16 bits I : J, de manera que la dirección real se calcula mediante la expresión:

DIRECCION REAL = I * 10 + J (en hexadecimal)

en donde 10 (hexadecimal) es el número 16 en decimal,
I se denomina dirección del segmento
J se denomina desplazamiento (OFFSET) relativo al segmento.

Así, por ejemplo, la dirección real definida por el par 3A05:2C35 es el número

$$3A05 * 10 + 2C35 = 3A050 + 2C35 = 3CC85$$

en donde los datos y cálculos se han expresado en el sistema hexadecimal.

El espacio de direccionamiento tiene una estructura de segmentación. Esto significa que se compone de áreas de memoria llamadas **segmentos** que pueden variar en tamaño desde 0 a 64K octetos. Un segmento arbitrario empieza en cualquier múltiplo de 10 (hex) y distintos segmentos se pueden trasla par o no tener áreas comunes. De esta manera, si se desea, es posible utilizar distintas áreas de trabajo: una para el código del programa (segmento de código), otra para los datos (segmento de datos), otra para la pila (segmento de pila) y otra para el segmento extra de datos, cuyas direcciones se guardan en los registros correspondientes: CS, DS, SS y ES.

1.4. LENGUAJE ENSAMBLADOR DEL 8086

Las instrucciones de un lenguaje de máquina se expresan en forma numérica por lo que resulta complicado y tedioso el escribir programas en este lenguaje. Un lenguaje ensamblador nos permite utilizar expresiones mnemónicas para referirnos a las instruciones del lenguaje de máquina y también valernos de símbolos o nombres para designar datos, direcciones, variables, etc. Para un mismo procesador pueden disponerse de varios lenguajes ensambladores. En esta exposición sobre el computador 8086 emplearemos los lenguajes ensambladores escritos por Microsoft (Assembler o Macro Assembler de Microsoft) que se ejecutan bajo control del sistema operativo de la misma firma.

Ejemplos:

1) MOV (mover datos)
2) CMP (comparar)
3) INC (incrementar)
4) JMP (saltar)
5) JE (saltar si datos son iguales)
6) CALL (llamar subrutina)
7) XCHG (intercambiar datos)
8) ADD (sumar)
9) SUB (restar)

Notación.

1) Para indicar un número en el sistema hexadecimal el lenguaje ensamblador MASM (o ASM) utiliza el sufijo h (ó H): 13h, 2A4H. Y es necesario precederlo por cero si el número hexadecimal empieza con uno de los dígitos A, B, ..., F, para evitar confusiones con otros símbolos: 0Ah, 0F3A2h

2) Se emplea la notación S:[D] para indicar el contenido de la dirección de S:D. En muchas instrucciones se omite S: , cuando S se refiere a uno de los registros de segmentos como DS, SS, etc, pues por defecto la dirección del segmento queda determinada por la misma instrucción teniendo en cuenta la forma de los datos.

Ejemplos:

```
MOV BYTE 1000:[0120],5h   ; mueve el octeto 5 a la dirección
                            1000:0120
MOV BYTE [0120],5h   o  MOV BYTE DS:[0120],5h
CMP AX,[BP] o CMP AX,SS:[BP]  ; compara contenidos (tipo word) de
                                AX y SS:[BP]
```

Funciones del sistema operativo DOS

El sistema operativo DOS tiene codificadas un conjunto de funciones de servicios del sistema denominadas funciones del DOS, por ejemplo:

CODIGO DE FUNCION (en base 16)	**EFECTO**
1	Leer carácter de teclado con eco
2	Imprimir carácter en pantalla
5	Imprimir carácter en impresora
9	Imprimir cadena en pantalla
2A	Obtener fecha
3D	Abrir archivo
40	Escribir en archivo/dispositivo
4C	Terminar proceso

Para usar una función se almacena el código correspondiente en el registro AH y se ejecuta la instrucción de interrupción 21h: INT 21h.

Por ejemplo, una manera de finalizar un programa (y retornar al DOS) consiste en tener 4Ch en AH y ejecutar INT 21h. Otra forma de hacerlo es simplemente con INT 20h.

1.5. PROGRAMA: Imprime caracteres

Escribir un programa que imprima los caracteres A,...,F.

Los códigos ASCII de los caracteres indicados son 41,... , 46 (hexadecimal).

Para imprimir un carácter se requiere tener su código ASCII en el registro DL, el número 2 (función del DOS para imprimir caracteres) en el registro AH y ejecutar la instrucción INT 21h.

Las instrucciones que el programa debe cumplir son:

(1) Poner en el registro DL el valor 41h
(2) Imprimir el carácter en DL
(3) Comparar el contenido de DL con 46h (valor límite)
(4) Si son iguales saltar a (7) para finalizar el programa
(5) Aumentar en 1 el valor de DL (pues es menor que el límite)
(6) Saltar a (2)
(7) Finalizar.

El código correspondiente en lenguaje ensamblador es:

```
INSTRUCCIONES                COMENTARIOS

MOV DL,41h          ;  (1)  mover el octeto 41h a DL
MOV AH,2h           ;  (2)  función para imprimir
INT 21              ;       ejecutar
CMP DL,46h          ;  (3)  comparar con límite
JE  dirección de (7)  ;  (4)  (jump equal) saltar a la dirección de
                    ;       la intrucción (7) si son iguales
INC DL              ;  (5)  incrementar DL en una unidad
JMP dirección de (2)  ;  (6)  saltar a la dirección de la instruc-
                    ;       ción (2)
INT 20h             ;  (7)  finalizar proceso
```

Ahora escribiremos este programa usando el programa SYMDEB (o DEBUG).

PASO 1. Corra el programa SYMDEB, por ejemplo desde el disco A:, ingresando:

```
A>SYMDEB <enter>
```

Aparece el símbolo - , lo que indica que el programa se está ejecutando.

PASO 2. Ingrese el comando R (para ver los contenidos de los registros)

```
- R <enter>
```

Debe aparecer una tabla parecida a la siguiente:

```
AX=0000  BX=0000  CX=0000  DX=0000  SP=9123  BP=0000 SI=0000  DI=0000
DS=6DCC  ES=6DCC  SS=6DCC  CS=6DCC  IP=0100  NV UP EI PL NZ NA PO NC
6DCC:0100 014389          ADD [BP+DI-77],AX                SS:FF89=DA8C
```

Se observan los valores de los distintos registros, por ejemplo CS es 6DCC (hexadecimal), y también los valores de los bits indicadores de estado: NV (no desborde), UP (índices crecientes), EI (interrupciones permitidas), PL (positivo), NZ (no cero), NA (no acarreo auxiliar), PO (paridad impar) y NC (no acarreo).

El par CS:IP contiene la dirección de la próxima instrucción a ser ejecutada, esta dirección es 6DCC:0100. Los valores del registro puntero de instrucciones IP son relativos al segmento dado por CS (en su programa el valor de CS puede ser diferente).

PASO 3. Ingrese ahora las siguientes instrucciones usando el comando A (assemble) para ensamblar instrucciones a partir de la dirección 0100 (ó CS:0100):

```
- A 0100 <enter>

6DCC:0100   MOV  DL,41     <enter>
6DCC:0102   MOV  AH,02     <enter>
6DCC:0104   INT  21        <enter>
6DCC:0106   CMP  DL,46     <enter>
6DCC:0109   JE   0100      <enter>     ← valor temporal
6DCC:010B   INC  DL        <enter>
6DCC:010D   JMP  0104      <enter>
6DCC:010F   INT  20        <enter>
6DCC:0111                  <enter>
```

Puesto que no se sabe de antemano -a menos que se calcule- la dirección de la instrucción INT 20, con la que se termina el programa, puede ponerse temporalmente cualquier valor, por ejemplo 100 : JE 100. Una vez que se ingresen todas las instrucciones se conocerá la dirección del salto -esto es, el desplazamiento- de la última instrucción, y se puede volver a escribir la instrucción JE con la dirección correcta 010F. Así, a continuación ingrese:

```
- A 109  <enter>
6DCC:0109  JE 10F  <enter>
                   <enter>
```

Para verificar que las instrucciones han sido escritas en forma correcta use el comando U (unassemble) que permite desensamblar un intervalo de direcciones:

```
- U 100 10F <enter>        [equivale a  U  CS:100  CS:10F ]

6DD0:0100 B241            MOV  DL,41                    ;' A'
6DD0:0102 B402            MOV  AH,02
6DD0:0104 CD21            INT  21
6DD0:0106 80FA46          CMP  DL,46                    ;' F'
6DD0:0109 7405            JZ   010F
6DD0:010B FEC2            INC  DL
6DD0:010D EBF5            JMP  0104
6DD0:010F CD20            INT  20
6DD0:0111 0106FE06        ADD  [06FE],AX
```

Observe que entre una dirección y su correspondiente instrucción en ensamblador aparece la instrucción codificada numéricamente. Por ejemplo, para CMP DL,46 se tiene 80FA46 (hexadecimal). También es de notar que en lugar de JE 10F aparece otra equivalente JZ 010F (jump if zero).

PASO 4. Corra el programa empleando el comando G (go):

```
- G  <enter>
```

Y se imprime:

```
ABCDEF
Program terminated normally (0)
```

PASO 5. El programa ingresado puede ser almacenado como un programa de tipo COM. Le daremos el nombre PROG1.COM, por ejemplo.

Ingrese el comando N (name) para nombrar archivos seguidos del nombre elegido:

```
-N PROG1.COM <enter>
```

El programa comprende las instrucciones en el intervalo de direcciones 100 - 112, relativas a CS, y éstas son las que serán almacenadas. Es preciso que IP tenga su valor igual a 100 (Si no lo tiene, ingrese -R IP 100) y que el par BX:CX contenga el número de octetos que componen el programa: desde 100 hasta 111 hay 11 (hexadecimal) direcciones u octetos. Por lo tanto debe ponerse BX=0 y CX=11.

Escriba en el registro CX el valor 11:

```
-R CX 11  <enter>
```

Compruebe tales valores con -R <enter>.

Grabe el programa PROG1.COM con el comando W (write):

```
-W  <enter>
```

Termine la sesión de trabajo con SYMDEB escribiendo Q (quit):

```
-Q
```

Ahora se dispone del programa PROG1.COM (con 11h = 17 bytes) que puede ser corrido directamente:

```
A>PROG1 / <enter>
```

1.6. ALGUNOS COMANDOS DEL PROGRAMA SYMDEB (o DEBUG)

1) Para ensamblar instrucción en una dirección: - A dirección

Ejemplos:

- A 400:10D <enter> [ensamblar a partir de 400:10D]
- A DS:0 <enter> [ensamblar a partir de DS:0]

También es posible almacenar datos en la memoria, por ejemplo:

-A 256 6 CS:256 <enter>

CS:010E DW 1F20 <enter> [se almacena la palabra 1F20]
CS:0110 DB 'Esta es una cadena' <enter>
[se almacenan todos los caracteres del texto en forma consecutiva]

- A <enter> [ensamblar a partir de CS:IP]

2) Para desensamblar instrucciones entre dos direcciones:
- U dirección1 dirección2

Ejemplo: - U 10D 11F [Equivale a U CS:10D CS:11F]

3) Para nombrar archivo: - N nombre_archivo

Ejemplo: - N P1.EXE

4) Para cargar en memoria el archivo nombrado con el comando N:
- L

5) Para salvar en archivo (nombrado con el comando N): - W

Se graban BX:CX instrucciones a partir de la dirección CS:IP.

6) Para mostrar los valores de los registros: -R

7) Para cambiar el valor de un registro: - R registro valor

Ejemplo: R AX 3F01

8) Para ejecutar paso a paso un programa presente en memoria (a partir del valor dado por el puntero de instrucciones IP) usar las órdenes T o P : - T

9) Para cambiar los valores de los bits de estado: R F

Ejemplo: R F <enter>
NV UP DI NZ NG AC PE NC - DN ZR <enter>

que cambia UP por DN (down) y NZ (no zero) por ZR (cero).

ALGUNAS INSTRUCCIONES DEL LENGUAJE ENSAMBLADOR

2.1 INSTRUCCIONES MOV, INT Y CMP

Instrucción MOV

Sirve para mover o copiar un octeto o una palabra de la fuente al destino. Tiene la siguiente forma: MOV destino, fuente en donde ambos operandos deben ser octetos (bytes) o dos octetos (words).

Ejemplos:

1) Mover valor inmediato a registro:

```
MOV CL,2h          ;     mueve octeto (8 bits) 02h
MOV DX,2h          ;     mueve palabra (16 bits) 0002h
```

2) Mover valor inmediato a memoria:

```
MOV BYTE PTR [0102] ,5h  ; mueve octeto 05h a la dirección DS:0102
```

La expresión BYTE PTR (apuntador de octeto) especifica que el movimiento es de tipo byte ya que también es posible mover dicho valor como dos octetos o word, en cuyo caso se emplea el modificador WORD PTR:

```
MOV WORD PTR [0102] , 5h ;  mueve 0005h a  DS:0102
```

Si LIMITE es el nombre de una dirección de palabra (dos octetos, definido por: LIMITE DW ...) entonces

```
MOV LIMITE,5h     ; mueve la palabra 0005h a LIMITE
```

Otro ejemplo es: MOV BYTE PTR [BP],23h
que copia (el octeto) 23h en la dirección SS:BP

3) Mover de un registro a otro:

```
MOV AH,DL         ;        octeto
MOV DS,AX         ;        palabra
MOV BP,SP
```

4) Mover de memoria a registro:

```
MOV BL, [0102]    ; mueve contenido de DS:0102 a BL
MOV CX,LIMITE     ; mueve contenido de LIMITE a CX
```

en donde en el primer caso se trata de bytes (el registro BL es de este tipo) y en el segundo de words (por CX y LIMITE, que debe ser también de tipo word).

5) `MOV CS:[BX], LIMITE ; mueve el valor de la variable de pala-`
`; bra LIMITE a la dirección CS:BX`

Nota: No se puede mover un dato al registro CS con MOV.

Instrucción INT n

Ejecuta la subrutina de número n (n varía entre 0 y 255) cuya dirección se guarda en 4 octetos a partir de la dirección absoluta 4*n. Por ejemplo, la dirección de la subrutina de INT 21h se almacena en las direcciones 84h -87h.

Indicadores de estados afectados: IF, TF

Ejemplos:
```
INT 20h     ; termina proceso y ejecuta retorno al pro-
              grama de nivel anterior
INT 21h     ; ejecuta las funciones del DOS
```

Función del DOS que imprime carácter en pantalla: 02h

Valores de ingreso: Registro DL = código ASCII de carácter
Registro AH = 2h

Se ejecuta con INT 21h

Función del DOS que imprime carácter en impresora: 05h

Igual que en el caso anterior con AH = 5h.

Instrucción CMP

Sirve para comparar dos valores: CMP valor1, valor2 y frecuentemente es seguida por instrucciones de saltos que dependen del resultado de la comparación.

valor1 puede ser: registro o memoria,
y valor2 puede ser: registro, memoria o valor inmediato;
sin embargo, no se pueden comparar dos valores de memoria.

Ejemplos:

```
1) registro, registro:    CMP  DL,CH
2) registro, inmediato:   CMP  AL,5h
3) registro, memoria:     CMP  AX,LIMITE
4) memoria, registro :    CMP [102],BX   ;  [102] es DS:[102]
5) memoria, inmediato:
   CMP BYTE PTR [102],5H                 ;  [102] es DS:[102]
   CMP WORD PTR [BP], 5H                 ;  [BP] es SS:[BP]
   CMP WORD PTR [BX+2],17H               ;  [BX+2] es DS:[BX+2]
   CMP WORD PTR ES: [SI], 0FFH
   CMP LIMITE, 56H
```

Después de ejecutarse la instrucción CMP valor1, valor2, algunos indicadores de estado reflejan el resultado de la comparación a través de la diferencia valor1 - valor2. Por ejemplo:

```
si valor1 = valor2 : se establece el indicador de cero ZF (valor 1)
si valor1 < valor2 : se establece el indicador de signo SF (valor 1), y
si valor1 > valor2 : se anula SF (valor 0)
```

2.2. INSTRUCCIONES DE SALTOS: JMP, JE, JNE, ...

Sirven para transferir la ejecución del programa a la instrucción de la dirección que se especifique.

Todas se escriben indicando a continuación la dirección de destino del salto (salto directo) o un operando que hace referencia a la dirección del salto (indirecto).

La dirección de destino puede ser dada mediante valores inmediatos, registros y datos de memoria. El lenguaje ensamblador permite el empleo de etiquetas o nombres para designar direcciones.

1) Salto incondicional : JMP destino (jump)

El salto puede ser corto, si la dirección de destino no excede de la distancia de -128 y 127 de la actual, cercano, cuando se salta dentro del mismo segmento de código, o lejano, si se salta a otro segmento, en cuyo caso cambia el valor de CS.

Para las siguientes instrucciones el salto debe ser corto.

2) Saltos condicionales: Estos saltos tienen lugar de acuerdo a los valores que asumen algunos indicadores de estados, por ejemplo, cuando son precedidos por CMP valor1, valor2, su efecto es el siguiente:

- Salto si son iguales : JE (jump if equal) ó JZ (jump if zero)
- Salto si son distintos: JNE (jump if not equal) ó JNZ (jump if nonzero)

Saltos según comparaciones de números interpretados sin signo:

- Salto si valor1 está arriba de valor2: JA (jump if above)
- Salto si valor1 está debajo de valor2: JB (jump if below)
- Salto si arriba o igual: JAE (jump if above or equal)
- Salto si debajo o igual: JBE (jump if below or equal)

También se dispone de un grupo de instrucciones de salto para números interpretados con signo, y saltos según algunos indicadores de estados : JC (jump if carry), JNC (jump if not carry), JG (jump if greater), JO (jump if overflow), etc.

La instrucción JE produce la transferencia si el indicador de cero ha sido establecido, por ejemplo, si son iguales los valores de una instrucción de comparación. De otra manera, no se ejecuta la transferencia.

Ejemplos:

```
1) JMP  300              ; salto a CS:300
2) JB   etiqueta_corta   ; salto a etiqueta_corta
3) JMP  BX               ; salto a CS:BX
4) JMP  [BX]   ; salto a dirección CS:dest, siendo dest=DS:[BX]
   Así, si BX=120 y DS:120 contiene 300, entonces el
   destino del salto es CS:300.
5) JMP  SS:[120]         ; salto a CS:dest, con dest=SS:[120]
```

Todos los saltos mencionados son de tipo cercano (no cambian el valor de CS). Más adelante se tratarán los saltos lejanos.

2.3. INSTRUCCIONES INC y DEC

Tienen la siguiente forma: INC operando o DEC operando y sirven para aumentar o disminuir en 1 el operando (registro o memoria)

Ejemplos:

```
INC DL          ;  aumenta en 1 el valor actual de DL
DEC ES:[120]    ;  disminuye en 1 el contenido de ES:120
```

2.4. INSTRUCCIONES DE SUMAR Y RESTAR: ADD, SUB, ADC, SBB

Las instrucciones para sumar y restar son:

```
ADD destino,fuente      (resultado=destino+fuente)
SUB destino,fuente      (resultado=destino-fuente)
```

También se tienen instrucciones para sumar y restar teniendo en cuenta el valor actual del indicador de acarreo (acarreo para la suma y préstamo para la resta):

```
ADC destino,fuente      (resultado=destino-fuente+acarreo)
SBB destino,fuente      (resultado=destino-fuente-acarreo)
```

en donde acarreo=1 ó 0, según CF haya sido establecido o nó.

En todos los casos el resultado se pone en el destino y quedan afectados los indicadores: AF, CF, OF, PF, SF y ZF.

Ejemplos:

```
ADD CL,0F3h     ;  suma el octeto 0F3h a CL
SUB BX,AX
ADD [34A],DX    ;  suma DX a DS:[034A]
ADC CX,LIMITE   ;  LIMITE es un área de memoria (varia-
                ;  ble)definida por:  LIMITE DW ...
SBB VALOR,10
SUB BYTE PTR [BX+DI],20
```

2.5. PROGRAMA: Usando MASM

Ahora mostramos cómo generar un programa usando MASM o ASM. Para ello se requieren los siguientes programas: Un editor de textos, MASM.EXE (o ASM.EXE) y LINK.EXE.

El siguiente programa imprime las letras A, ... , F, como el programa desarrollado anteriormente con SYMDEB.

PASO 1. Usando un editor de texto escriba el siguiente texto como un archivo de nombre PROG2.ASM (programa fuente)

```
CODIGO     SEGMENT
           ASSUME CS:CODIGO
COMIENZO:  MOV DL,41h          ; mover 41h (letra A)
           MOV AH,2h           ; función 2 para imprimir
OTRA_VEZ:  INT 21h             ; ejecutar
           CMP DL,46h          ; comparar DL con 46h (letra F)
           JE FIN              ; si son iguales saltar a FIN
           INC DL              ; es menor, incrementarlo
           JMP OTRA_VEZ        ; y saltar a OTRA_VEZ
FIN:       MOV AH,4Ch          ; código de fin de programa
           INT 21h
CODIGO     ENDS                ; fin del segmento CODIGO
           END  COMIENZO       ; fin del programa fuente
                               ; y define entrada de programa en la
                               ; etiqueta COMIENZO
```

PASO 2. Procesamiento con MASM y LINK

```
A> MASM  PROG2; <enter>     (produce PROG2.OBJ si no hay errores)
A> LINK  PROG2; <enter>
Se produce el programa PROG2.EXE que puede correrse con:
A> PROG2 <enter>
```

EXPLICACION

1. Un programa fuente en lenguaje ensamblador puede ser procesado por MASM se compone de segmentos de programas y termina con la directiva END.

   La definición de segmentos de programas para MASM se hace en la forma:

```
nombre    SEGMENT
          instrucciones (una por línea)
nombre    ENDS
```

   y tiene por propósito formar un bloque de instrucciones o de variables (datos) relativas a un mismo registro de segmento(CS,DS o SS).

   Un programa puede consistir de varios segmentos de programas. Estos pueden ser separados o encajados. Todos los segmentos con el mismo nombre son tratados como un único segmento.

2. Normalmente se emplean tres tipos de segmentos de programa:

   (1) para el código del programa (instrucciones), con dirección en el registro CS.

   (2) para los datos, con dirección en DS.

   (3) para la pila (stack), con dirección en SS

   En el programa anterior sólo se ha utilizado un segmento de programa, al que hemos denominado CODIGO.

   La directiva ASSUME CS: CODIGO

   instruye al programa ensamblador para que las direcciones de las variables y etiquetas definidas dentro del segmento de programa CODIGO queden referidas al registro de segmento CS. Por ejemplo, las etiquetas COMIENZO, OTRA_VEZ y FIN son direcciones con segmento en CS.

3. Una instrucción puede ser precedida por una etiqueta que es un nombre que representa la dirección de dicha instrucción:

   ETIQUETA: INSTRUCCION

   En este caso los dos puntos indican que se trata de una etiqueta cercana (Near), esto es respecto al segmento de programa en donde se define.

4. Al lado de END se escribe la dirección de entrada al programa que en este caso es dada por la etiqueta COMIENZO.

5. Se utiliza ";" para escribir comentarios en cada línea de programa: El texto restante en dicha línea es ignorado por el programa ensamblador.

2.6. EJERCICIOS

1. Escriba un programa que compare dos números en los registros AX y BX y que de acuerdo a esto imprima >, = o <.

2. Escriba un programa de manera que imprima las letras A, ...,G, una por línea.

   Sugerencia: Imprima los códigos 0Dh (13 = retorno de carro) y 0Ah (10 = avance de línea).

3. Escriba los programas anteriores de modo que la impresión se haga en la impresora.

4. Escriba un programa que imprima en la pantalla los dígitos 0, 1, ..., 9, uno por línea.

   Sugerencia: Los códigos ASCII de los dígitos son 30h, ..., 39h, respectivamente.

5. Obtenga una versión PROG2.COM con el programa PROG2.ASM.

   Sugerencia: Incluya en PROG2.ASM la directiva ORG 100h antes de la etiqueta COMIENZO. Procese de nuevo con MASM y LINK y corra el programa EXE2BIN:
   EXE2BIN PROG2.EXE PROG2.COM

2.7. ETIQUETAS

Una etiqueta (label) es un nombre que se emplea para indicar una dirección. Toda etiqueta tiene tres atributos: segmento, desplazamiento y tipo (BYTE, WORD, DWORD). Las etiquetas definidas en segmentos de código (asociados a CS) pueden ser a su vez de dos tipos: cercanas (NEAR) o lejanas (FAR) según el modo en que serán referidas posteriormente: sólo dentro del mismo segmento o desde otro segmento.

Ejemplos de etiquetas en segmentos de código

1) Las etiquetas cercanas pueden ser definidas usando dos puntos:

```
REPETIR : INC CX  ; REPETIR es una etiqueta cercana
```

2) Etiquetas definidas por procedimientos (PROC ...) :

```
CALCULAR   PROC NEAR  ; CALCULAR es una etiqueta cercana
CALCULARL  PROC FAR   ; CALCULARL es una etiqueta lejana
```

3) Etiquetas definidas con la directiva LABEL:

```
CAMBIAR   LABEL NEAR  ; etiqueta cercana CAMBIAR
CAMBIARL  LABEL  FAR  ; etiqueta lejana CAMBIARL
```

Las etiquetas se utilizan frecuentemente en las instrucciones de salto, por ejemplo:

```
JMP CAMBIAR   ;  CAMBIAR es una etiqueta cercana o lejana
JNE REPETIR   ;  REPETIR es una etiqueta cercana
```

o para llamar a un subprograma (subrutina): CALL CALCULAR.

2.8. SEUDO OPERACIONES

Para facilitar la redacción de programas con el lenguaje ensamblador se disponen de algunas expresiones y símbolos que no representan instrucciones de máquina sino directivas al programa ensamblador.

2.8.1 DECLARACION DE SIMBOLOS CONSTANTES: Con la directiva EQU se pueden nombrar expresiones numéricas constantes y textos de manera que el programa ensamblador remplace luego tales nombres por el valor de la expresión o por el texto correspondiente:

límite EQU 20*10 ; define la constante numérica límite igual a 200
mensaje EQU "Ingrese cantidad "; define mensaje igual a un texto

También se puede emplear el signo de igualdad (=) para definir (o redefinir) constantes numéricas (16 bits):
valor = 100.

2.8.2 DECLARACION DE DATOS: Se pueden usan las directivas DB (define byte), DW (define word) y DD (define double word) para reservar un área de memoria o bloque de direcciones formado por datos de uno, dos y cuatro octetos, respectivamente.

Es posible hacer reservaciones múltiples con el operador de duplicación DUP y asignar valores iniciales.

Un bloque puede ser identificado por un nombre, llamado variable (o arreglo si consiste de más de un dato).

Los operadores SEG y OFFSET pueden ser aplicados a una variable o a una etiqueta. Devuelven el segmento y el desplazamiento de estos objetos, respectivamente.

Ejemplos

```
1) BB   100h        ; define un bloque de 100h octetos sin nombrarlo
2) LIMITE DB 10h    ; la variable LIMITE es de un octeto o byte con
                    ; valor inicial 10h
3) CONTADOR DW ?    ; CONTADOR es una variable de una palabra sin va-
                    ; lor inicial
4) CASOS  DB 2,11,25,31 ; CASOS es un arreglo de 4 octetos al
        ; cual se le han asignado valores iniciales
        ; OFFSET CASOS representa el desplazamiento del arreglo
        ; CASOS respecto del segmento en donde se define
        ; CASOS[0],..., CASOS[3] (ó CASOS+0 = CASOS, ... ,CASOS+3)
        ; representan los contenidos respectivos.
```

```
5)  CASOS1 DB 4h DUP(?) ;CASOS1 es un arreglo sin valores iniciales

6)  PRUEBA DW 10h DUP(8h,2h);define 10h pares consecutivos de octetos
                            ;con los valores 8h,2h

7)  MENSAJE DB "ELIJA OPCION : "  ;  define una cadena de carac-
        ; teres es decir un bloque de octetos formado por los códigos
        ; ASCII de los caracteres E,L, ..., :,  y espacio en blanco.
        ; En este caso, por ejemplo, MENSAJE[5] representa al octeto
        ; 41h (código ASCII del carácter A).
```

Nota. En lugar de comillas(") se puede emplear apóstrofos (').

```
8)  TITULO DB "LENGUAJE ENSAMBLADOR  ",0Ah,0Dh,"$"
    ;0Ah y 0Dh son los códigos ASCII de los caracteres de avance de lí
     nea retorno de carro, y el signo $ se usa como terminador de cade
     na para imprimir en la pantalla con la función 09h del DOS

9)  MATRIZ DW 3h DUP ( 2h DUP (0)) ; define 3h grupos consecuti-
                                   ; vos de 2h DUP (0) palabras
```

Se ilustran algunas aplicaciones con estos datos:

```
(1) CMP CL,LIMITE
(2) MOV CONTADOR,CX
(3) MOV BX,OFFSET CASOS ; mueve desplazamiento de CASOS a BX
    MOV [BX+2], AL      ; mueve el contenido de AL al tercer compo
                        ; nente de CASOS
(4) MOV CL,CASOS[2]
(5) INC  CONTADOR
(6) MOV DX, OFFSET MENSAJE ; pone en DX desplazamiento de mensaje
(7) MOV AX, SEG LIMITE     ; carga segmento de LIMITE
    MOV DS, AX             ; en registro DS
```

(8) MOV LIMITE,7Fh ; mueve 7Fh a LIMITE (octeto)

(9) La siguiente sección de programa pone caracteres de blancos en todos los componentes del arreglo CASOS (se supone DS igual al segmento del arreglo) :

```
            MOV CX,0
            MOV BX,OFFSET CASOS
SIGUIENTE:  CMP CX,3
            JA  LISTO
            MOV [BX],' '      ; ó  MOV [BX],20h
            INC CX
            INC BX
            JMP SIGUIENTE
LISTO:
```

2.9. INSTRUCCIONES PARA MULTIPLICAR Y DIVIDIR ENTEROS (sin signo)

Instrucción MUL

La instrucción MUL admite las dos siguientes formas:

1) MUL operando de 8 bits

que multiplica el operando por AL y pone el producto en AX, y

2) MUL operando de 16 bits

que multiplica el operando por AX y deja el producto en el par de registros DX AX, con la parte de mayor orden en DX.

En ambos casos operando puede ser un registro o una dirección de memoria.

Los indicadores de estado CF y OF tienen el valor 0 si la parte de mayor orden es nula, y 1 de otra manera.

Ejemplos

1)
```
MOV AL,80H
MOV CL,0DH
MUL CL
```

da como producto el número 0680 en AX, con AH=06 y AL=80.

2) Si se ha definido la variable de palabra LIMITE, entonces

```
MUL LIMITE
```

multiplica su valor actual por AX y el producto (de 32 bits) se obtiene en los registros DX (parte de orden mayor) y AX (parte de orden menor).

3)
```
MUL BYTE PTR [BX+SI]
```

multiplica AL por el (octeto) contenido en la dirección dada por BX+SI, esto es DS:BX+SI.

Instrucción DIV

Da el cociente y el resto de la división de dos números y presenta dos formas:

1) Dividendo (de 16 bits) en AX y el operando divisor es un octeto:

```
DIV operando 8 bits
```

pone el cociente en AL y el resto de la división en AH.

2) Dividendo (32 bits) en el par DX AX , y el operando divisor es de 16 bits:

```
DIV operando 16 bits
```

calcula el cociente en AX y el residuo en DX.

Nota: Si el cociente excede la capacidad del registro receptor se obtiene un desbordamiento por división.

Ejemplos

1. Si AX y CL contienen los valores 0034h y 06h, respectivamente, entonces :

```
DIV CL
```

divide 34h entre 6h dando AX=0408h, con cuociente AL=08h y resto AH=04h.

2. Para dividir un octeto doble entre otro se pone el dividendo en AX y se hace DX igual a cero (extensión sin signo.

3. Si el cociente excede la capacidad de almacenamiento del registro receptor se produce un desbordamiento por división. Así por ejemplo, si AX tiene el valor 4000h y se divide entre el octeto 20h, el cociente es 200h que no puede ser almacenado en AL.

Nota. Para evitar el desbordamiento por división pueden compararse, antes de realizar la división, los valores de la parte superior S del dividendo con el divisor d: Habrá desbordamiento por división si y sólo si d está debajo de S. Por ejemplo (división de 32 bits):

```
CMP DX, divisor          ; DX es la parte superior del dividendo
JAE desbordamiento       ;
DIV divisor
```

2.10. FUNCIONES DEL DOS DE ENTRADA Y SALIDA

a) Función de impresión de carácter en pantalla : 02h

Se requiere:

código 02h en AH
código de caráter en DL

y se ejecuta con INT 21h.

b) Función de lectura de carácter de teclado con eco : 01h

Se requiere:

código 01h en AH

y se ejecuta con INT 21h.

Resultado: código de carácter leído se almacena en AL.

c) Función de impresión de carácter en impresora : 05h

Se requiere:

código 05h en AH
código de carácter en DL

y se ejecuta con INT 21h

d) Función de impresión de cadenas en pantalla : 09h

Se requiere:

código 09h en AH
dirección de (comienzo de la) cadena en DS:DX

y se ejecuta con INT 21h

Nota: Se imprimen los caracteres hasta encontrar el signo $, por lo cual, al usar esta función, la cadena debe terminar con dicho símbolo.

e) Función de lecturas de cadenas por teclado : 0Ah

Se requiere:

código 0Ah en AH
dirección de área de recepción o buffer en DS:DX
número máximo N de caracteres a ser leídos en la dirección inicial del área de recepción.

y se ejecuta con INT 21 h

Resultado: A partir de la dirección DS:DX+2 se almacenan consecutivamente los caracteres ingresados incluyendo al final el carácter 0Dh (retorno de carro). El número de caracteres ingresados, sin contar 0Dh, se pone en la dirección DS:DX+1.

Nota: N-1 es en realidad el máximo número de caracteres que se puede ingresar por el teclado. Además, N no puede ser mayor que 255.

Ejemplo

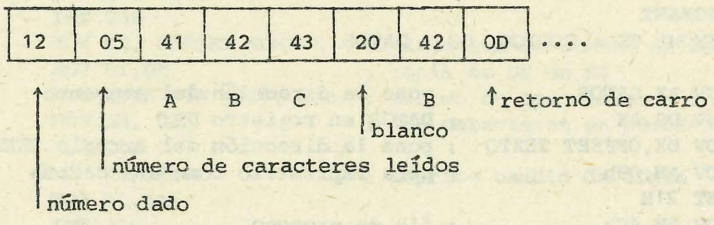
Si se ingresa la cadena:

ABC B<enter>

en respuesta al conjunto de instrucciones:

```
MOV DX,150h             ; buffer empieza en DS:DX = DS:0150
MOV BYTE [0150],12h     ; número máximo de caracteres = 18-1
MOV AH,0Ah              ; función de lectura
INT 21h                 ; ejecución
```

a partir de la dirección DS:0150 se tendrá:



2.11. SEGMENTO DE DATOS

Se suele emplear un segmento de programa para definir las áreas de datos con dirección en el registro DS. Con la directiva

ASSUME DS : Nombre del segmento de datos

se establece que el registro DS será asociado con este segmento.

El programa ensamblador asigna o almacena los segmentos de código y los de pila a los registros CS y SS, respectivamente, de modo que durante la ejecución del programa estos registros ya tienen las correspondientes direcciones de sus segmentos de programa. Por el contrario, al registro DS, no obstante la asociación indicada a través de ASSUME, no le es automáticamente dada la dirección de un nuevo segmento de datos y por lo tanto para accederlo es preciso asignarle dicha dirección, por ejemplo, en la forma:

```
MOV AX, nombre del nuevo segmento de datos
MOV DS,AX
```

en donde se podría haber usado otro registro general en lugar de AX.

El acceso a los datos de este segmento se puede hacer indirectamente a través de los registros BX y SI.

2.12. PROGRAMA: Imprime cadena

```
        ; versión 1
CODIGO  SEGMENT
        ASSUME CS : CODIGO, DS : DATOS
ENTRADA:
        MOV AX,DATOS            ; pone la dirección del segmento
        MOV DS,AX               ; DATOS en registro DS
        MOV DX,OFFSET TEXTO     ; pone la dirección del arreglo TEXTO en DX
        MOV AH,09h              ; para imprimirlo como una cadena
        INT 21H
        MOV AH,4Ch              ; fin de proceso
        INT 21H
CODIGO  ENDS

DATOS   SEGMENT
TEXTO   DB  "Esta cadena ha sido ",0Ah,0Dh
        DB  "definida en un segmento de datos$"
DATOS   ENDS
        END  ENTRADA
```

```
      ; versión 2: contiene datos en el segmento de código
CODIGO2   SEGMENT
          ASSUME CS:CODIGO2, DS:CODIGO2
ENTRADA2: MOV AX,CS
          MOV DS,AX            ; DS recibe dirección de CS=CODIGO2
          MOV DX,OFFSET TEXTO2
          MOV AH,09H
          INT 21H
          MOV AH,4CH
          INT 21H
TEXTO2    DB  "Imprime cadena, versión 2 ",0Ah,0Dh,"$"
CODIGO2   ENDS
          END ENTRADA2
```

2.13. PROGRAMA: Lee e imprime cadena

```
AREA_DATOS  SEGMENT
  MENSAJE DB "Escriba su nombre : $"
  NOMBRE  DB  40h DUP (?)     ;  40h octetos para lectura de cadena
AREA_DATOS  ENDS

PROGRAMA SEGMENT
         ASSUME CS:PROGRAMA, DS:AREA_DATOS
COMIENZO: MOV AX,AREA_DATOS       ; dirección de segmento de datos en DS
          MOV DS,AX
          MOV DX, OFFSET MENSAJE  ; imprime MENSAJE
          MOV AH,09h
          INT 21H
          MOV DX, OFFSET NOMBRE   ; dirección de NOMBRE en DS:DX
          MOV SI,DX               ; copia de DX en SI
          MOV BYTE PTR [SI], 31h  ; máximo de caracteres a leer=30h
          MOV AH, 0Ah             ; lee caracteres en DS:DX+2
          INT 21h
          MOV AH,02h              ; imprime cambio de línea
          MOV DL,0Ah
          INT 21H
          MOV DL,0Dh
          INT 21h

     ; Imprimir cadena leída: a  partir de NOMBRE+2 con  "$" al final.
     ; DX recibe comienzo NOMBRE+2
          MOV DX, OFFSET NOMBRE
          ADD DX,2
          MOV BL, BYTE PTR [SI+1]     ; número de caracteres leídos en BX
          MOV BH,0h
          MOV BYTE PTR [SI+BX+2],"$" ; $ al final de la cadena e imprimirla
          MOV AH,09h
          INT 21h
          MOV AH,4Ch                  ; fin de proceso
          INT 21h
PROGRAMA  ENDS
          END COMIENZO
```

2.14. PROGRAMA: Determina si número es primo

El siguiente programa determina si un número es primo. Este dato se halla almacenado en la variable NUM (que hemos supuesto igual a 25h = 37). El procedimiento es muy simple: Se prueba sucesivamente si NUM es divisible por BX=2,3,..., NUM-1; si esto sucede (resto de la división es cero) termina la prueba y se emite el mensaje : ES PRIMO, de lo contrario BX alcanza el valor NUM y se indica que el número es NO ES PRIMO.

En el programa se realizan divisiones de 16 bits y por lo tanto se almacena NUM en AX y se hace DX igual a 0.

```
DATOS           SEGMENT
   NUM          DW 25h              ; variable NUM contiene 37 (decimal)
   MENSAJE1     DB "ES PRIMO",0Ah,0Dh,"$"
   MENSAJE2     DB "NO ES PRIMO",0Ah,0Dh,"$"
DATOS           ENDS

PROGRAMA        SEGMENT
                ASSUME CS:PROGRAMA, DS:DATOS
COMIENZO:       MOV AX,DATOS
                MOV DS,AX

                MOV BX,02h
SIGUIENTE:      CMP BX,NUM
                JE  PRIMO           ; NUM es primo si BX=NUM
                MOV DX,0h           ; poner NUMERO en DX AX, con DX=0
                MOV AX,NUM
                DIV BX              ; división de 16 bits, resto en DX
                CMP DX,0h
                JE  NO_PRIMO        ; si resto es cero: saltar a NO_PRIMO
                INC BX
                JMP SIGUIENTE       ; volver a probar el siguiente

PRIMO:          MOV DX,OFFSET MENSAJE2    ; imprime MENSAJE1
                MOV AH,09h
                INT 21h
                JMP FIN

NO_PRIMO:       MOV DX,OFFSET MENSAJE2    ; imprime MENSAJE2
                MOV AH,09h
                INT 21h

FIN:            MOV AH,4Ch                ; fin de proceso
                INT 21h
PROGRAMA        ENDS

                END COMIENZO
```

2.15. EJERCICIOS

1. Compile el programa de impresión de cadena (Sección 12) con el nombre CADENA.EXE. Utilice el programa SYMDEB.EXE para procesarlo efectuando las siguientes acciones:

```
A>SYMDEB <enter>
- N CADENA.EXE <enter> {nombra el programa CADENA.EXE}
- L <enter>            { y lo almacena en memoria}
- U <enter>            { observe las instrucciones del programa }
                  { observe la primera instrucción: MOV AX, xxxx }
                  { xxxx es la dirección del segmento DATOS       }
                  { observe la cadena TEXTO almacenada ingresando}
- DA xxxx:0 <enter> {0 es el desplazamiento de TEXTO en DATOS }

- G <enter>     { corra el programa }
- R  IP 100 <enter>    { reinicie el puntero de instrucciones
                         IP para efectuar otra corrida}

- U <enter>           {observe otra vez las instrucciones }
- T <enter>           {ahora ejecute el programa paso a paso in-
                       gresando sucesivamente el comando Trace}
  ...
- Q <enter>           {termine SYMDEB con Quit}
```

2. Escriba un programa que lea una cadena de caracteres formada por dos dígitos, la convierta a número y determine si es primo. Por ejemplo, si se ingresa la cadena 37 (compuesta por los códigos ASCII de 3 y 7, es decir 51h y 55h, resp.) la conversión será:

   número = 10*(código de 3 - 48h) + (código de 7 - 48h)
   = 10*(51h-48h) + (55h-48h) = 10*3 + 7 = 37,

   en donde 48h es el código del carácter de 0.

3. Escriba un programa que lea dos cadenas y las compare, imprimiendo el mensaje apropiado.

4. Escriba un programa que cuente el número de palabras de una cadena leída durante corrida. Suponga que las palabras se separan sólo por caracteres de blancos.

5. Escriba un programa que imprima un número, contenido en CX, en el sistema hexadecimal. Escriba otro programa para hacer la impresión en el sistema decimal.

6. Escriba un programa que lea un número en el sistema decimal (máximo 5 dígitos), e imprima su raíz cuadrada entera, por defecto, en el sistema decimal.

7. Escriba todos los programas anteriores como programas de tipo .COM. (Véase Ejercicio 5 de la Sección 6).

SUBRUTINAS CERCANAS

3.1. SEGMENTO DE PILA

El segmento de pila (stack segment) es un área de memoria que se reserva para hacer almacenamiento temporal de datos (16 bits) o direcciones de retorno para las subprogramas (subrutinas). La dirección de este segmento se mantiene en el registro SS. El registro puntero de pila SP (stack pointer) contiene el desplazamiento de la cima de la pila cuya dirección absoluta es por tanto SS:SP. El registro de base de pila BP se utiliza para acceder a los datos del segmento de pila así: [BP], [BP+2], [BP+SI], etc, representan los datos de la dirección SS:BP, SS:BP+2, SS:BP+SI, respectivamente.

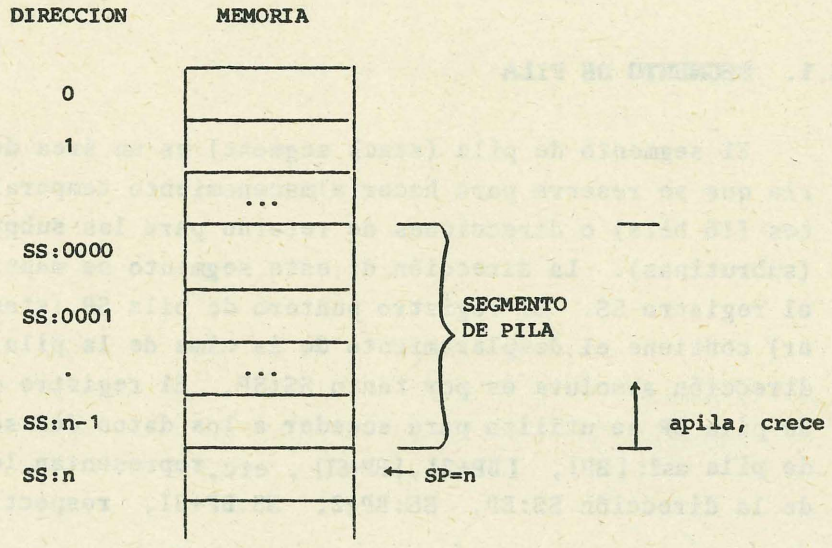
Un segmento de programa de tipo pila con n direcciones se puede declarar en la forma:

```
nombre    SEGMENT STACK
          DB  n    DUP (?)
nombre    ENDS
```

en donde:

- nombre es el identificador del segmento de programa de tipo pila
- n es el número de direcciones o tamaño del segmento de pila declarado (que estará comprendido entre las direcciones SS:0000 y SS:n-1).

En este caso el valor del registro SS es igual a la dirección del segmento de programa y SP es iniciado con el valor de n.



3.2. FUNCIONAMIENTO DE LA PILA

3.2.1 Apilamiento: PUSH

Con la instrucción: PUSH dato (16 bits)

se apila o pone el valor de dato sobre la cima de la pila, es decir SP disminuye su valor en dos unidades y se escribe el valor del dato en la dirección actual (con los octetos en orden invertido).

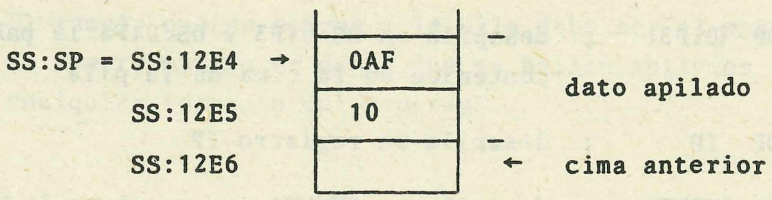
El dato puede ser un registro o un valor de memoria.

Ejemplos

1. Si AX = 10AFh y SP = 12E6h, entonces

PUSH AX

hace SP = 12E4 (= 12E6-2) y pone el valor de AX en SS:SP. Así, en la pila se tiene:



Nótese como se almacenan los datos de 16 bits en la memoria con los octetos en orden invertido.

2. PUSH [0120] ; apila los dos octetos ubicados en DS:0120
3. PUSH BP ; apila valor actual del registro BP
4. PUSH CS ; apila valor de registro CS
5. PUSH ES:[SI] ; apila palabra de la dirección ES:SI
6. PUSH CUENTA ; CUENTA es una variable de palabra
7. PUSH [BX+SI] ; apila palabra de DS:BX+SI
8. PUSH [BP] ; apila palabra de SS:BP
9. PUSH SI ; apila valor actual de SI

3.2.2 Desapilamiento: POP

Mediante la instrucción: POP destino (16 bits)

se desapila la palabra a la que apunta SP, esto es, se copia esta palabra en el destino (con los octetos en orden invertido) y SP aumenta su valor en dos unidades (la cima de la pila disminuye en dos unidades).

El destino puede ser un registro (distinto de CS) o un valor de memoria.

Ejemplos

1. Si la pila se encuentra como en la figura del ejemplo 1 anterior, la instrucción: POP BX

```
POP BX
```

pone SP en 12E6 y BX recibe el valor 10AF (exactamente el valor previamente apilado).

```
2. POP [01F3]   ; desapila en DS:01F3 y DS:01F4 la palabra
                ; contenida en la cima de la pila
3. POP  IP      ; desapila en registro IP
4. POP CUENTA   ; desapila en CUENTA que se supone de 16 bits
5. POP [BX-2]   ; desapila palabra en DS:BX-2
6. POP BP       ; desapila en BP
```

3.2.3 Acceso a los datos del segmento de pila

Los datos del segmento de pila pueden ser accedidos (sin cambiar la cima de la pila) a través del registro base de pila BP:

```
MOV AX,[BP]   ; mueve a AX los dos octetos ubicados en
              ; SS:BP
CMP BYTE PTR [BP+4],12h ; compara el octeto dado en
                        ; SS:BP+4 con 12h
```

3.2.4 Control del segmento de pila

Se indican las siguientes reglas:

1) Los datos almacenados mediante PUSH son estraídos con POP en orden inverso (el último se extrae primero).

2) Cuando se termine un proceso se debe cuidar que la cima de la pila se encuentre en el lugar en donde se empezó, para lo cual en muchos casos es suficiente que las instrucciones de apilamiento (PUSH o CALL) tengan sus correspondientes de desapilamiento (POP, RET, RET m), y en otros casos se puede mover manualmente la cima de la pila, por ejemplo con : ADD SP,4

3) El tamaño que se asigne a la pila debe ser al menos igual al número máximo de datos que se hallan apilados en cualquier instante del programa.

3.3. SUBRUTINAS: CALL, RET

Una subrutina o subprograma es una sección de programa, compuesta por instrucciones ejecutables (y posiblemente datos), cuya ejecución se invoca con la instrucción (de llamada) : CALL destino

y contiene al menos una instrucción de retorno: RET

para continuar con la instrucción que sigue a la de la llamada.

Usando MASM o ASM se puede formar subrutinas usando la directiva de formación de bloques o procedimientos PROC ... ENDP que tiene la forma:

```
NP      PROC [NEAR o FAR]
        instrucciones

        RET

        instrucciones
NP      ENDP
```

en donde:

- NP es el nombre o etiqueta del procedimiento
- NEAR o FAR especifica el tipo de llamada (y retorno) que tendrá el procedimiento: cercano o lejano según se realicen en el mismo segmento de código del procedimiento (intrasegmento) o desde otro segmento (intersegmento)

Las llamadas cercanas requieren sólo el desplazamiento de la subrutina (16 bits), en cambio las llamadas lejanas requieren además la dirección del nuevo segmento de código (en total 32 bits).

Nota: 1) Si se omite el tipo, se asume por defecto NEAR.

2) También se pueden formar subrutinas cercanas en la forma:

```
NP:
        instrucciones
        RET
```

en donde NP es igualmente un nombre o etiqueta (cercana)

Las llamadas pueden ser de dos clases: Directas, si el destino es la dirección de la subrutina, o indirectas, por medio de registros o datos de memoria, si el destino hace referencia a la dirección de la subrutina.

Ejemplos de llamadas cercanas

1) Directas:

```
CALL etiqueta cercana
CALL procedimiento cercano
```

2) Indirectas

```
CALL   BX              ; dirección de la subrutina en BX
CALL   WORD PTR [BX]   ; dirección se halla en DS:BX DS:BX+1
CALL   HACER           ; la variable HACER es de 16 bits y
                       ; contiene la dirección de entrada a subruti-
                       ; na cercana.
```

3.4. EJECUCION DE LLAMADAS CERCANAS

Cuando se ejecuta una llamada cercana, que siempre es relativa al segmento CS, se apila la dirección de la instrucción siguiente a la de la llamada y el puntero de instrucciones IP recibe la dirección de entrada a la subrutina. Al producirse el retorno de la subrutina (cuando se encuentra un RET), se desapila en IP la dirección anteriormente almacenada, de modo que el programa puede proseguir después del punto donde fue interrumpido por la llamada.

3.5. PROGRAMA: Imprime número en forma decimal

Escribir un programa que imprima un número contenido en la variable NUM en el sistema decimal.

En el programa se asigna el valor 4321 (decimal) a la variable NUM y se utiliza el método de encontrar los dígitos, a partir de las unidades de mayor orden, por divisiones sucesivas entre las potencias de 10:

```
4 = cociente entero de 4321/1000,  resto = 321
3 = cociente entero de 321/100,    resto = 21,
```

y así sucesivamente.

```
DATOS       SEGMENT
  NUM       DW 4321             ; número decimal de prueba
DATOS       ENDS

PILA        SEGMENT  STACK      ; segmento para apilamiento de datos
            DB 10h DUP(?)       ; y llamadas a subrutinas
PILA        ENDS


PROGRAMA    SEGMENT
            ASSUME CS: PROGRAMA, DS:DATOS
COMIENZO:
            MOV AX,DATOS        ; carga dirección de segmento de DATOS en DS
            MOV DS,AX           ;
            CALL IMPRIME_DEC    ; llamada a subrutina
            MOV AH,4Ch          ; fin de proceso
            INT 21h             ;

      ; ******  subrutina imprime en decimal ******
IMPRIME_DEC PROC                ; tipo cercano o NEAR
            MOV BX,10           ; base 10 en BX
            MOV AX,1            ; calcula menor potencia de 10 que exceda a NUM
REPETIR:    CMP AX,NUM
            JA LISTO
            MUL BX
            JMP REPETIR

LISTO:      DIV BX              ; potencia correcta en AX, con igual número
                                ; de dígitos que NUM
            XCHG AX,CX          ; ponerla en CX
            MOV AX,NUM          ; prepara división

DIGITO:     MOV DX,0            ; extensión de dividendo
            DIV CX              ; AX o AL contiene dígito buscado
            PUSH DX             ; salva resto de división en pila
            ADD AL,"0"          ; convertir a carácter de dígito
            MOV DL,AL           ; y pasarlo a DL
            MOV AH,2            ; para imprimir
            INT 21h
            POP AX              ; recupera resto en AX
            CMP AX,0
            JE  FIN             ; si es cero finalizar subrutina
            XCHG AX,CX          ; en caso contrario, intercambio para hallar
            MOV  DX,0           ;    siguiente potencia de 10 dividiendo  AX
            DIV  BX             ;    entre BX (10 decimal)
            XCHG AX,CX          ; datos preparados para tratar el siguiente
            JMP DIGITO          ;    dígito
   FIN:     RET                 ; retorno de subrutina
IMPRIME_DEC ENDP                ; fin de procedimiento

PROGRAMA    ENDS                ; fin de segmento PROGRAMA

            END COMIENZO        ; fin de programa (módulo) e indicación de
                                ;    entrada al programa.
```

COMENTARIOS

(1) En el programa:

- BX siempre contiene el valor 10 (decimal)
- CX almacena la potencia de 10 que tiene tantos dígitos como el valor de NUM y cada vez que se obtiene el dígito de orden mayor se divide entre BX (=10) para obtener la potencia inmediata inferior.

(2) En este programa hemos empleado la instrucción:

```
XCHG AX, CX
```

para intercambiar los valores de los registros AX y CX.

En general se puede usar esta instrucción con dos registros o con un registro y un dato de memoria. Se excluyen los registros de segmentos.

Ejemplos

```
XCHG AL,AH          ; intercambio de los octetos del acu-
                    ;   mulador AX
XCHG BX,BP
XCHG DI,DX
XCHG BL, 0120       ; intercambio de octetos
XCHG [BX+SI+3],AX
```

3.6. PROGRAMA: Calcula recursivamente factorial de un número

El siguiente programa solicita el ingreso de un número por el teclado, calcula su factorial e imprime este valor. El número se ingresa como una cadena compuesta por un carácter de dígito (por ejemplo, se ingresa el carácter "5", código ASCII 35h), que luego se convierte a número y se almacena en el registro CX. El programa contiene una subrutina cercana

llamada FACTORIAL para efectuar el cálculo, la cual se ejecuta en forma recursiva, es decir, se llama a sí misma:

```
FACTORIAL PROC NEAR
          ...
          CALL FACTORIAL
          ...
FACTORIAL ENDP
```

El texto del programa es:

```
DATOS SEGMENT
   RES            DW ?
   DIEZ           DW 10
   MENSAJE        DB "Ingrese número (1 - 8) : $"
   CAMBIO_LINEA   DB 13,10,"$"
   CADENANUM      DB 4  DUP(?)
   CADENARES      DB 10 DUP(?)
DATOS ENDS

PILA SEGMENT STACK
   DB 252  DUP(?)
PILA ENDS

CODIGO SEGMENT
    ASSUME CS:CODIGO, DS:DATOS
COMIENZO:  MOV AX, DATOS
           MOV DS, AX
           MOV DX, OFFSET MENSAJE     ;  imprimir MENSAJE
           MOV AH, 9
           INT 21h

           MOV DX, OFFSET CADENANUM   ;  lee cadena numérica
           MOV SI, DX
           MOV BYTE PTR [SI],2        ;  un sólo dígito
           MOV AH,0Ah
           INT 21h
           SUB CH,CH                  ;  hacer CH=0;
           MOV CL, [SI+2]             ;  carácter de dígito a número
           SUB CL,"0"                 ;  CX contiene número

           CALL FACTORIAL          ;  valor de factorial en variable RES
           CALL NUM_A_CADENA       ; convertir RES a cadena en CADENARES

           MOV DX,OFFSET CAMBIO_LINEA;  imprimir cambio de línea
           MOV AH,9
           INT 21h
```

```
            MOV DX, OFFSET CADENARES  ;  imprimir cadena de resultado
            MOV AH,9
            INT 21h

            MOV AH,4Ch                ;  y terminar programa con
            SUB AL,AL                 ;  código de salida AH=0 (normal)
            INT 21h

FACTORIAL   PROC NEAR
            CMP CX,1
            JA SIGUIENTE
            MOV RES,1     ;  si CX es 1 entonces RES vale 1
            JMP FIN
SIGUIENTE:  PUSH CX       ;  si CX es mayor que 1, salvar su valor actual
            DEC  CX       ;  y calcular FACTORIAL de CX-1
            CALL FACTORIAL
            MOV  AX, RES  ;  RES contiene factorial de CX-1
            POP CX        ;  restaurar valor actual de CX
            MUL CX        ;  y hallar en AX nuevo valor de factorial=CX*RES
            MOV RES,AX    ;  actualizar RES
FIN:        RET
FACTORIAL   ENDP

NUM_A_CADENA PROC NEAR
            MOV CX, 0     ;  contador de número de dígitos salvados
                          ;  en la pila
            MOV AX,RES    ;  número a convertir en cadena
OTRO:       MOV DX,0      ;  extensión DX AX
            DIV DIEZ      ;  para división de 16 bits
                          ;  DL=resto de división es último dígito
            ADD DL,"0"    ;  y debe ser convertido a carácter
            PUSH DX       ;  salvarlo en la pila
            INC  CX       ;  aumentar contador
            CMP  AX,0     ;  comprobar si ya se redujo a cero
            JE  LISTO1
            JMP OTRO
LISTO1:     MOV DX, OFFSET CADENARES ;  pasar los dígitos de la pila
            MOV SI,DX     ;  a CADENARES
OTRO1:      CMP CX,0      ;  comprobar si la pila contiene dígitos
            JE LISTO2
            POP  AX       ;  obtener dígito de la pila
            MOV [SI],AL   ;  y pasarlo a DS:SI
            INC  SI       ;  a la siguiente posición en CADENARES
            DEC  CX       ;  disminuir contador
            JMP  OTRO1
LISTO2:     MOV BYTE PTR [SI],"$"     ;  añadir "$" para imprimir cadena
            RET
NUM_A_CADENA ENDP
CODIGO      ENDS
            END COMIENZO
```

3.7 EJERCICIOS

1. Escriba un programa que lea un número N en forma de cadena e imprima la suma de los números 1+2+...+N.

2. Modifique el programa anterior de manera que use una subrutina recursiva para calcular la suma.

3. Se lee una colección de fechas como cadenas de la forma dd/mm/aa.

   Escriba un programa que imprima la menor de tales fechas.

4. Escriba un programa que lea una lista de cadenas de longitud máxima 40, las ordene alfabéticamente y las imprima en forma ordenada.

   Asuma un máximo de 100 cadenas y que la lectura de éstas se hace hasta que se ingrese una cadena formada sólo por caracteres de blanco.

   Utilice una subrutina cercana para efectuar la lectura, otra para la comparación de cadenas y una tercera para copiar cadenas.

5. Extienda el programa anterior de modo que después de ordenadas las cadenas solicite el ingreso de otra cadena XXXX y determine si se halla o nó en la lista usando el método de búsqueda binaria:

   1) Se comienza con BAJO=posición de primera cadena
      ALTO=posición de última cadena
      HALLADO=-1;

   2) mientras BAJO<=ALTO hacer
      a) MEDIO =(BAJO+ALTO)/2
      b) Si XXXX coincide con cadena ubicada en MEDIO se hace HALLADO=MEDIO y BAJO=ALTO+1 para salir de ciclo mientras
      c) de otra manera
         si XXXX < cadena de MEDIO se hace ALTO=MEDIO-1
      d) de otra manera
         si XXXX >cadena de MEDIO se hace BAJO=MEDIO+1;

3) si HALLADO=-1 se imprime mensaje "No se encuentra!" de otra manera se imprime "Se encuentra!".

6. (Torres de Hanoi)

Escriba un programa que imprima los movimientos que deben realizarse para llevar los discos de la torre 1 a la torre 3.

Se indican las siguientes reglas:

1) La torre 1 empieza con N discos ordenados de menor a mayor.
2) Se puede emplear una torre auxiliar 2.
3) Se debe mover un disco cada vez.
4) No se puede poner un disco de tamaño mayor sobre uno de tamaño menor.

Emplee una subrutina recursiva para realizar y mostrar los movimientos.

ARCHIVOS

Un archivo es una colección de datos (bloques de octetos) que con un nombre se escriben o almacenan en algún dispositivo magnético (discos, cintas, etc). Estos datos pueden ser después leídos o recuperados del archivo.

El sistema operativo DOS provee varias formas para trabajar con archivos usando el lenguaje ensamblador. Una de ellas consiste en asignarle a un archivo un número de 16 bits al cual denominaremos **descriptor** del archivo (file handle). Por medio del descriptor se realizan las operaciones básicas de escritura, lectura y movimiento en el archivo.

Las funciones del DOS para manejar archivos son:

Creación	: 3Ch	(60 decimal)
Apertura	: 3Dh	(61 decimal)
Cierre	: 3Eh	(62 decimal)
Lectura	: 3Fh	(63 decimal)
Escritura	: 40h	(64 decimal)
Movimiento del puntero de archivo	: 42h	(66 decimal)

Todas se ejecutan con INT 21h.

Estas instrucciones utilizan el indicador de acarreo CF para determinar si se han ejecutado correctamente (sin acarreo).

Para desactivar o borrar el acarreo se pueden utilizar, por ejemplo:

```
CLC         ;  (clear carry) borra acarreo
SUB AL,AL   ;  hace  AL = 0 y borra acarreo
```

Se pueden emplear las instrucciones de salto corto según el acarreo:

```
JC  destino ;  (jump if carry) saltar si hay acarreo
JNC destino ;  (jump if not carry) saltar si no hay acarreo
```

Nota: El valor del descriptor de un archivo que se crea o abre es asignado (elegido) por el sistema operativo DOS.

Dos también considera como archivos a un conjunto de dispositivos con los cuales se realizan frecuentes transferencia de datos: entrada y salida estándar (por defecto el teclado y la pantalla, resp.), impresora estándar, etc. Estos archivos y sus correspondientes descriptores (0, 1 y 4) han sido previamente definidos por el DOS y por tanto siempre están disponibles.

a) Creación de archivos: 3Ch

Se debe tener:

- en AH el código 3Ch
- en DS:DX la dirección del comienzo del nombre del archivo a crearse (este nombre debe terminar con 0)
- en CX el valor 0 (modo normal de creación)

y se ejecuta con INT 21h.

Resultado:
- Si hay acarreo no se crea el archivo y AX resulta con el código de error (por ejemplo, disco lleno)
- Si no hay acarreo, se crea el archivo y AX resulta con el descriptor asignado.

b) **Apertura de archivos: 3Dh**

Se debe tener:

- en AH el código 3Dh
- en DS:DX la dirección de inicio del nombre del archivo a abrirse (el nombre debe terminar con 0)
- en AL el código de acceso:

  0 = sólo lectura
  1 = sólo escritura
  2 = lectura y escritura

y se ejecuta con INT 21h.

Resultado:

- Si hay acarreo, no se puede abrir el archivo yy AX contiene el código de error (por ejemplo, archivo inexistente)
- Si no hay acarreo, se abre el archivo y AX resulta con el descriptor del archivo.

c) **Cierre de archivos: 3Eh**

Se debe tener:

- en AH el código 3Eh
- en BX el descriptor del archivo a cerrar

y se ejecuta con INT 21h.

Resultado:

- Si hay acarreo no se puede cerrar el archivo y AX contiene el código de error
- Si no hay acarreo se cierra el archivo

d) Lectura de archivos: 3Fh

Se debe tener:

- en AH el código 3Fh
- en BX el descriptor del archivo a leer
- en CX el número de octetos a ser leídos
- en DS:DX la dirección del área de recepción de datos (buffer)

y se ejecuta con INT 21h.

Resultado:

- Si hay acarreo, error de lectura con código de error en AX
- Si no hay acarreo, AX contiene el número de octetos leídos, siendo éste cero si se está en el final del archivo.

e) Escritura de archivos: 40h

Se debe tener:

- en AH el código 40h
- en BX el descriptor del archivo
- en CX el número de octetos a ser escritos
- en DS:DX la dirección del área de datos a copiar (buffer)

y se ejecuta con INT 21h.

Resultado:

Si no hay acarreo la operación de escritura ha sido correcta, y es incorrecta de otra manera.

4.2. PROGRAMA: Crea archivo

En el siguiente programa se crea el archivo "TEXTO" y se escribe en él los 12 primeros octetos del área de datos CADENA.

```
DATOS       SEGMENT
 ARCH_NOMBRE DB "TEXTO",0              ; nombre termina con cero
 CADENA      DB "ARCHIVO CREADO ...!"
 MENS_ERROR  DB "NO SE PUEDE CREAR !$"
DATOS       ENDS

PROGRAMA    SEGMENT
            ASSUME CS : PROGRAMA, DS:DATOS
COMIENZO:   MOV AX,DATOS              ; carga segmento DATOS en DS
            MOV DS,AX

            MOV DX,OFFSET ARCH_NOMBRE ; nombre de archivo en DS:DX
            MOV CX,0                  ; modo normal de creación
            MOV AH,3Ch                ; función para crear archivo
            INT 21h

            JNC ABIERTO               ; si no hay acarreo ir a ABIERTO
            MOV DX,OFFSET MENS_ERROR  ; hay acarreo, imprimir mensaje
            MOV AH,09h                ;                      de error
            INT 21h
            JMP FIN                   ;              y saltar a FIN

ABIERTO:    XCHG AX,BX                ; poner descriptor en BX
            MOV DX,OFFSET CADENA      ; y dirección de CADENA
            MOV CX,9h                 ; escribir 9 octetos
            MOV AH,40h                ;            en archivo
            INT 21h

CIERRE:     MOV AH,3Eh                ; función para cerrar archivo
            INT 21h                   ; asociado a BX

FIN:        MOV AH,4Ch
            INT 21h

PROGRAMA    ENDS
            END COMIENZO
```

4.3. PROGRAMA: Procesa archivo de texto: Cambia minúsculas por mayúsculas

Este programa procesa un archivo "MIN", formado por caracteres, para obtener una copia "MAY", que contiene letras mayúsculas en lugar de las minúsculas.

APLICACION: Si el presente programa fuente lo denominamos PROG8.ASM :

1) obtenga PROG8.EXE
2) produzca una copia de PROG8.ASM con nombre MIN :
   A> COPY PROG8.ASM MIN
3) y ejecute PROG8.

Luego vea el archivo "MAY".

```
programa    segment
            assume cs:programa, ds:programa

comienzo:   mov  ax,programa
            mov  ds,ax                  ; carga segmento de datos
            mov  dx,offset archmin      ; abre archivo MIN
            mov  ah,3Dh
            mov  al,0                   ; modo=sólo lectura
            int  21h
            jc   fin
            mov  dmin,ax                ; ... descriptor en dmin
            mov  dx,offset archmay      ; crea archivo MAY
            mov  ah,3Ch
            mov  cx,0                   ; modo normal=0
            int  21h
            jc   fin
            mov  dmay,ax                ; ... descriptor en dmay
            call procesar               ; llamada a subrutina
fin:        mov  ah,4Ch                 ; fin de proceso
            int  21h

procesar    proc near     ; ********** subrutina procesar **********
            mov  dx,offset area         ; área de datos en DS:DX
            mov  si,dx                  ; registro de índice si con valor
                                        ; igual a dx
```

```
siguiente:  mov  bx,dmin                    ; leer en dmin
            mov  cx,longitud                ;  100 octetos
            mov  ah,3Fh
            int  21h                        ; número leído en ax
            cmp  ax,0                       ; probar fin del archivo
            jne  convertir                  ; no: ir a convertir
            jmp  cerrar                     ; sí: ir a cerrar

convertir:  mov  bx,1                       ; iniciar cuenta
otro:       cmp  bx,ax                      ; comparar límite
            ja   escribir                   ; si es mayor: ir a escribir
            cmp  byte ptr [si+bx-1],"a"     ; no: comparar con "a"
            jb   continuar                  ; si es menor: continuar
            cmp  byte ptr [si+bx-1],"z"     ; comparar con "z"
            ja   continuar                  ; si es mayor: continuar
            sub  byte ptr [si+bx-1],20h     ; no: es minúscula,convertir
continuar:  inc  bx                         ; aumentar la cuenta
            jmp  otro

escribir:   mov  cx,ax                      ; pasar a cx número de
            mov  ah,40h                     ; caracteres e imprimirlos
            mov  bx,dmay
            int  21h
            jmp  siguiente                  ; volver a leer

cerrar:     mov  ah,3Eh                     ; cerrar bx=dmin
            int  21h
            mov  bx,dmay                    ; y también dmay
            int  21h
            ret

procesar    endp          ;*********  fin de procesar  ***********

                          ;*********  área de datos    ***********

archmin     db "MIN",0
dmin        dw ?
archmay     db "MAY",0
dmay        dw ?
longitud    dw 100
area        db 100 dup(?)

programa    ends          ; fin del segmento programa (código y datos)

pila        segment stack
            db 20 dup(?)
pila        ends

            end comienzo
```

Explicación

Indiquemos algunas observaciones sobre la subrutina procesar:

1) Se emplea el arreglo **area** con 100 octetos para realizar las operaciones de transferencia de datos entre los archivos.

2) La dirección de este arreglo se almacena en dos registros: DX, para leer y escribir, y SI, a efecto de poder acceder a cada octeto del arreglo.

3) El rango de direcciones posibles para este arreglo de 100 octetos es:

SI+0 ,..., SI+99

4) Como se puede apreciar las funciones básicas de la subrutina son:

- Leer el archivo "MIN" poniendo los datos en el arreglo
- Tratar los caracteres válidos del arreglo, cuyo número se guarda en AX, a fin de convertir las letras minúsculas en mayúsculas
- Escribir estos datos en el archivo "MAY"

5) Haciendo que bx recorra los valores desde 1 hasta ax, se accede a cada octeto del arreglo mediante direccionamiento indirecto utilizando como subíndice el valor si + bx - 1:

byte ptr [si+bx-1]

6) Se compara este octeto con "a" y "z" a fin de determinar si se trata de un carácter de letra minúscula. En caso afirmativo se convierte en mayúscula sustrayéndole 20h (=32 decimal), que es la diferencia que existe entre los códigos ASCII de una letra mayúscula y su correspondiente minúscula:

```
A = 41h  ,      B = 42h , ... , Z = 5Ah
a = 61h  ,      b = 62h , ... , z = 7Ah
```

4.4. MOVIMIENTO DEL PUNTERO (de lectura/escritura) DE UN ARCHIVO : función del DOS 42h

Dentro de un archivo es posible desplazarse a una dirección específica y luego ejecutar operaciones de lectura o de escritura a partir de la dirección siguiente.

Utilizando esta función, el archivo puede ser tratado como uno de acceso aleatorio o directo. Por ejemplo, si se trabaja con un archivo formado por registros (o bloques) de 150 octetos de longitud, para acceder al cuarto registro será necesario localizar el comienzo del cuarto bloque de 150 octetos, este es dado por el número 3*150, pues los octetos del archivo se numeran empezando por 0.

Nota

1) Las clases de operaciones a realizar en un archivo dependen del modo de acceso que se haya escogido en el momento de crearlo o de abrirlo.

2) Igual que en el direccionamiento de memoria, las direcciones en un archivo emplean un par de palabras de 16 bits I: J. En particular se reitera que estas direcciones se numeran a partir de 0.

Para mover el apuntador de un archivo se requiere tener:

- en AH el código 42h
- en BX el descriptor del archivo
- en el par CX:DX el desplazamiento del apuntador, esto es, el número de direcciones que ha de moverse con respecto a un punto de referencia
- en AL el código del punto de referencia:

  0 = para el comienzo del archivo
  1 = para la posición actual
  2 = para el final del archivo

y se ejecuta con INT 21h.

Resultado:

- Si hay acarreo, operación incorrecta.
- Si no hay acarreo, el par DX:AX contiene la dirección actual del apuntador, que debe ser igual a:

  = dirección del punto de referencia + valor dado en CX:DX

4.5. PROGRAMA: Archivo de acceso aleatorio

El siguiente programa procesa en forma aleatoria un archivo "LISTA" compuesto por 100 registros de 30 octetos.

Las funciones básicas del programa son:

1) Abrir el archivo en el modo de lectura/escritura, si existe, o crearlo iniciando todos sus registros con caracteres de blanco.

2) Solicitar el ingreso de un número de registro en respuesta al mensaje:

   Registro ? : _

   hasta que se ingrese un número inválido, es decir fuera del rango 1,...,100, en cuyo caso el programa finaliza.

   Si el registro existe, se muestra su contenido y se espera el ingreso de la nueva información:

   Ingrese información : _

   pudiendo pulsarse solamente <enter> para no modificar el registro.

Esta vez el programa ejecutable "PROG9.COM" será de tipo COM, por lo cual consistirá de un sólo segmento de programa, con la instrucción:

```
ORG 100h  ; dirección (origen) de entrada para esta
          ; clase de programas
```

al comienzo del mismo.

Los pasos a seguir para obtener PROG9.COM son:

1) Escribir el programa fuente PROG9.ASM
2) Procesar PROG9.ASM con [M]ASM yyoobtener PROG9.OBJ
3) Procesar PROG9.OBJ con LINK.EXE y obtener PROG9.EXE
4) Procesar PROG9.EXE con EXE2BIN.EXE:

EXE2BIN PROG9.EXE PROG9.COM

```
programa    segment
            assume cs: programa, ds:programa

            org 100h
comienzo:   mov  ax,cs              ; ********************************
            mov  ds,ax              ; *                              *
            call borrar             ; *                              *
            call abrir              ; *                              *
            jnc  abierto            ; *         PROGRAMA             *
            call crear              ; *                              *
            jc   fin                ; *         PRINCIPAL            *
abierto:    call procesar           ; *                              *
            call cerrar             ; *                              *
fin:        mov  ah,4Ch             ; *                              *
            int  21h                ; ********************************

borrar      proc near               ; subrutina para borrar pantalla
            mov  ah,6h              ; mover arriba (scroll up)
            mov  al,0               ; toda la pantalla
            mov  cx,0               ; desde fila=ch=0, columna=cl=0
            mov  dx,184fh           ; hasta fila=dh=24,columna=dl=79
            mov  bh,07h             ; video normal=fondo negro,
                                    ; y relieve=blanco
            int  10h                ; ejecución: rutina de video

            mov  ah,02h             ; y ubicar cursor en
            mov  dx,0               ; fila=dh=0, columna=dl=0
            mov  bh,0               ; de la página actual
            int  10h                ;
            ret
borrar      endp

abrir       proc near
            mov  dx, offset nombre  ; subrutina para abrir archivo
            mov  ah, 3Dh            ; (si existe) en el modo al=2,
            mov  al, 2h             ; para leer o escribir
            int  21h                ; variable arch contiene descrip-
            jc   fin_abrir          ;   tor ...
            mov  arch,ax
fin_abrir:  ret
abrir       endp
```

```
crear       proc near
            mov  dx, offset nombre  ; SUBRUTINA PARA CREAR ARCHIVO
            mov  ah, 3Ch            ; e iniciar con  blancos 100
            mov  cx, 0              ; registros con 30 octetos c/u
            int  21h
            jc   fin_crear
            mov  arch, ax
            mov  cx, longitud_reg
            mov  dx, offset cadena  ; (cadena ha sido iniciada
            add  dx,2h              ; con blancos, ver área de datos)
            mov  bx, arch           ; escribir 100 veces (secuencial-
            mov  codigo, minimo     ; mente) contenido de cadena en

siguiente:  cmp  codigo, maximo     ; archivo ...
            ja   fin_crear
            mov  ah, 40h
            int  21h
            inc  codigo
            jmp  siguiente
fin_crear:  ret
crear       endp

cerrar      proc near
            mov  bx, arch           ; SUBRUTINA PARA CERRAR ARCHIVO
            mov  ah, 3Eh
            int  21h
            ret
cerrar      endp


procesar    proc near
            call leer_cod           ; SUBRUTINA DE PROCESAMIENTO
            cmp  codigo, invalido   ; DE REGISTROS. Invoca a
            jne  continuar          ; varias subrutinas: leer_cod,
            jmp  fin_procesar       ; mostrar_reg y actualizar.
continuar:  call mostrar_reg        ; PROCESAR se ejecuta mientras
            call actualizar         ; código sea válido
            jmp  procesar

fin_procesar: ret
procesar    endp
```

```
leer_cod        proc near
                mov  dx, offset mensaje1 ; SUBRUTINA DE LECTURA DE
                mov  ah, 9               ; CADENA DE CODIGO y con-
                int  21h                 ; versión a binario
                mov  codigo, 0
                mov  dx, offset cadena_cod
                mov  si, dx
                mov  ah, 10
                int  21h
                mov  dx, offset clinea
                mov  ah, 9
                int  21h
                mov  ch, 0
                mov  cl, byte ptr [si+1]
                mov  bx, 0
sig_car:        cmp  bx, cx
                jae  probar
                xchg ax, codigo
                push cx
                mov  cx, 10
                mul  cl
                pop  cx
                xchg ax, codigo
                mov  al, byte ptr [si+bx+2]
                sub  al, "0"
                sub  ah, ah
                add  codigo,ax
                inc  bx
                jmp  sig_car
probar:         cmp  codigo, minimo
                jb   valor_invalido
                cmp  codigo, maximo
                jbe  fin_leer_cod
valor_invalido::mov  codigo, invalido
fin_leer_cod:   ret

leer_cod        endp

mostrar_reg     proc near
                call mover_apunt         ; SUBRUTINA PARA MOSTRAR
                mov  dx, offset cadena   ; REGISTRO de código leído.
                add  dx, 2               ; Llama a la subrutina
                mov  cx, longitud_reg    ; mover_apunt para ubicar
                mov  ah, 3Fh             ; el apuntador del archivo
                int  21h                 ; al comienzo del registro
                mov  dx, offset cadena   ; del código correspon-
                add  dx,2                ; diente.
                mov  ah, 9
                int  21h
                ret
mostrar_reg     endp
```

```
actualizar      proc near

                mov  dx, offset mensaje2 ; SUBRUTINA PARA ACTUALIZAR
                mov  ah, 9               ; REGISTRO
                int  21h
                mov  dx, offset cadena
                mov  si, dx
                mov  ah, 10
                int  21h
                mov  dx, offset clinea
                mov  ah, 9
                int  21h
                mov  bl, byte ptr [si+1]
                mov  bh,0                ; si no hay ingreso de
                cmp  bx,0                ; datos ir a fin_actual
                je   fin_actual
blanco:         cmp  bx, longitud_reg    ; justifica cadena por la
                ja   escribir            ; derecha: llena de blancos
                mov  byte ptr [si+bx+2]," "; caracteres restantes ...
                inc  bx
                jmp  blanco
escribir:       call mover_apunt         ; ubica apuntador
                mov  dx, offset cadena   ; para escribir nueva
                add  dx, 2               ; información
                mov  cx, longitud_reg
                mov  ah, 40h
                int  21h
fin_actual:     ret

actualizar      endp

mover_apunt     proc near
                mov  bx, arch            ; SUBRUTINA PARA UBICAR
                mov  ax, codigo          ; REGISTRO correspondiente
                dec  ax                  ; al código ...
                mov  cx, longitud_reg
                mul  cl
                mov  cx, 0
                xchg ax, dx
                mov  ah, 42h
                mov  al, 0
                int  21h
                ret
mover_apunt     endp

minimo          equ  1                   ; DEFINICION DE CONSTANTES
maximo          equ  100                 ; DE PALABRAS (16 bits)
longitud_reg    equ  30
invalido        equ  0                   ; indicador de código
```

```
                        ;   DEFINICION DE VARIABLES

mensaje1        db   "Registro ? : $"
mensaje2        db   "Ingrese información : $"
nombre          db   "LISTA",0
arch            dw   ?                      ; = descriptor de archivo
cadena          db   31,?, 31 dup (" ")     ; área de transferencia
clinea          db   10,13,"$"              ; iniciada con blancos
cadena_cod      db   4,?, 4 dup (?)         ; =cadena para lectura del
codigo          dw   ?                      ; código en caracteres
                                            ; ASCII
programa        ends
                end  comienzo
```

4.6. EJERCICIOS

1. Escriba un programa que lea una cadena e imprima sus caracteres en orden inverso.

2. Escriba un programa que lea dos números enteros no negativos (máximo 5 dígitos) e imprima su suma.

   Modifique el programa de modo que imprima la diferencia de dos números no negativos (con signo - si es negativa)

3. Escriba un programa que lea 10 nombres y los salve en el archivo LISTA, cada uno separado por cambios de línea (código 13 y 10). Utilice una subrutina para leer un nombre y salvarlo en el archivo.

4. Escriba un programa que imprima en la impresora el contenido de un archivo con la función 05h de INT 21h (AH=05h, DL=carácter a imprimir).

5. Modifique el programa anterior para hacer la impresión tratando a la impresora como un archivo con descriptor 4.

6. Escriba un programa que imprima el número de octetos que componen a un archivo cuyo nombre es leído durante corrida.

   Para simplificar suponga que el tamaño del archivo no excede 64K.

   Indicación: Utilice la función de movimiento del apuntador con los siguientes parámetros:

   ```
   CX:DX = 0:0      (desplazamiento cero)
      AL = 2        (respecto del final del archivo)
   ```

   que después de ser ejecutada pondrá el tamaño del archivo en el par DX:AX.

7. Escriba un programa que una los datos de un archivo al final de otro archivo. Se debe solicitar los nombres de los archivos.

   Sugerencia: Abra el primer archivo en el modo de acceso aleatorio ubicando el puntero al final del mismo. Luego proceda a copiar los datos del segundo archivo a partir de este lugar.

8. Escriba un programa que cuente el número de líneas que forman un archivo de texto. Suponga que cada línea termina con el carácter de código 10.

9. Escriba un programa que cuente el número de palabras de un archivo. Asuma que cada palabra se separa por caracteres de blancos o cambios de línea (código 10).

10. Escriba un programa que lea un número hexadecimal de dos dígitos, lo convierta a binario (entre 0 y 255) y lo imprima en decimal.

11. Escriba un programa para borrar un archivo cuyo nombre se pide en corrida.

Sugerencia: Utilice la función para borrar archivos ingresando:

```
AH=41h
DS:DX = dirección de nombre de archivo (termina con 0).
INT 21h (ejecución)
```

12. Escriba un programa para cambiar el nombre de un archivo. El programa pide el ingreso del nombre actual del archivo y el nuevo nombre.

Sugerencia:

```
Ingrese AH=56h   ; función para cambiar nombre de archivo
        DS:DX    = dirección de nombre actual
        ES:DI    = dirección de nuevo nombre
        INT 21h  (ejecución)
```

Nota: Suponiendo que la dirección del nuevo nombre se halla en la variable NUEVO del segmento DS, para cargar su dirección en el par ES: SI, proceda así:

```
MOV AX, OFFSET NUEVO
MOV DI,AX
MOV AX, DS
MOV ES,AX
```

o bien con la instrucción equivalente:

```
LES DI, NUEVO
```

OTRAS INSTRUCCIONES DE LENGUAJE ENSAMBLADOR

5.1. PROCESAMIENTO DE CADENAS: MOVSB, MOVSW

Una cadena (string) es un grupo de octetos o palabras que se almacenan en direcciones consecutivas de la memoria.

Consideramos dos operaciones con las cadenas:

- copiar una cadena a otro lugar de la memoria
- comparar dos cadenas

Es muy simple el escribir las instrucciones que realizan tales operaciones. En efecto, basta hacer que las direcciones de las dos cadenas se hallen en DS:SI y DS:DI, respectivamente, de modo que las expresiones [SI] y [DI] representen los contenidos de los componentes de las cadenas; entonces los registros SI y DI pueden ser incrementados para acceder a los siguientes datos.

Por ejemplo, para copiar los N primeros octetos de una cadena fuente a otra de destino (suponiendo que ambos se encuentran en el segmento DS):

```
            MOV SI., desplazamiento de fuente en SI

            MOV DI , desplazamiento de destino en DI

            MOV CX, N       ;  número de caracteres restantes

siguiente:  CMP CX, 0       ;  probar si quedan
            JE  completo
            MOV AL, [SI]    ;  copiar [SI] a [DI]
            MOV  DI , AL
            INC SI          ;  incrementar índice SI
            INC DI          ;  incrementar índice DI
            DEC CX          ;  disminuir contador
            JMP siguiente
completo:
```

Sin embargo, el procesador 8086 cuenta con instrucciones que permiten llevar a cabo operaciones similares, por ejemplo las instrucciones:

MOVSB (move string by bytes = mover cadena por octetos)

MOVSW (move string by words = mover cadena por palabras)

Para utilizarlas se requiere tener:

1) en CX el número de datos a copiar
2) en DS:SI la dirección de la cadena fuente
3) en ES:DI la dirección de la cadena de destino (por ejemplo, si destino está en DS hay que hacer ES=DS:

```
            MOV AX,DS
            MOV ES,AX )
```

4) el indicador de dirección DF en el modo deseado:

- creciente, para que los índices SI y DI se incrementen (automáticamente) cada vez
- o decreciente, para que decrementen

Si es necesario, este indicador puede ser establecido con dicho propósito mediante las instrucciones:

```
    CLD    ;  creciente (UP) = clear direction flag
ó   STD    ;  decreciente (DN) = set direction flag
```

La transferencia de datos se ejecuta empleando una de las instrucciones:

```
    REP  MOVSB    ;  movimiento por octetos
    REP  MOVSW    ;  movimiento por palabras (dos octetos)
```

con el prefijo REP para repetir el proceso mientras CX sea distinto de cero: Cada vez que se copia un dato, CX decrece en una unidad y también los registros de índices crecen o decrecen, según el indicador de dirección, en una o dos u unidades, si es por octetos o palabras, respectivamente.

5.2. PROGRAMA: Mueve cadena

El siguiente programa lee una cadena en la variable FUENTE, copia los primeros 15 caracteres de ésta en la variable DESTINO y los imprime.

```
programa    segment
            assume cs: programa, ds: programa

comienzo:
            mov ax, cs
            mov ds, ax
            mov dx, offset FUENTE    ; lectura de FUENTE
            mov ah, 10
            int 21h
            add dx, 2                ; dirección de primer carácter
            mov si, dx               ; de FUENTE en DS:SI
```

```
                mov dx, offset CLINEA    ; imprime cambio de línea
                mov ah, 9
                int 21h
                cld
                mov di, offset DESTINO

                mov ax, ds
                mov es, ax               ; cadena DESTINO en ES:DI = DS:DI
                mov cx, 15               ; número de octetos : 15
                rep movsb                ; copiar

                mov byte ptr [di],"$"    ; carácter $ al final
                mov dx, offset DESTINO   ; de DESTINO
                mov ah, 9                ; para imprimir
                int 21h

FUENTE          db  25,?, 25 dup (?)
DESTINO         db  40 dup (?)
CLINEA          db  10,13,"$"
programa        ends
                end     comienzo
```

5.3. COMPARACION DE CADENAS: CMPSB, CMPSW

También existen instrucciones para comparar cadenas:

```
REP    CMPSB     ; comparación por octetos
REP    CMPSW     ; comparación por palabras (16 bits)
```

que modifican los apuntadores SI y DI, según el indicador de dirección, mientras CX sea distinto de cero y sean iguales los datos de las cadenas fuente y destino.

Para usar estas instrucciones se requiere la misma disposición de las cadenas que en el caso del movimiento, pero ahora CX debe contener el número de datos a ser comparados.

Debe señalarse que el proceso de comparación equivale a sustraer los datos de la cadena de destino de la cadena fuente y que el resultado sólo afecta los indicadores de estado. Así, las dos cadenas son iguales si los primeros CX caracteres son los mismos en las respectivas posiciones; de lo contrario el orden de las cadenas es determinado por el primer par de caracteres respectivos que son distintos.

Se pueden emplear las instrucciones de salto según comparación de cadenas:

```
JE    ; saltar si son iguales
JB    ; saltar si fuente es menor (sin signo)
JA    ; saltar si fuente es mayor (sin signo)
JAE   ; saltar si fuente es mayor o igual que
        destino
etc
```

Reiteramos que la cadena fuente debe ser puesta en la dirección DS:SI y la cadena de destino en ES:DI.

5.4. PROGRAMA: Compara cadena de octetos

Este programa ilustra el uso de la instrucción de comparación de cadenas REP CMPSB.

Se consideran las cadenas:

```
        fuente  ="CADENA FUENTE"
y       destino ="CADENA DESTINO"
```

y se comparan solamente los 8 primeros octetos de ambas, resultando por tanto la primera mayor (o mayor o igual) que la segunda: Pues el primer carácter de fuente ("F" con código 46h) distinto del correspondiente en destino ("D" con código 44h) es mayor (o mayor o igual).

```
programa    segment
            assume cs: programa, ds: programa

comienzo:   mov ax, cs
            mov ds, ax                  ; carga dirección en ds
            mov es, ax                  ; carga dirección en es (=ds=cs)
            cld
            mov si, offset fuente
                                        ; cadena fuente en ds:si
            mov di, offset destino
                                        ; cadena destino en es:di
            mov cx, limite              ; número de octetos a
            rep cmpsb                   ; comparar

            ja  mayor                   ; seleccionar mensaje
            jb  menor
            mov dx, offset mens_igual
            jmp fin
mayor:      mov dx, offset mens_mayor
            jmp fin
menor:      mov dx, offset mens_menor

fin:        mov ah, 9                   ; imprimir mensaje
            int 21h
            mov ah, 4Ch                 ; fin de proceso
            int 21h

fuente      db "CADENA FUENTE"
destino     db "CADENA DESTINO"
limite      dw 8
mens_mayor  db "fuente es mayor $"
mens_igual  db "son iguales $"
mens_menor  db "fuente es menor $"

programa    ends

            end comienzo
```

5.5. INSTRUCCIONES LOGICAS

Indicamos ahora las instrucciones para realizar operaciones con los datos teniendo en cuenta sus dígitos binarios. Estas operaciones, llamadas **lógicas**, se aplican aquí tanto a octetos como a palabras y son :

Operación	Efecto
NOT operando	negación: cambia en operando ceros por uno y éstos por ceros
AND operando1, operando2	"y" lógico: para cada par de bits correspondientes da 1, si son iguales, y 0, de otra manera
OR operando1, operando2	"o" lógico: para cada par de bits correspondientes da 1, si al menos uno de ellos es 1, y 0 si ambos son 0
XOR operando1, operando2	"o exclusivo" : para cada par de bits correspondientes da 1, si son distintos, y 0, en caso contrario.

Nota

1. Todas ponen el resultado de la operación en el primer operando
2. La operación NOT no afecta a los indicadores de estados
3. Las restantes operaciones anulan los indicadores de acarreo y de desbordamiento: CF y OF. Según el resultado de la operación son afectados: CF, OF, PF, SF y ZF.

   Por ejemplo: Se anula el indicador de cero ZF si el resultado de la operación es cero, y se establece en caso contrario.

Ejemplos

1. Si AL = 15h = 0001 1C01 (binario) entonces: NOT AL da como resultado AL = 1110 0110 (binario) = 0CAh

2. Si DX = 320Ah = 0011 0010 0000 1010 (binario)
   y ALFA = 07F4h = 0000 0111 1111 0100 (binario)

   entonces AND ALFA, DX da:
   ALFA = 0000 0010 0000 0000 (binario)
   = 0 2 0 0 h

   OR DX, ALFA da:
   DX = 0011 0111 1111 1110 (binario)
   = 3 7 F E h

   y XOR ALFA, DX da:
   = 0011 0101 1111 1110 (binario)
   = 3 5 F E h

   Para anular el bit más significativo de un octeto - se suele decir "enmascarlo" - se puede emplear:

   AND octeto,0111111 (binario)
   ó AND octeto,7Fh

5.6. INSTRUCCION TEST

Tiene la forma:

TEST operando1, operando2

en donde ambos operandos deben ser del mismo tipo: octetos o palabras.

Esta instrucción produce el mismo efecto sobre los indicadores de estado que la instrucción AND operando1, operando2, pero no altera el contenido de operando1.

Así, puede ser empleada para determinar si son nulos un grupo de bits de operando1 para lo cual basta hacer igual a uno todos los bits correspondientes de operando2 y cero los restantes. Entonces se tendrá una respuesta afirmativa en caso de anularse el indicador de cero - resultado de la operación AND es cero -. El tratamiento de los bits no nulos de operando1 puede reducirse a la situación anterior.

Ejemplo

Para probar si los bit 6 y 3 de AL - se cuenta desde 0 a partir de la derecha - son ceros se puede utilizar:

```
TEST AL, 0100 1000 binario        (ó TEST AL, 48h)
JZ   SON_CEROS
```

y si se desea determinar si tales bits son unos:

```
NOT  AL           ; intercambia unos por ceros
TEST AL, 48h      ; prueba ceros
NOT  AL           ; regreso al valor original
JZ   SON_UNOS
```

5.7. ROTACION Y DESPLAZAMIENTO DE BITS

Los bits de un octeto o palabra pueden ser rotados o desplazados a la izquierda o a la derecha.

En una rotación los bits son movidos en el operando según el sentido indicado de modo que cada vez que sale un valor éste se ubica en el extremo que queda libre.

En todos los casos el indicador de acarreo CF resulta con el valor del último bit que ha salido del operando, que en el operando resultante es el bit del extremo opuesto a la dirección del movimiento.

Para rotar bits se utilizan las instrucciones:

```
ROL destino, n    ; rotación de n bits a la izquierda
ROR destino, n    ; rotación de n bits a la  derecha
```

El resultado se pone en destino y CF recibe el último bit que ha salido. n puede ser el valor constante 1 ó dado por el registro CL.

Ejemplos

1. Si AH = 01011010 binario (=59h) entonces: ROL AH,1

   rota operando 1 bit a la izquierda: todos los bits de operando se mueven un lugar hacia la izquierda y el bit salido se pone en el extremo derecho, y por lo tanto se tiene

   AH = 10110100 binario (=0A4h) y CF=0 (=bit salido de operando)

2. Si CL=4 y BL= 1111000 binario, entonces: ROR BL,CL

   da BL= 00001111 y CF=1 (=último bit salido)

   que equivale a intercambiar los cuartetos (nibbles) de BL.

Un desplazamiento (lógico) de bits consiste en mover los bits en el sentido indicado de modo que cada bit que queda libre es reemplazado con cero.

Igual que con las rotaciones, el indicador de acarreo CF resulta con el valor del último bit salido del operando.

Las instrucciones para desplazar bits son:

```
SHL operando,n  ; desplazamiento de n bits a la izquierda
                  (shift left)
SHR operando,n  ; desplazamiento de n bits a la derecha
                  (shift right)
```

n puede ser el valor constante 1 ó un valor contenido en CL.

Ejemplo

Si (la palabra) ALFA = 01010110 11110001 binario (=56F1h)
y CL =3, entonces: SHL ALFA, CL

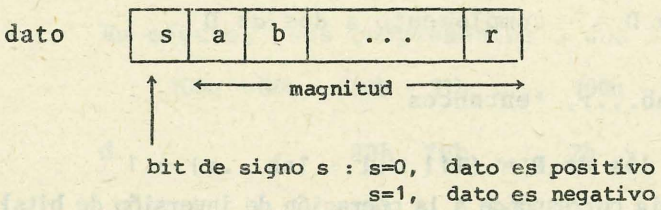
desplaza 3 bits de ALFA a la izquierda: sale 010, resultando

ALFA = 10110111 10001000 binario (=0B788h)
y CF =0 (=último bit salido)

5.8. ARITMETICA DE ENTEROS CON SIGNO: Números positivos y negativos

Para tratar los números con signo (positivos y negativos) se conviene en considerar el bit de la izquierda de un octeto (o de una palabra) como bit de signo: Se asume que representa un número positivo o negativo según este sea igual a cero o a uno.

Así en general para datos interpretados con signo se tiene:



en donde magnitud designa al número formado con los restantes bits (s excluye el del signo).

Para obtener el valor de un dato con signo se considera negativa, por definición, la unidad de orden correspondiente al bit del signo, de modo que: valor = - s00...0 + 0ab...r

De este modo, los valores de los datos positivos coinciden con los de sus respectivas representaciones.

El valor de un dato negativo se suele calcular usando su com complemento a dos. Para obtener el complemento a dos de un dato binario se invierten sus bits, esto es, se cambia cada bit 1 por 0 y cada bit 0 por 1, y al resultado se suma 1. Se ignora el acarreo.

Ejemplos

Determinamos los complementos a dos de los octetos 0110 1101 y 1111 1111.

```
(1)  invirtiendo los bits:  1001 0010
     sumando 1          :       +    1
     complemento a dos  =  1001 0011

(2)  invirtiendo los bits:  0000 0000
     sumando 1          :       +    1
     complemento a dos  =  0000 0001
```

Mediante el uso del complemento a dos del cálculo del valor de un dato negativo D es dado por:

valor de D = - complemento a dos de D

En efecto, si D=1ab...r, entonces

```
complemento a dos de D = (111...1 - 1ab...r) + 1
    [la diferencia corresponde a la operación de inversión de bits]
  = (011...1 - 0ab...r) + 1  [restanto los dígitos de la izquierda]
  = (100...0 - 0ab...r)      [sumando 1 al primer número]
  = - (- 100...0 + 0ab...r)
  = - valor                  [definición de valor con signo]
```

Regla Práctica

El complemento de un dato (binario o hexadecimal) se puede calcular directamente restándolo del número formado por 1 seguido por tantos ceros como dígitos tenga el dato.

Ejemplos

1. Calculamos el valor de la palabra negativa FF00h. Usando la regla para hallar el complemento a dos:

   complemento a dos = 10000h - FF00 = 0100 (hexadecimal)

   y por tanto, el valor de FF00h es - 0100h.

2. Los valores de los octetos negativos

   80h, 81h, ..., FEh y FFh,
   son -80h, -7Fh, ..., -2h y -1h, respectivamente.

   En efecto, sus complementos a dos son:

   100h - 80h, 100h - 81h, ..., 100h - FEh y 100h - FFh,

   ó 80h, 7Fh, ..., 2h y 1h.

5.9. SUMA Y RESTA DE ENTEROS CON SIGNOS

La representación de los datos con signo usando el bit de la izquierda como bit de signo tiene la ventaja de utilizar las mismas operaciones de los datos sin signo; por ejemplo las operaciones de sumar y restar son idénticas para ambas clases de datos. No obstante, para interpretar correctamente los resultados en el caso de los datos con signo es preciso utilizar los indicadores de estado de signo y de desbordamiento: SF y OF. En este caso se ignora el acarreo.

Los resultados de la suma y resta de dos datos con signo deben ser considerados con sus correspondientes signos. A veces estas operaciones originan un desborde o sobreflujo si el resultado escapa del rango de los datos. Hay desborde cuando:

1) Suma de dos positivos (negativos) es negativa (positiva)
2) La diferencia de dos datos de distintos signos tiene signo distinto del minuendo.

Ejemplos

Las siguientes sumas de octetos con signo son correctas (sin desborde) :

ambos positivos : 24h + 3Ah = 5Eh, resultado positivo
ambos negativos : D5h + C3h = 98h, resultado negativo
signos distintos: 15h + 97h = ACh, resultado negativo

En cambio las siguientes sumas dan lugar a desborde:

ambos negativos: 24h + 6Ah = 8Eh, resultado negativo
ambos negativos: F0h + 8Bh = 0Bh, resultado positivo

5.10. SALTOS CONDICIONALES SEGUN RESULTADOS CON SIGNOS: JL, JG, JLE, JGE

Para los datos con signo se dispone de un conjunto de instrucciones de saltos cortos condicionales, similares a las de los datos sin signo, que frecuentemente son precedidas con instrucciones de comparación: CMP valor1,valor2. Estas instrucciones son:

```
saltar si valor1 es menor que valor2 : JL (jump if less)
saltar si valor1 es mayor que valor2 : JG (jump if greater)
saltar si menor o igual              : JLE
saltar si mayor o igual              : JGE
```

Es importante reiterar que estas instrucciones tienen en cuenta el signo de los datos a diferencia de las anteriores JB, JA, JAE y JBE que se aplican a los datos sin tener en cuenta sus signos.

Por ejemplo, si valor1=45h y valor2=80h, y se comparan, entonces la instrucción

```
JB destino      ; jump if below
```

producirá el salto a la dirección indicada por destino,pues 45h es menor que 80h si se consideran sin signos.

Por el contrario,

```
JL destino      ; jump if less
```

no efectuará el salto ya que 45h (positivo) no es menor que 80h (negativo).

Existen instrucciones para multiplicar y dividir datos en las que el resultado toma en cuenta el signo de los operandos: IMUL y IDIV, y se usan en la misma forma que las correspondientes instrucciones sin signo.

5.11. PROGRAMA: Salva pantalla actual en archivo.

El siguiente programa salva la pantalla actual en el archivo PANTALLA.MEM. Los datos de la pantalla se almacenan en la memoria en bloques de 4000 o 16000 octetos, según se use el adaptador monocromático o el de color. En el primer caso la dirección del bloque es B000:0000 y se compone de 25 filas x 80 columnas con 2 octetos por posición (en total 4000 octetos). En el segundo caso el bloque empieza en B800:0000 y se le puede dividir en 4 páginas de 25 filas x 80 columnas.

El programa es:

```
; Estos valores se usan para pantallas monocromáticas.
; Para pantallas de color o de gráficos emplear los correspondientes:
; 0b800h :0  y  16000
seg_pantalla  equ 0b000h
off_pantalla  equ 0
tam           equ 4000

programa      segment
              assume cs:programa
comienzo:     jmp  siguiente
archivo       db   "PANTALLA.MEM",0
siguiente:    mov  ax, cs
              mov  ds, ax          ; ds:dx =dirección de nombre de archivo
              mov  dx, offset archivo
              mov  ah, 3ch         ; crear archivo
              mov  cx, 0           ; modo normal
              int  21h
              jc   fin             ; si hay error de creación ir a fin
              xchg bx, ax          ; descriptor en bx
              mov  ax, seg_pantalla; DS:DX= dirección de pantalla
              mov  ds, ax
              mov  dx, off_pantalla
              mov  cx, tam         ; número de datos
              mov  ah, 40h         ; escribir en archivo bx
              int  21h
              mov  ah, 3eh         ; cerrar archivo, descriptor en bx
              int  21h
fin:          mov  ah, 4ch         ; fin de proceso
              int  21h

programa      ends
              end comienzo
```

5.12. PROGRAMA: Restablece pantalla salvada en archivo

El siguiente programa muestra la pantalla salvada anteriormente en el archivo "PANTALLA.MEM"

```
seg_pantalla equ 0b000h          ; monocromático
off_pantalla equ 0
tam          equ 4000

programa     segment
             assume cs: programa
comienzo:    jmp siguiente
archivo      db "PANTALLA.MEM",0
siguiente:   mov  ax, cs
             mov  ds, ax         ; ds:dx= dirección de nombre de archivo
             mov  dx, offset archivo
             mov  ah, 3dh        ; abrirlo en modo de lectura
             mov  al, 0
             int  21h
             jc   fin            ; si hay error ir a fin
             xchg bx, ax         ; descriptor en bx y se mantiene allí
             mov  ax, seg_pantalla ; ds:dx= dirección de pantalla
             mov  ds, ax         ; para copiar desde archivo
             mov  dx, 0
             mov  cx, tam        ; número de datos
             mov  ah, 3fh        ; leer de archivo bx
             int  21h
             mov  ah, 3eh        ; cerrar archivo bx
             int  21h
fin          mov  ah, 4ch        ; fin de proceso

programa     ends
             end comienzo
```

5.13. PROGRAMA: Determina si pantalla es monocromática o de color./gráfico.

Este programa determina la clase de adaptador de pantalla: monocromático o de color/gráfico.

Se usa la interrupción de manejo video int 10h con el valor 0Fh en el registro ah para obtener el modo de despliegue por pantalla: El valor del registro al resulta igual a 7 si el adaptador es monocromático o un valor distinto si éste es de color o de gráfico.

```
programa    segment
            assume cs:programa
comienzo:
            mov  ax, cx
            mov  ds, ax          ; hacer ds=cs para acceder mensajes

            mov  ah, 0Fh         ; determinación de la clase de pantalla
            int  10h             ;
                                 ;
            cmp  al, 7           ;
            je   es_mono
            mov  dx, offset mens_color ;  es color
            jmp  short imprimir
es_mono:    mov  dx, offset mens_mono  ;
imprimir:   mov  ah, 9                 ; función para imprimir cadena
            int  21h                   ; ubicada en DS:DX

            mov  ah, 4ch               ; fin de proceso
            int  21h

;  ***************  área de datos

mens_mono   db 13,10,"Pantalla es monocromática",13,10,"$"
mens_color  db 13,10,"Pantalla es de color o de gráfico",13,10,"$"

programa    ends
            end comienzo
```

5.14. EJERCICIOS

1: Escriba los programas de salvado y restauración de pantalla anteriores de modo que tome en cuenta la clase de pantalla.

2. Escriba un programa que escriba directamente en la pantalla el carácter "A" en la posición fila=4 y columna =10.

Sugerencia: Establezca en el par ES:SI la dirección de la posición indicada:

comienzo de pantalla + 660,

pues 660 = (4*80 + 10) * 2.

Emplee un registro general para escribir el carácter, por ejemplo DX:

```
DH=7      ;  atributo normal para el carácter
DL="A"
```

y mueva DX a ES:SI:

```
MOV ES:[SI],DX
```

3. Escriba un programa que lea un archivo cuyo nombre se ingresa durante corrida y escriba otro con los datos del primero pero en el cual cada octeto tenga su bit más significativo igual a cero.

4. Escriba un programa usando las operaciones lógicas para imprimir cada cuarteto (nibble) de la variable

```
NUMERO DW 3A21h
```

5. Escriba un programa que lea dos cadenas de números (máximo cinco dígitos) e imprima su producto.

M A C R O S

6.1. DIRECTIVA DE FORMACION DE MACROS

La directiva MACRO ... ENDM tiene la forma:

```
N       MACRO lista de parámetros formales
        instrucciones
        ENDM
```

en donde: N es el nombre o identificador de la macro
la lista de parámetros formales tiene la forma
PI,P2,..,,PK (la lista puede ser vacía)

Con esta directiva el programa MASM reemplaza cada ocurrencia de N por el conjunto de instrucciones definidas en el cuerpo de la macro. Si se incluyen parámetros, la macro N se utiliza así:

```
N   V1,V2,...,Vk
```

y MASM reemplazará cada parámetro formal P1,P2, ..., Pk, por los símbolos V1,V2,...,Vk, respectivamente, en cada una de las instrucciones del cuerpo de N en donde éstos aparezcan.

Ejemplos

1. Se tiene la macro con un parámetro formal "valor"

```
función  macro valor
         mov ah,valor
         int 21h
         endm
```

Luego si en el programa fuente se encuentra el texto

```
         función 9
```

MASM lo sustituirá por:

```
         mov ah,9
         int 21h
```

2. La siguiente macro ordena en forma ascendente dos datos de memoria (8 bits):

```
ordena       macro  dato1,dato2
             mov AH,dato1      ; mueve dato1 a AH
             cmp AH,dato2      ; compara dato1,dato2
             jbe  ordenados    ; si dato1 < =dato2:ir a ordenados
             xchg AH,dato2     ; no: intercambiarlos
             mov  dato1,AH     ; y poner dato2 en dato1
ordenados:
             endm
```

6.2. LA DIRECTIVA LOCAL DENTRO DE MACROS

Si se emplean etiquetas en una macro, MASM repetirá éstos símbolos cada vez que reemplace la macro originando un error de símbolos definidos en forma múltiple. En este caso se emplea la directiva LOCAL dentro del cuerpo de la macro:

```
LOCAL   lista de etiquetas
```

con la que se instruye a MASM que considere a dichas etiquetas como símbolos locales, esto es, como símbolos distintos en las sucesivas sustituciones de la macro.

Ejemplo

Véase la macro imprime del programa que se presenta más adelante. Dicha macro se utiliza dos veces en la forma:

```
imprime lista,total
```

y contiene la etiqueta repetir definida localmente:

```
local repetir
```

6.3. OPERADOR $ CONTADOR DE POSICION

Durante el proceso de ensamblaje de instrucciones el operador $ representa el desplazamiento actual respecto del segmento de programa en donde aparece.

Ejemplo

Si se define la constante total dentro del segmento A mediante:

```
total equ $+5
```

su valor será el desplazamiento respecto del comienzo del segmento más 5.

6.4. INSTRUCCIONES LOOP Y LEA

La instrucción LOOP permite repetir un número n de veces la ejecución de un grupo de instrucciones.

Se usa en la forma:

```
                mov  CX, n
    etiqueta:
                instrucciones
                loop etiqueta
```

Efecto: Cada vez que se completa una ejecución del grupo de instrucciones, CX decrece en una unidad. El proceso se repite mientras CX sea distinto de cero.

Ejemplo

El siguiente grupo de instrucciones calcula en el registro AX la suma de los 10 primeros impares: 1,3,...,19.

```
                mov AX,0        ; suma inicial
                mov BX,1        ; primer impar
                mov CX,10       ; número de repeticiones
    otra_vez:   add AX,BX       ; sumar impar actual
                add BX,2        ; siguiente impar
                loop otra_vez
```

La instrucción LEA (LOAD EFFECTIVE ADDRESS) se usa para almacenar directamente la dirección efectiva (desplazamiento) de la fuente en el destino:

```
LEA destino,fuente
```

Además, permite que el operando fuente sea dado mediante expresiones con índices.

Ejemplos

1. lea si, arreglo ; mueve a si el desplazamiento de varia
   ; ble arreglo
2. lea dx, arreglo[bx] ; mueve a dx desplazamiento de
   ; arreglo + bx

6.5. PROGRAMA: Ordena arreglo de octetos

El siguiente programa ordena en forma ascendente un arreglo de octetos designado con el nombre lista.

Se han escrito dos macros:

1) ordena dato1,dato2

   que ha sido mostrada como ejemplo antes.

2) imprime arreglo, cuenta

   con los parámetros formales arreglo y cuenta para ser reemplazados por el nombre del área de datos y el tamaño de la misma, respectivamente.

   Se emplea dos veces en la misma forma:

   imprime lista,total1

   para imprimir los datos de la lista antes y después de ser ordenados.

```
; ***************  programa  ***************

ordena      macro  dato1,dato2
            mov AH,dato1
            cmp AH,dato2
            jbe   ordenados
            xchg  AH,dato2
            mov dato1,AH
ordenados:
            endm

imprime     macro arreglo,cuenta
            local repetir
            lea  si,arreglo
            mov  cx,cuenta
            mov  ah,2
repetir:    mov  dl,[si]          ; mueve octeto a dl
            add  dl,'0'           ; ajuste ASCII de número
            int  21h              ; ejecución
            inc  si               ; siguiente dirección
            loop repetir          ; lazo
            mov  dl,10            ; imprime cambio de línea
            int  21h
            mov  dl,13
            int  21h
            endm

pila   segment stack
       dw 20 dup(?)
pila   ends

codigo segment
       assume cs:codigo, ds:codigo

lista   db 9,8,7,6,5,4,3,2,1,0
total   equ $ - lista
lim     equ total - 1

comienzo:   push cs
            pop  ds
            imprime lista,total  ; imprime lista inicial
            lea  si,lista
            mov  cx,lim

lazo1:      mov bx,0
lazo2:      inc bx
            cmp bx,cx
            ja  fin_lazo2
            ordena [si],[si+bx]
            jmp lazo2
```

```
fin_lazo2:  inc si
            loop lazo1

            imprime lista,total    ; imprime lista ordenada
            mov  ah,4Ch
            int  21h
codigo      ends
            end      comienzo      ; ********  fin de programa  *********
```

6.6. EJERCICIOS

1. Escriba una macro cls para limpiar la pantalla y poner el cursor en la posición 0,0.

2. Escriba una macro cursor F,C para ubicar el cursor en la fila F y en la columna C.

   Indicación: Use

```
mov AH,2          ; función de ubicación de cursor
mov BH,0          ; página actual
mov DX,posición ; DH=fila,  DL=columna
int 10h           ; rutina de video (BIOS)
```

3. Escriba dos macros para leer e imprimir cadenas:

```
leer pcadena
imprimir pcadena
```

   en donde a través del parámetro formal pcadena se pasa la dirección de un arreglo de caracteres.

4. Escriba una macro que lea un número decimal en forma de cadenas de caracteres ASCII (máximo 4 dígitos) y obtenga su valor:

```
lee_número pcadena,pvalor
```

   en donde pcadena representa la dirección de la cadena y pvalor la variable donde se almacenará el valor del número.

5. Escriba una macro para crear un archivo de nombre dado y devuelva su descriptor : crea_archivo pnombre, pdescriptor.

   pdescriptor será -1 en caso que falle la operación de apertura.

6. Escriba una macro que calcule el número de caracteres que forman una cadena. Asuma que la cadena termina con el carácter 0 (no se cuenta) y que consta a lo sumo de 254 caracteres:

   longitud pcadena,pl

   en donde pcadena es la dirección de pcadena y pl es una variable de octeto para recibir el número de caracteres de la cadena.

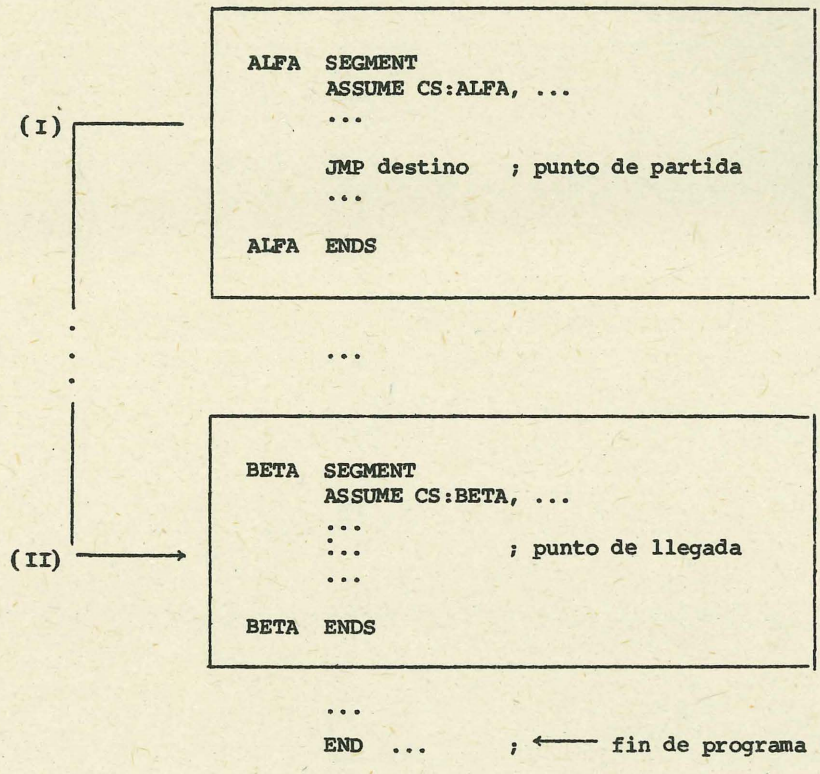
SUBRUTINAS Y LIBRERIAS

7.1. SALTOS LEJANOS (FAR)

Con la instrucción de salto: JMP destino

se puede saltar entre segmentos de código, es decir compuestos por instrucciones ejecutables, los que siempre son referidos por el registro CS. Al ejecutarse esta instrucción, el par CS:IP recibe la dirección real del destino, esto es, en CS se cambia la dirección del segmento de partida por la del segmento de destino y el registro IP recibe la dirección (desplazamiento) del lugar del destino.

Supongamos dos segmentos de código (en el mismo programa fuente) a los que denominaremos ALFA y BETA, respectivamente, y que desde un lugar de ALFA se desea transferir la ejecución del programa a otro punto en el segmento BETA:



a) Salto directo o por etiqueta

En (I) se escribe : JMP FAR PTR N_ET

En (II) : N_ET LABEL FAR

en donde N_ET es una etiqueta de tipo lejano (FAR) que representa la dirección del punto de llegada.

Así, destino = N_ET = dirección de llegada.

Ejemplo

```
datos       segment
  ma    db  "en segmento alfa ",10,13,"$"
  mb    db  "en segmento beta ",10,13,"$"
datos       ends


alfa        segment
            assume cs:alfa, ds:datos
inicio:     mov  ax,datos
            mov  ds,ax              ; segmento datos en ds
            lea  dx, ma             ; imprime mensaje ma
            mov  ah,9
            int  21h
            jmp  far ptr bb         ; salto lejano a la dirección
                                    ; de (la etiqueta) bb
alfa        ends


beta        segment
            assume cs:beta

            ;  dirección de segmento datos sigue en ds  ...

bb          label far               ; etiqueta es de tipo lejano
            lea dx, mb              ; imprime mensaje mb
            mov ah,9
            int 21h


            mov ah,4Ch              ; fin de proceso
            int 21h
beta        ends

            end inicio              ; fin de programa fuente
```

b) Saltos lejanos indirectos

Tiene la forma JMP DWORD PTR destino

en donde destino es una dirección de memoria que contiene la dirección del punto de llegada. Esto significa que en las dos primeras palabras (palabra doble o 4 octetos) a partir de la dirección de destino se guardan el desplazamiento y el segmento (en este orden) del punto de llegada.

La especificación de puntero a palabra doble (DWORD PTR) indica que el acceso se hará teniendo en cuenta a una palabra doble. Dicha especificación puede omitirse si destino es una variable de tipo palabra doble, es decir definida por la directiva DD.

Ejemplos

1. Si la variable de palabra dobre SIGUE ha sido declarada en el segmento de datos actual así:

```
SIGUE   DD LLEGADA          ; inicializa SIGUE con la dirección
                            ; de la etiqueta lejana LLEGADA
```

se puede escribir,

```
en (I)  del segmento ALFA:  JMP  SIGUE
en (II) del segmento BETA:  LLEGADA  LABEL  FAR
```

2. Si a partir de la dirección DS:BX, con BX=4000, se almacenan los octetos

00 10 80 60 ...

la instrucción JMP DWORD PTR [BX]

efectuará un salto a la dirección absoluta 6080:1000, esto es, hará CS=6080, IP=1000

3. JMP DWORD PTR [BP+DI]
la dirección de transferencia se halla contenida en SS:BP+DI.

7.2. LLAMADAS Y SUBRUTINAS LEJANAS (FAR)

En forma análoga a los saltos lejanos existen llamadas lejanas, esto es, a subrutinas que se hallan en otros segmentos de códigos. Tales subrutinas, en las que las instrucciones RET se consideran de retorno lejano, se declaran como procedimientos lejanos:

```
Nombre o etiqueta de subrutina  PROC FAR
```

La instrucción de llamada de subrutina: CALL destino

en donde con destino se refiere a la dirección de entrada en un procedimiento lejano, tiene el siguiente efecto:

1) Se apila la dirección de retorno como una palabra doble, es decir, primero se apila el valor de CS (segmento de código actual) y a continuación se apila la dirección (desplazamiento) de la instrucción que sigue a la de la llamada.

2) El par CS:IP recibe la dirección referida por destino, con lo cual se transfiere la ejecución a la subrutina. De este modo CS contiene la dirección del nuevo segmento de código.

3) Al encontrarse la instrucción de retorno (RET), se desapila la palabra doble en el par CS:IP, con lo cual se recupera la dirección previamente salvada en la pila y el programa puede continuar después del punto desde donde fue desviado.

Para una subrutina definida como un procedimiento lejano el programa MASM generará una instrucción de retorno lejano, esto es, que desapila en IP y CS.

a) Llamada directa o por etiqueta

En el punto de llamada del segmento actual se escribe:

```
CALL   FAR PTR   N_ET
```

y en el punto de entrada se declara a la subrutina del segmento de llegada en la forma:

```
N_ET    PROC FAR
        ...
        RET
        ...
N_ET    ENDP
```

en donde N_ET es una etiqueta de tipo lejano (FAR).

Ejemplo

En este programa desde el segmento ALFA se hace una llamada a la subrutina lejana bb.

```
imprime     macro  cadena
            lea dx,cadena
            mov ah,9
            int 21h
            endm

pila        segment  stack
     db     30 dup(?)
pila        ends

datos       segment
ma   db     "comienzo en segmento alfa ",10,12,"$"
mb   db     "en subrutina bb de segmento beta ",10,13,"$"
mc   db     "retorno a segmento alfa",10,13,"$"
datos       ends

alfa        segment
            assume cs:alfa, ds:datos
inicio:     mov ax,datos
            mov ds,ax
            imprime ma
            call far ptr bb        ;  llama a subrutina lejana bb

            imprime mc
            mov ah,4Ch             ;  fin de programa ejecutable
            int 21h
alfa        ends

beta        segment
            assume cs:beta

bb          proc far               ;  bb es un  procedimiento lejano (FAR)
            imprime mb
            ret                    ;  y por lo tanto RET es lejano
bb          endp

beta        ends

            end inicio             ;  fin de programa fuente
```

b) Llamadas lejanas indirectas

Tienen la forma CALL DWORD PTR destino

en donde destino es una dirección de memoria que contiene la dirección del punto de llegada. Como en los saltos lejanos, esto significa que en las dos primeras palabras (palabra doble o 4 octetos), a partir de la dirección de destino, se tienen almacenados el desplazamiento y el segmento (en este orden) del punto de entrada a la subrutina.

Recordemos que estas subrutinas deben estar contenidas en procedimientos lejanos (PROC FAR) de manera que las instrucciones RET sean interpretadas por MASM de retorno lejano.

Ejemplo 1

Consideremos un procedimiento lejano, de nombre o etiqueta PROC_LEJANO, definido así:

```
PROC_LEJANO   PROC FAR
              ...
ET_CERCANA:   ...        ; etiqueta cercana

              RET        ; este es un retorno lejano
              ...

PROC_LEJANO   ENDP
```

Supongamos que en una dirección de memoria se tiene almacenados (como palabra doble o cuatro octetos) el desplazamiento y la dirección del segmento del punto de entrada a la subrutina (por ejemplo, PROC_LEJANO o ET_CERCANA). Entonces dicha dirección de memoria es dada por una variable RUT1, DS:BS+SI ó SS:BP+2, se puede llamar a la subrutina mediante:

```
CALL DWORD PTR RUT1   ; si RUT1 es de tipo DD se puede omitir
                      ; la especificación DWORD PTR
CALL DWORD PTR [BX+SI]
CALL DWORD PTR [BP+2]
```

respectivamente.

Ejemplo 2

En el siguiente programa se llama a la subrutina lejana nbb que se halla dentro del procedimiento lejano bb. La dirección de nbb se almacena en los cuatro octetos de la variable salto.

La llamada se hace mediante call dword ptr salto.

```
                ; *************  programa  ****************************

imprime         macro  cadena
                lea dx,cadena
                mov ah,9
                int 21h
                endm

pila            segment  stack
        db       30 dup(?)
pila            ends

datos           segment
ma      db       "comienzo en segmento alfa ",10,13,"$"
mbb     db       "en subrutina bb de segmento beta ",10,13,"$"
mnbb    db       "ingreso en nbb de procedimiento bb ",10,13,"$"
mc      db       "retorno a segmento alfa",10,12,"$"
salto   db       4 dup (?)              ; para dirección de subrutina nbb
datos           ends

alfa            segment
                assume cs:alfa, ds:datos
inicio:         mov ax,datos
                mov ds,ax
                imprime ma
                lea si,salto           ; dirección de variable en si
                mov word ptr [si],offset nbb     ; desplazamiento
                mov [si+2],seg nbb               ; y segmento de nbb
                                                 ; puestos en "salto"
                call dword ptr salto             ; ← llamada lejana
                imprime mc
                mov ah,4Ch             ; fin de proceso
                int 21h
alfa            ends

beta            segment
                assume cs:beta
bb              proc far               ; bb es un procedimiento lejano (FAR)
                imprime mbb
                imprime mnbb           ; aquí comienza subrutina nbb
                ret                    ; RET es un retorno lejano
bb              endp
beta            ends

                end  inicio            ; fin de programa fuente
```

7.3. PASAJE DE PARAMETROS A SUBRUTINAS

Frecuentemente se requiere que las subrutinas formen subprogramas independientes del resto del programa, de modo que las tareas que en ellas se realizan no afecten los datos de otras partes del programa. Para lograr este propósito se emplea el método de comunicarse con las subrutinas utilizando la pila como medio para transmitir un grupo de valores o parámetros. Es decir, antes de llamar a una subrutina, se apilan dichos datos y luego pueden ser accedidos en el cuerpo de la subrutina. Allí, estos datos pueden ser utilizados y posiblemente modificados. Al regresar de la subrutina se deben desapilar (o decrecer la cima de la pila) en el número de datos apilados previamente a la llamada.

El proceso para pasar parámetros a una subrutina se describe a continuación. Supongamos que se trata de pasar n datos (palabras) D1,...,Dn a una subrutina:

1) Se apilan sucesivamente D1,..., DN (SP disminuye en 2n)

2) Se ejecuta la llamada a la subrutina (SP disminuye en 2 ó en 4, según sea una llamada cercana o lejana: en el primer caso se apila IP, y en el segundo caso CS y luego IP).

3) Al comenzar la subrutina se apilan los registros cuyos valores se deseen preservar al término de la subrutina, sólo si son modificados allí. Estos registros son usualmente los de segmentos y el registro BP para acceder a los datos de la pila (SP diminuye en 2*m, siendo m el número de registros apilados)

4) Para acceder a los datos apilados D1,... Dn, utilizando el registro de base BP, se pone en BP el valor actual de SP: MOV BP,SP

Entonces los valores de Dn,..., D1 (en este orden) se hallan en las direcciones:

```
BP + nret + 2*m            ; dirección de Dn
BP + nret + 2*m + 2        ; ...
...                        ; ...
BP + nret + 2*m + 2*(n-1)  ; dirección de D1
```

respectivamente.

Aquí, nret es 2 ó 4 de acuerdo a lo indicado en **2)**

5) Antes de retornar de la subrutina se deben desapilar los registros apilados (SP aumenta en 2*m, y por tanto apunta a la dirección de retorno).

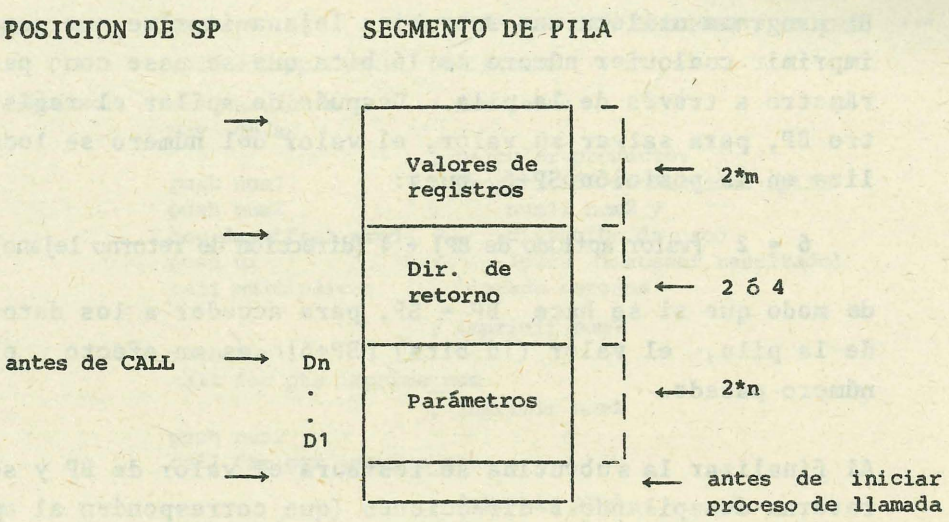
6) Al regresar de la subrutina con la instrucción RET, se desapila la dirección de retorno (SP aumenta en 2 ó en 4, y ahora apunta al último dato apilado : DN)

Si se requiere se puede acceder a los nuevos valores de los datos apilados.

A continuación es conveniente restaurar la cima de la pila a la posición que tenía antes de iniciar el apilamiento de parámetros, mediante instrucciones de desapilamiento efectuadas después del retorno. Esto también puede hacerse en el mismo instante del retorno usando:

```
RET 2*n
```

en cuyo caso se regresa y SP aumenta en 2*n.



7.4. PROGRAMA: Muestra uso de parámetros en subrutinas

En el siguiente programa se calcula el producto de dos números almacenados en las variables num1 y num2. El resultado se obtiene en la variable prod.

Se ha escrito una subrutina cercana, llamada multiplicar, para efectuar la multiplicación de dos números cualesquiera. Esta subrutina utiliza tres parámetros: los dos primeros se reservan para recibir los valores de los factores y el tercero contiene la dirección de memoria en donde se debe alojar el producto resultante. Al comenzar la subrutina se salva en la pila el valor de BP, de manera que el último parámetro apilado se ubica en la posición SP+4:

4 = 2 (valor apilado de BP) + 2 (dirección de retorno cercano)

Si luego se hace BP=SP, se tendrá (valores de 16 bits):

[BP+4]= tercer parámetro apilado ; contiene dirección de resultado prod
[BP+6]= segundo parámetro apilado; valor de segundo factor
[BP+8]= primer parámetro apilado ; valor de primer factor

El programa utiliza una subrutina lejana imprime_num para imprimir cualquier número de 16 bits que se pase como parámetro a través de la pila. Después de apilar el registro BP, para salvar su valor, el valor del número se localiza en la posición SP+6, pues:

6 = 2 (valor apilado de BP) + 4 (dirección de retorno lejano)

de modo que si se hace BP = SP, para acceder a los datos de la pila, el valor (16 bits) [BP+6] es en efecto el número pasado.

Al finalizar la subrutina se restaura el valor de BP y se retorna desapilando 2 direcciones (que corresponden al apilamiento del número pasado). Así, SP sigue con el mismo valor que tenía antes de iniciarse el proceso de preparación para la llamada.

En esta subrutina hemos empleado el método de obtener los dígitos del número a partir de sus unidades de orden inferior. Estos no se imprimen conforme se calculan, pues en caso de hacerlo el número aparecería escrito con sus dígitos en orden inverso. Simplemente primero se apilan todos los dígitos y luego se procede a desapilar e imprimir sucesivamente cada dígito.

```
; ************** PROGRAMA FUENTE ********************

pila  segment stack              ; ****** SEGMENTO DE PILA *********
       db 50 sup (?)             ;        con 50 octetos
pila  ends

datos segment                    ; ****** SEGMENTO DE DATOS ********
 num1   dw  14                   ; primer número
 num2   dw  30                   ; segundo número
 prod   dw  ?                    ; producto
datos ends
```

```
programa    segment                 ; **** SEGMENTO PROGRAMA PRINCIPAL ****
            assume cs:programa, ds:datos
comienzo:   mov  ax,datos           ; ds= datos
            mov  ds,ax
                                    ; calcular producto:
            push num1               ; preparación de llamada: apila
            push num2               ;     num1, num2 y
            mov dx,offset prod      ;     dirección de prod
            push dx                 ;     (para almacenar resultado)
            call multiplicar        ; llamada cercana
                                    ; imprimir num1
            push num1
            call far ptr imprime_num
                                    ; imprimir num2
            push num2
            call far ptr imprime_num
                                    ; imprimir prod
            push prod
            cal far ptr imprime_num
                                    ; fin de proceso
            sub ax,ax               ; código de retorno normal (0)
            mov ah,4Ch
            int 21h
                                    ; subrutina cercana, usa 3 parámetros
multiplicar proc near
            push bb                 ; salva bp
            mov  bb,sp              ; valor de sp en bp para acceder a datos
            mov  ax, [bp+8]         ; ax recibe valor de num1 = [bp+8]
            mul  word ptr [bp+6]    ; multiplica ax con valor de num2= [bp+6]
            mov  bx, [bp+4]         ; dirección de variable prod
            mov  [bx],ax            ; almacena resultado en prod
            pop  bp                 ; restaura bp
            ret  6                  ; retorna y restaura sp (desapila 6)
multiplicar endp
programa    ends                    ; *** FIN DE SEGMENTO PRINCIPAL ********
subprograma segment                 ; *** SUBPROGRAMA EN OTRO SEGMENTO *****
            assume cs:subprograma
imprime_num proc far                ; subrutina lejana, usa 1 parámetro
            push bp                 ; salva bp
            mov  bp,sp              ; valor de sp en bp para acceder a datos
            mov  ax, [bp+6]         ; ax recibe valor para imprimir
            mov  bx,10              ; 10 en bx para dividir
            sub  cx,cx              ; cx=0, contador de número de dígitos
otro_dig:   sub  dx,dx              ; dx=0, extiende dividendo a 32 bits
            div  bx                 ; divide dx ax entre 10: cociente=ax,
                                            resto en dx  (o en dl),
            add  dl,"0"             ; ajuste a carácter de dígitos obtenidos
            cmp  ax,0               ; prueba si hay más dígitos
            jne  otro_dig
            mov ah,2                ; función DOS para imprimir carácter
```

```
imprime_d:  pop  dx              ; imprime dígitos ASCII en el orden
            int  21h             ; correcto, se controla con CX
            loop imprime_d       ; (desapila todos los dígitos)

                                 ; imprimir cambio de línea (10 13)
            mov  dl,10           ;      ah sigue con 2
            int  21h
            mov  dl,13
            int  21h

            pop  bp              ; restaura valor de bp
            ret  2               ; retorna y restaura valor de sp

imprime_n   endp

subprograma ends                 ; ****** FIN DE SUBPROGRAMA **********

            end comienzo         ; ****** FIN DE PROGRAMA FUENTE ******
```

7.5. INCLUSION DE PROGRAMAS FUENTES: La orden INCLUDE

La directiva: INCLUDE np

en donde np es el nombre (o especificación) de otro archivo fuente, instruye al programa ensamblador a leer y ensamblar las instrucciones contenidas en np, como si estuvieran escritas en este lugar. Cuando se termina de ensamblar np se procede con las instrucciones que siguen a INCLUDE.

La formación y uso de archivo de inclusión es conveniente cuando éstos contienen objetos (constantes, variables, programas, etc) que pueden ser usados por otros archivos, sin tener que reescribirlos.

Ejemplo

El procedimiento lejano imprime_num escrito en el programa anterior puede ser salvado como un archivo para ser incluído en cualquier programa que requiera la impresión de un número. Si dicho procedimiento, comprendido entre las líneas (inclusive):

```
imprime_num proc far
...
imprime endp
```

se salva en un archivo con el nombre salida.asm, puede entonces reemplazarse esta sección de programa por su equivalente:

```
INCLUDE SALIDA.ASM
```

7.6. PROGRAMACION MODULAR: Programas módulos. Enlazamiento: Programa LINK

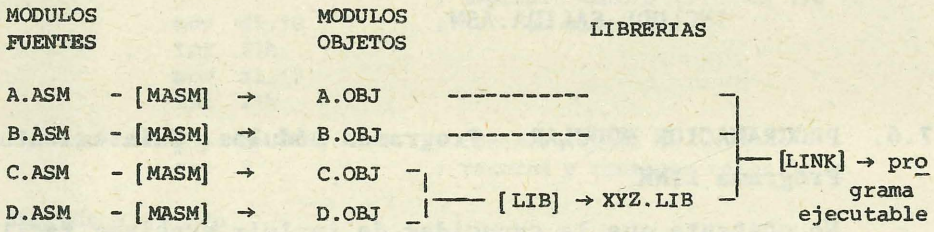
No obstante que la capacidad de incluir archivos facilita las tareas de diseño en la programación, usando instrucciones que ya han sido salvadas en otros archivos, tiene la desventaja de tener que compilarse tales instrucciones cada vez que sean incluídas: El programa ensamblador las volverá a traducir. De mayor utilidad son los programas ya compilados que pueden ser directamente empleados por otros programas. Estos programas (de tipo .OBJ) son almacenados individualmente o formando parte de un grupo de programas llamado librería, de manera que al momento de generar el programa ejecutable que los requiera puedan ser copiados o extraídos. Esta tarea y la de enlazar o unir adecuadamente varios módulos objetos es realizada por un programa enlazador o montador tal como LINK.

Para el lenguaje ensamblador cualquier programa fuente (que termina con la directiva END) es un módulo fuente. Estos módulos pueden ser compilados (con MASM ó ASM) separadamente generándose programas de tipo .OBJ llamados módulos objetos o simplemente módulos. Varios módulos objetos pueden ser enlazados para obtener un programa ejecutable. En este caso exactamente uno de los módulos fuentes (llamado principal, pues gobierna el procesamiento del programa final), debe contener necesariamente la dirección de entrada del programa:

```
END entrada a programa
```

como se ha mostrado en todos los programas desarrollados.

El siguiente esquema ilustra gráficamente el proceso de generación de un programa ejecutable:



En la figura se muestra 4 módulos fuentes A.ASM, B.ASM, C.ASM y D.ASM, que son procesados por el programa ensamblador MASM (ó ASM), para obtener los respectivos módulos objetos. Los módulos objetos C.OBJ y D.OBJ son puestos en la librería XYZ.LIB empleando el programa LIB. Finalmente se enlazan los módulos A.OBJ, B.OBJ y la librería XYZ.LIB para producir el programa ejecutable.

Los módulos objetos se componen de instrucciones de máquina (numérica) e información relativa a los símbolos que allí se usan (etiquetas, variables, segmentos de programas, subrutinas, etc). Algunas instrucciones pueden estar incompletas, por ejemplo algunas direcciones no son conocidas (direcciones reubicables y direcciones de objetos en otro módulo) que deben ser calculadas o resueltas por el programa enlazador. Por esto es preciso que en tales módulos fuentes se incluya información adicional que resuelva el programa de comunicación entre ellos en el momento del enlace.

En esta breve exposición sólo mencionaremos las siguientes directivas de comunicación:

1) Combinación de segmentos de programas.

   Mediante la declaración de segmentos:

```
NS  SEGMENT PUBLIC ; NS es el nombre del segmento de programa
    ...
NS  ENDS
```

se instruye al programa de enlace que una o concatene todos los segmentos de nombre NS que aparezcan en los distintos módulos objetos en uno sólo.

De otro modo (si no se especifica PUBLIC) el segmento no será combinado con otros y tendrá su propio segmento de memoria.

2) Por defecto los símbolos (variables, constantes, etiquetas, procedimientos) son locales al módulo en donde se definen. Esto es son desconocidos fuera de dicho módulo. Si se requiere que un símbolo sea utilizado por otros módulos (símbolo global) se debe especificar ello con la directiva PUBLIC:

```
PUBLIC  ns          ; ns es el nombre del símbolo
```

y en los otros módulos en los que ns será usado:

```
EXTRN   ns : tipo
```

en donde tipo, por ejemplo, puede ser: byte, word, dword, near, far, abs (para nombres de constantes).

Nota

1) Un símbolo puede ser declarado público en exactamente un módulo y puede ser declarado de tipo externo en todos los módulos en los que se requiera su acceso.

2) Las directivas PUBLIC y EXTRN pueden emplearse dentro o fuera de los segmentos de programa. Sin embargo, si se aplica EXTRN dentro de un segmento de programa, se asume que los símbolos afectados se encuentran en dicho segmento.

Ejemplo

Si en el programa A se declaran los símbolos públicos:

```
PUBLIC  num1, num2, imprime_num, otra_vez
```

entonces en el programa B se puede acceder a tales objetos del módulo A mediante:

```
EXTRN  num1:word, num2:word ; variables de tipo word o palabra
EXTRN  imprime_num:far      ; etiqueta o procedimiento de tipo lejano
EXTRN  otra_vez:near        ; etiqueta o procedimiento de tipo cercano
```

7.7. PROGRAMA: Muestra enlace de módulos objetos con LINK

1) El siguiente programa, al que llamaremos IO.ASM será compilado como un módulo separado IO.OBJ. Contiene dos subrutinas de tipo cercano, una lee_cad, para leer una cadena por el teclado, y otra, imprime_cad, para imprimir una cadena por la pantalla. Ambas serán llamadas pasando la dirección de la cadena a través de la pila: Primero se apila el segmento y luego el desplazamiento de dicha dirección.

   Puesto que estas subrutinas serán accedidas desde otros módulos se las declara de tipo público. Además, se ha supuesto llamadas cercanas y por tanto el segmento de programa, de nombre codigo, en donde se definen se declara de combinación pública, de manera que pueda unirse en un sólo segmento de memoria con aquéllos que tengan el mismo nombre y sean públicos en otros módulos de programas.

Usando MASM se compila el siguiente módulo fuente y se obtiene el módulo objeto IO.OBJ.

```
; **********  programa  IO.ASM  **************

codigo  segment  public
        assume cs:codigo
        public lee_cad, imprime_cad ; símbolos públicos pueden ser usados
                                    ; (como externos) desde otros módulos
lee_cad   proc near

        push bp                 ; salva valor actual de bp
        mov  bp,sp              ; bp recibe la cima de la pila para acce
                                ; der a los parámetros

        mov  dx, [bp+4]         ; establece dirección de lectura en DS:DX
        mov  ax, [bp+6]         ; NOTA: Se puede emplear: lds dx, dwo
        mov  ds, ax             ; dword ptr [bp+4] en lugar de estas 3 ins
                                ; trucciones
        mov  ah,10              ; imprimir DS:DX
        int  21h
        pop  bp                 ; restaurar valor de bp y poner cima en la
                                ; dirección de retorno
        ret  4                  ; regresar desapilando parámetros
lee_cad   endp

imprime_cad  proc near
        push bp
        mov  bp,sp
        lds  dx,dword ptr [bp+4]
        mov  ah,9
        int  21h
        pop  bp
        ret  4
imprime_cad  endp
codigo       end                ; ***** fín de IO.ASM *******
```

2) A continuación se presenta el programa fuente PRUEBA.ASM que hace uso de las subrutinas lee_cad e imprime_cad ya compiladas en el módulo IO.OBJ.

Nótese la declaración:

```
extrn imprime_cad : near,  lee_cad :near
```

para especificar que se tratan de etiquetas externas de tipo cercano.

Este programa contiene un segmento de código público y con el mismo nombre (codigo) que el que se halla en IO.ASM, y por tanto en el programa ejecutable ambos serán integrados en uno sólo, como ya ha sido indicado.

Compile el siguiente programa con MASM y obtenga PRUEBA.OBJ

```
; *******  programa  PRUEBA.ASM ********

datos segment
  cadena       db 80,0, 80 dup("$")
  linea_nueva  db 13,10,"$"
datos ends

s0  segment stack
   sb 100 dup(?)
s0  ends

codigo segment  public          ; segmento público para ser combinado
      ; con los que tengan igual declaración: codigo segment public
      ; en otros módulos

      assume cs:codigo, ds:datos
      extrn  lee_cad:near, imprime_cad:near  ; etiquetas externas
           ; de tipo cercano declaradas con PUBLIC en otro módulo
comienzo:
      mov  ax, datos
      mov  ds, ax
      push ds                   ; apilar dirección de cadena:segmento
      mov  ax, offset cadena    ;  y desplazamiento
      push ax
      call lee_cad              ; llamada cercana subrutina lee_cad

                                ; imprimir cambio de línea
      push ds                   ;   apilar dirección de linea nueva
      mov  ax, offset linea_nueva
      push ax
      call imprime_cad

           ; imprimir texto leído, empieza  en DS:cadena +2
      mov  ax, offset cadena
      add  ax, 2
      push ds                   ; apilar dirección
      push ax
      call imprime_cad

      mov  ah, 4Ch              ; fin de proceso
      int  21h
codigo ends
       end  comienzo    ; fin de PRUEBA.ASM y dirección de entrada
```

3) Se enlazan los dos módulos objetos PRUEBA.OBJ y IO.OBJ usando LINK.EXE, ingresando directamente:
LINK PRUEBA + IO, PRUEBA; <enter>

o de otro modo LINK <enter>
y respondiendo con:

```
Object modules [.OBJ] : PRUEBA +IO <enter>
Run file [prueba.EXE] : <enter>
List file   [NUL.MAP] : <enter>
Libraries      [.LIB] : <enter>
```

Resultando entonces el programa ejecutable PRUEBA.EXE

7.8. MANEJO DE LIBRERIAS: Programa LIB

Conforme se ha indicado en la sección anterior varios modulos objetos pueden agruparse en un único archivo formando lo que se suele llamar una librería de módulos objetos. Hay programas manejadores de librería que permiten crearlas y añadir, eliminar, extraer o reemplazar módulos, y también proporcionar un listado de su directorio, lo cual comprende a los módulos componentes con los objetos definidos en cada uno de ellos (por ejemplo, subrutinas).

Cuando se enlazan varios módulos con el programa LINK, si éste encuentra un símbolo externo no contenido en los módulos objetos presentes, lo buscará en las librerías que se indiquen a fin de extraer el módulo completo que lo contenga e incorporarlo.

Mostramos ahora un ejemplo de cómo formar una librería, a la que designaremos por ES.LIB, que contenga inicialmente el módulo objeto IO.OBJ.

Corra el programa LIB.EXE, por ejemplo así:

```
(1)  LIB <enter>
(2)  Library name: ES <enter>   [Nombre de librería es ES.LIB]
(3)  Library file does not exist. Create ? Y<enter>        [sí]
(4)  Operations: + IO <enter>          [añadir módulo IO.OBJ ]
(5)  List file: ES.LST <enter>   [listar directorio de ES.LIB]
                                      en archivo de texto ES.LST]
```

Se confecciona así la librería ES.LIB con un único módulo IO.OBJ, el cual contiene a las dos subrutinas mencionadas.

Podemos usar esta librería para enlazarla con el módulo PRUEBA.OBJ y ya no se requiere IO.OBJ en la lista de módulos objetos pues éste se obtendrá de la librería que lo contiene. Ingrese:

```
LINK PRUEBA,PRUEBA,NUL,ES <enter>
```

o ejecute LINK respondiendo:

```
LINK <enter>
Object modules [.OBJ] : PRUEBA <enter>
Run file  [prueba.EXE]: <enter>
List file     [NUL.MAP]: <enter>
Libraries        [.LIB]: ES <enter>
```

para obtener el programa ejecutable PRUEBA.EXE.

7.9. EJERCICIOS

1. Compile separadamente los dos programas fuentes siguientes y póngalos en la librería CONSOLA.LIB :

a) PANTALLA.ASM con subrutinas de manejo simple de pantalla:

	SUBRUTINA	DESCRIPCION	
a1)	_borrar	Efecto :	Borra la pantalla
		Uso :	Llamar con call _borrar
a2)	_cursor	Efecto :	Ubica el cursor en la posición x (columna), y (fila)
		Uso :	Apilar x, luego y Llamar con call _cursor

b) ENT-SAL.ASM, con subrutinas de entrada y salida de datos:

	SUBRUTINA	DESCRIPCION	
b1)	_lee_cad	Efecto :	Lee cadena en dirección M:N con número L de caracteres especificado.
		Convenio:	Cadena leída termina con "$" y ax retorna con número de caracteres leídos (sin contar "$").
		Uso :	Apilar segmento M, luego apilar desplazamiento N y finalmente apilar L. Llamar con call _lee_cad
b2)	_imprime_cad	Efecto :	Imprime cadena dada en dirección M:N
		Convenio:	La cadena debe terminar con "$"
		Uso :	Apilar M y luego N Llamar con call _imprime_cad
b3)	_lee_num	Efecto :	Lee número por teclado (máximo 5 dígitos)
		Convenio:	Ignora blancos iniciales. Valor del número se devuelve en registro ax
		Uso :	Llamar con call _lee_num
b4)	_imprime_num	Efecto :	Imprime número U (16 bits)
		Uso :	Apilar U Llamar con call imprime_num

Nota. Todas las subrutinas se definen públicas de tipo lejano en segmentos no públicos. Por lo tanto, para ser accedidas desde otros programas se debe declararlas de alcance externo y tipo lejano:

```
extr _borrar:far ,      _cursor:far,        _lee_cadena:far
extr _imprime_cad:far,  _lee_num:far,       _imprime_num:far
```

El programa PANTALLA.ASM se desarrolla a continuación:

```
; ****** módulo PANTALLA.ASM ******

public _borrar, _cursor

seg_pant    segment
            assume cs:seg_pant

 _cursor    proc far            ; subrutina lejana: ya se hallan apilados
                                ; parámetros x, y, y dirección de retorno
                                ; (segmento y desplazamiento)
            push bp             ; salvar bp para acceder a los parámetros
                                ; en la pila: x  en SS:BP+8, y en SS:BP+6
            mov  bp, sp
            mov  dx, [bp+8]     ; obtiene x  (16 bits): dl=valor de x
            mov  ax, [bp+6]     ; obtiene y  (16 bits): al=valor de y
            mov  dh, al         ; par dh dl contiene y x
            mov  ah, 02         ; función para ubicar cursor
            mov  bh, 0          ; en página 0
            int  10h            ; ejecutar (rutina de video 10h)
            pop  bp             ; restaurar bp
            ret 4               ; retorno lejano desapilando los parámetros
 _cursor    endp

 _borrar    proc far
            mov  ah,06          ; función para subir texto de ventana
            mov  al,0           ;            todas las líneas
            mov  bh,7           ; atributo normal
                                ; definición de ventana (toda la pantalla)
            mov  cl,0           ; esquina superior x=0
            mov ch,0            ;                  y=0
            mov  dl,79          ; esquina inferior x=79
            mov  dh,25          ;                  y=25
            int  10h            ; ejecución

                                ; y ubicar cursor en 0  0
            mov  ax,0
            push ax             ; x=0
            push ax             ; y=0
            call _cursor

            ret
 _borrar    endp

seg_pant    endp
            end                 ;  ****** fin de PANTALLA.ASM ******
```

Y el programa ENT-SAL.ASM es el siguiente:

```
;  ****** módulo  ENT-SAL.ASM  ******

public _lee_cad, _imprime_cad, _lee_num, _imprime_num

seg_entsal  segment
            assume cs: seg_entsal

_lee_cad    proc far
            jmp  lee_cad1

tempcad1    db 255 dup(?)       ; área temporal de datos para lectura de
                                ; cadena
lee_cad1:   push bp
            mov  bp,sp          ; salvar bp para acceder a parámetros

            push ds             ; salvar registros para tratar cadenas
            push es

                        ;  leer cadena en ds:dx=dirección de tempcad1

            mov  ax, cs
            mov  ds, ax
            mov  dx, offset tempcad1
            mov  ax, [bp+6]     ; al=número de caracteres a ser leídos
            mov  si, dx
            mov  [si],al        ; número en inicio de temp
            mov  ah, 10         ; leer cadena en ds:dx = ds:si
            int  21h            ; número de caracteres leídos en ds:si+1
            mov ch,0            ; pasarlo a cx
            mov cl,[si+1]
            mov ax, cx          ; y copiar en ax=longitud de cadena para
                                ; retorno

                        ;  copiar cadena de ds:si  a  es:di

            add si,2            ; ds:si=dirección de cadena fuente
            les di,[bp+8]       ; es:di=dirección de cadena destino
            cld                 ; establecer índices crecientes
            rep movsb           ; copiar cx octetos
            mov byte ptr es:[di],"$"  ; añadir "$"

            pop es              ; restaurar registros
            pop ds
            pop bp
            ret 6               ; retornar y desapilar parámetros
_lee_cad    endp
```

```
_imprime_cad    proc far
                push bp
                mov  bp,sp
                push ds
                lds  dx, [bp+6]  ; ds:dx=dirección de cadena
                mov  ah,9        ; imprimirla
                int  21h
                pop  ds          ; recuperar registros salvados
                pop  dp
                ret  4
_imprime_cad    endp

_lee_num        proc far
                jmp  lee_num1
  tempcad2      db  10 dup(?)
  lee_num1:     push pb
                mov  bp,sp
                push cs          ; apilar dirección de tempcad2
                mov  ax,offset tempcad2
                push ax
                mov  ax,6        ; apilar número máximo de
                push ax          ; caracteres a leer
                call _lee_cad
                mov  cx,ax       ; cx=ax=número de caracteres leídos
                mov  ax,0        ; empezar con número=0 en ax
                mov  si,offset tempcad2  ; apuntar al comienzo de tempcad2
                                         ; para acceder a sus componentes
                                         ; con cs:[si]

   blancos:     cmp  cx,0                ; ignorar blancos iniciales
                je   fin
                cmp byte ptr cs:[si]," "
                jne  digitos
                inc  si
                dec  cx
                jmp short blancos

                ;  conversión de cadena a número

digitos:        mov  bx,10               ; bx=10 para multiplicar
                cmp  cx,0
                je   fin
                mul  bx                  ; multiplicar ax por 10
                mov  dh,0
                mov  dl,cs:[si]
                sub  dl,"0"
                add  ax,dx               ; y sumarle el siguiente dígito
                inc  si
                dec  cx
                jmp  digitos
fin:            pop  bp                  ; restaurar bp
                ret                      ; y retornar
                                         ; ax=valor de número
  _lee_num      endp
```

```
_imprime_num    proc far
                push bp
                mov  bp,sp
                mov  ax,[bp+6] ; ax recibe valor de número a imprimir
                mov  bx,10     ; bx=10 como divisor
                sub  cx,cx     ; contador de dígitos se pone a 0
otro_dig :      sub  dx,dx     ; dx=0 extiende dividendo a 32 bits
                div  dx        ; divide dx ax entre 10
                               ; resto en dx (dl), cuociente en ax (al)
                add  dl,"0"
                push dx
                inc  cx
                cmp  ax,0
                jne  otro_dig
                mov  ah,2      ; función de impresión de caracteres
impr_dig:       pop  dx        ; dl=carácter a imprimir
                int  21h       ; ejecutar
                loop impr_dig  ; repetir impr_dig cx veces
                pop  bp
                ret  2
_imprime_num    endp

seg_entsal      ends

                end    ; ******  fin de ENT-SAL.ASM  ******
```

Nota

1) Después de compilar con MASM los dos programas fuentes individualmente, resultando PANTALLA.OBJ y ENT-SAL.OBJ, ejecute el programa LIB.EXE para construír la librería CONSOLA.LIB que contenga dichos módulos, respondiendo con:

Operations: +PANTALLA + ENT-SAL <enter>

2) Los programas que usen esta librería deberán incluir necesariamente un segmento de pila que contenga al menos 40 octetos o bytes para las llamadas a las subrutinas de la librería. Así, el tamaño del segmento de pila debe ser como mínimo: 40 + número de octetos que se requiera en el código del programa.

   Si no se realizan llamadas mutuas (o recursivas) entre las subrutinas del programa, el tamaño total del segmento de pila puede fijarse en 256 ó 100h octetos.

2. El siguiente programa PRUEBA.ASM realiza las siguientes funciones:

(1) borra la pantalla,
(2) imprime el mensaje "Ingrese nombre " en 10 5
(3) lee un nombre
(4) en las posiciones 10 6 y 10 7 imprime los mensajes "Ingrese números " y lee dos números, respectivamente
(5) en laposición 10 8 imprime el mensaje "Hola " seguido por el nombre leído en (3)
y (6) finalmente, en 10 9 imprime "El resultado de la suma es " seguido por dicho valor.

El programa usa las subrutinas de la librería CONSOLA.LIB.

Para simplificar la redacción del programa se han formado varias macros (cursor, lee_cad, imprime_cad, imprime_num) con las subrutinas que emplean parámetros en las llamadas. Por ejemplo:

```
cursor  macro  columna,fila
    mov  ax, columna
    push ax
    mov  ax, fila
    push ax
    call _cursor          ;  llamada a subrutina _cursor
    endm
```

Compile el programa con MASM y genere el programa ejecutable con

LINK: LINK PRUEBA,PRUEBA,,CONSOLA<enter>

```
; ****** programa PRUEBA.ASM ******

        extrn _borrar:far, _cursor:far, _lee_cad:far, _imprime_cad:far
        extrn _lee_num:far, _imprime_num: far

cursor  macro  columna, fila
        mov ax, columna
        push ax
        mov  ax, fila
        push ax
        call _cursor
        endm

lee_cad macro  cadena, max
        mov  ax,seg cadena
        push ax
        mov  ax, offset cadena
        push ax
        mov  ax,max
        push ax
        call  _lee_cad
        endm

imprime_cad macro  cadena
        mov  ax, seg cadena
        push ax
        mov  ax, offset cadena
        push ax
        call _imprime_cad
        endm

imprime_num macro num
        mov  ax,num
        push ax
        call _imprime_num
        endm


pila segment stack
        db 80 dup(?)
pila ends

datos   segment
        num1 dw ?
        num2 dw ?
        resultado dw ?
        mens_nombre db "Ingrese nombre ","$"
        mens_saludo db "Hola ","$"
        mens_num    db "Ingrese números", "$"
        mens_res    db "Resultado de suma es ","$"
        nl          db 10,13,"$"                    ; nueva línea
        nombre db 80 dup(?)
datos ends
```

```
codigo      segment
            assume cs: codigo, ds:datos

comienzo:   mov  ax, datos
            mov  ds, ax
            call _borrar                ; borrar pantalla
            cursor 10,5                 ; imprimir mensaje en 10 5
            imprime_cad mens_nombre
            lee_cad     nombre,40       ; leer cadena en variable nombre
                                        ; con 40 caracteres (máximo)

            cursor 10,6                 ; imprimir mensaje en 10 6
            imprime_cad  mens_num
            call  _lee_num              ; y leer número, valor en ax
            mov  num1,ax                ; pasarlo a num1

            cursor 10,7                 ; igual con otro número
            imprime_cad  mens_num
            call _lee_num
            mov  num2,ax

            add  ax,num1                ; sumarlos y
            mov  resultado,ax           ; almacenar suma en resultado

            cursor 10,8                 ; imprimir saludo en 10 8
            imprime_cad mens_saludo
            imprime_cad nombre

            cursor 10,9                 ; imprimir mensaje y resultado
            imprime_cad mens_res
            imprime_num resultado
            imprime_cad nl              ; imprimir cambio de línea

            mov ah, 4ch                 ; fin de proceso
            int 21h
codigo      ends
            end comienzo      ; ****** fin de PRUEBA.ASM ******
```

3. Escriba un programa que borre la pantalla e imprima un marco rectangular formado por "*" con extremos en 5 5 y 65 20.

   Use la librería CONSOLA.LIB.

4. Incluya una subrutina en el módulo PANTALLA para trazar marcos rectangulares:

SUBRUTINA (lejana)	DESCRIPCION
_marco	Efecto: traza un marco rescangular con extremos en (x1, y1) y (x2,y2)
	Uso: apilar x1, y2, x2, y2 y llamar con call _marco

Para usarla es conveniente definir una macro:

```
marco  macro  x1,y1,x2,y2
mov  ax,x1
push ax
mov  ax,y1
push ax
mov  ax,x2
push ax
mov  ax,y2
push ax
call  _marco
endm
```

Indicación

1) Imprima caracteres con ah=2 Int 21h , dl=código, usando la siguiente tabla

código ASCII (decimal)	
196	raya horizontal
179	raya vertical
218	esquina superior izquierda
191	esquina superior derecha
192	esquina inferior izquierda
217	esquina inferior derecha

2) Desapile los parámetros al retornar de la subrutina, por ejemplo con: RET 8

ENLACE CON LENGUAJES DE ALTO NIVEL

8.1. ENLACE DE SUBRUTINAS CON LENGUAJES DE ALTO NIVEL

A continuación se desarrollan algunos ejemplos que demuestran cómo se enlazan subrutinas en lenguaje ensamblador con programas escritos en lenguajes de alto nivel como BASIC, C, Pascal y dBASE III Plus.

A diferencia de los programas escritos en lenguajes de alto nivel, en donde una sola instrucción por lo general es traducida a varias instrucciones de máquina, en los programas en lenguaje ensamblador cada instrucción es traducida a una sola instrucción de máquina. Por esto, los últimos son más breves en código y se ejecutan en tiempos comparativamente más cortos que los primeros.

Algunas instrucciones en un lenguaje de alto nivel equivalen a muchas de máquina y por lo tanto su codificación en lenguaje ensamblador demandará un buen tiempo del programador. Como regla se recomienda escribir los programas con un

lenguaje de alto nivel y sólo cuando hayan secciones del mismo que requieran ser ejecutadas de una manera más eficiente, y sean fáciles de escribir, intentar reemplazarlas por subrutinas en lenguaje ensamblador.

Ciertos lenguajes de alto nivel ofrecen la capacidad de llamar a subrutinas en lenguaje ensamblador y describen el procedimiento para hacer las llamadas: cómo pasar parámetros, el tipo de llamada (cercana o lejana), los nombres de los segmentos de trabajo, los registros que deben salvarse en caso de ser usados en la subrutina, si la subrutina debe desapilar los parámetros al retornar o si esta tarea será realizada más bien por el propio compilador luego del retorno.

A) COMPILADOR DE BASIC (IBM o Microsoft)

El siguiente programa escrito en BASIC ordena en forma ascendente un arreglo de enteros A%(1), ..., A%(500), que contiene los valores 500, 499, ..., 1, respectivamente, e imprime los tiempos antes y después del proceso de ordenación.

PASO 1. Edite el siguiente programa con un editor de texto y sálvelo con el nombre EJ-BAS.BAS

```
100  CLS
110  DIM A%(500)
120  FOR I%=1 TO 500: A%(I%)=501-I%:NEXT I%
130  PRINT "COMIENZO : " +TIME$
150  CALL ORDENAR(A%(1),500)
160  PRINT "FIN DE PROCESO : " +TIME$
170  PRINT "VALOR DE A%(1)=";A%(1)
```

Nota: En este caso ORDENAR es el nombre del procedimiento lejano y público (FAR, PUBLIC) que se define en el programa en lenguaje ensamblador del paso 3.

PASO 2. Compile EJ-BAS.BAS con el programa BASCOM (compilador de BASIC) :

BASCOM EJ-BAS; <enter>

y obtenga EJ-BAS.OBJ

PASO 3. Escriba y salve el siguiente programa texto con el nombre ORD-BAS.ASM.

```
PROGRAMA    SEGMENT   'CODE'
            ASSUME CS: PROGRAMA
            PUBLIC ORDENAR      ;  ORDENAR es un símbolo público
ORDENAR     PROC FAR            ;  y procedimiento lejano
            PUSH BP
            MOV  BP,SP
            ; obtener parámetros de la pila
            ; SS:BP+8 = contiene dirección (desplazamiento) de primer
            ;            parámetro  (se tomó  A%(1))

            ; SS:BP+6 = contiene dirección desplazamiento) de número
            ;           de datos a ordenar  (se tomó 500)
            MOV  SI,[BP+8]      ; dirección (desplazamiento) de dato
                                ;                  inicial de arreglo
            MOV  BX,[BP+6]      ; dirección (desplazamiento) de número
                                ;                           de datos
            MOV  CX, [BX]       ; número en CX
            DEC  CX             ; establecer límite de comparación
            ADD  CX,CX          ;

MINIMO:
            MOV  AX, [SI]       ; valor de mínimo temporal
            MOV  DX,0           ; índice (posición) de mínimo temporal
            MOV  BX,0           ; índice de valor en prueba

SIGUIENTE:  CMP  BX,CX          ; prueba si se trató último dato
            JE   LISTO          ; sí: ir  a LISTO
            ADD  BX,2           ; nó: acceder al siguiente
            CMP  AX,[SI+BX]     ; es mínimo temporal <= siguiente dato ?
            JLE  SIGUIENTE      ; sí: pasar al siguiente
            MOV  DX,BX          ; no: actualizar índice y
            MOV  AX, [SI+BX]            valor de mínimo temporal
            JMP  SIGUIENTE      ; pasar al siguiente
```

```
LISTO:      XCHG BX,DX          ; intercambiar valores [SI] con
            XCHG AX,[SI]        ; [SI+BX]=mínimo
            XCHG AX,[SI+BX]
            ADD  SI,2           ; tratar resto de arreglo
            DEC  CX             ; reducir    límite en 2: 1 aquí,
            LOOP MINIMO         ; y otro por medio de LOOP

            POP  BP
            RET  4
ORDENAR     ENDP
PROGRAMA    ENDS
            END
```

PASO 4. Enlace EJ-BAS.OBJ y ORD-BAS.OBJ utilizando la librería del compilador de BASIC y obtenga el programa ejecutable EJ-BAS.EXE:

Cuando el programa LINK solicite los módulos objetos (object modules :) responda con: EJ-BAS + ORD-BAS <enter>

B) LENGUAJE C (de Microsoft o Lattice)

El siguiente programa en Lenguaje C llama a las subrutinas cls y cursor escritas en lenguaje ensamblador:

```
            /*  Programa  PRG.C   */

main()
{ cls();          /* borra pantalla */
  cursor(10,20);    /* ubica cursor en 10,20 */
  printf("FIN DE PROGRAMA\n");
}
```

En estas versiones de C (modelo SMALL) las llamadas a las subrutinas son cercanas (NEAR).

La primera subrutina cls no tiene parámetros. La segunda pasa dos parámetros por valor, esto es, en la pila se encontrarán los valores pasados 10 y 20. En C se apilan los parámetros empezando por el último. Así, en este caso primero se apila 20 y luego 10.

También se pueden pasar parámetros por referencia, es decir los valores pasados son direcciones (por ejemplo cuando se emplean arreglos y apuntadores) en donde se localizan los datos.

El programa en lenguaje ensamblador, al que llamaremos VIDEO.ASM, es el siguiente:

```
_TEXT   segment byte public 'CODE'    ; nombre _TEXT y clase CODE
        assume  cs:_TEXT              ; requeridos por el compilador
        public  cls,   cursor         ; símbolos públicos
cls     proc near                     ; subrutina cercana _cls
        push bp
        mov  ah,6
        mov  al,0
        mov  cx,0
        mov  dx,184Fh
        mov  bh,7
        int  10h
        pop  bp
        ret
cls     endp

cursor proc near                      ; subrutina cercana cursor
        push bp                       ; salva bp para acceder a la pila
        mov  bp,sp
        mov  dh, [bp+4]               ; número de fila = parámetro 1
        mov  dl, [bp+6]               ; número de columna = parámetro 2
        mov  bh,0
        mov  ah,2
        int  10h
        pop  bp                       ;    desapila valor inicial de bp
        ret  4                        ; retorna y desapila dos parámetros
cursor endp

_TEXT   ends
        end             ; ****  fin de VIDEO.ASM  ****
```

Procedimiento para obtener la versión ejecutable PRG.EXE

1) Se compila PRG.C usando LC o MC para obtener PRG.OBJ
2) Se compila VIDEO.ASM con MASM y se obtiene VIDEO.OBJ
3) Se produce PRG.EXE con LINK respondiendo con:

   CS + PRG + VIDEO <enter>

   a la pregunta de módulos de objetos (Object Modules),

   y con:
   PRG <enter>
   para el programa ejecutable.

C) LENGUAJE TURBO C

El ejemplo desarrollado en B) puede ser modificado para obtener una versión ejecutable con el compilador TCC:

1 En VIDEO.ASM cambiar cls por _cls y cursor por _cursor
2 Al usar LINK emplee el módulo COS en lugar de CS
3 Usar ret en lugar de ret 4, pues Turbo C se encarga de reajustar la pila desapilando los parámetros.

D) LENGUAJE FORTRAN (Microsoft)

El siguiente programa en FORTRAN, PRG.FOR, es similar al anterior:

```
CALL  cls
CALL  cursor(10,20)
WRITE (*,*) 'FIN DE PROGRAMA'
END
```

Esta versión de FORTRAN hace llamadas a subrutinas lejanas y pasa las direcciones completas (4 bytes) de los parámetros. Estos son apilados comenzando por el de la izquierda.

El programa VIDEO.ASM en lenguaje ensamblador es:

```
_TEXT  segment
assume cs: _TEXT
       public cls, cursor
cls    proc far                ; subrutina lejana cls
     pupush bp
       mov  ah,6
       mov  al,0
       mov  cx,0
       mov  dx,184Fh
       mov  bh,7
       int  10h
       pop  bp
       ret
cls    endp

cursor proc far                ; subrutina lejana cursor

       push bp
       mov  bp,sp
       mov  si,[ bp+10]        ; dirección de parámetro 1
       mov  dh,[ si]
       mov  si,[ bp+6]         ; dirección de parámetro 2
       mov  dl,[ si]
       mov  bh,0
       mov  ah,2
       int  10h
       pop  bp
       ret  8                  ; retorna y desapila los dos
                               ; parámetros (8 bytes)
cursor endp

_TEXT  ends
       end
```

Procedimiento

Para obtener la versión ejecutable PRG.EXE :

1) Compile PRG.FOR con el compilador de FORTRAN (Microsoft) y obtenga PRG.OBJ

2) Compile VIDEO.ASM con MASM y obtenga VIDEO.OBJ

3) Enlace los dos módulos objetos con la librería de FORTRAN: LINK PRG + VIDEO,PRG, ...

E) LENGUAJE PASCAL (Microsoft y Turbo)

Los dos programas siguientes en Pascal utilizan una función SUMA(X,Y) con argumentos y valor de tipo INTEGER escrita en lenguaje ensamblador. La función devuelve la suma de los valores representados por X e Y.

(I) Versión Pascal de Microsoft

```
{ programa PRUEBAMS.PAS }

program pruebams(input,output);
var x,y :integer;

function suma(a,b:integer):integer;
external ;                { para subrutina suma en lenguaje ensamblador}

begin
     write('Ingrese dos enteros : ');
     readln(x,y);
     writeln('La suma es = ',suma (x,y))
end.
```

```
; Programa SUMAMS.ASM

CSEG      segment byte public 'CODE'
          assume  cs:CSEG
          public suma

  suma    proc far
          push bp
          mov  bp,sp
          mov  ax,[ bp+8]       ; par1 = X
          add  ax,[ bp+6]       ; suma par2 = Y al acumulador
          pop  bp
          ret  4                ; retorna y desapila 4 bytes
                                ; AX tiene resultado de la función

  suma    endp

CSEG      ends
          end
```

Procedimiento para obtener PRUEBAMS.EXE

1) Usando el compilador de Pascal Microsoft con el progra PRUEBAMS.PAS se obtiene PRUEBAMS.OBJ

2) Se obtiene SUMAMS.OBJ con: MASM SUMAMS;

3) Y se enlazan los dos módulos objetos obtenidos con la librería de Pascal:

   LINK PRUEBAMS SUMAMS,PRUEBAMS,, ...

   para generar PRUEBAMS.EXE

(II) Versión Turbo Pascal

Se edita el programa PRUEBATP.PAS con el sistema Turbo Pascal:

```
  { programa PRUEBATP.PAS }

program pruebatp;
var x,y :integer;
function suma(a,b:integer):integer; external 'sumat.com' ;

begin
     clrscr;
     write('Ingrese dos enteros : ');
     readln(x,y);
     writeln ('La suma es = ',suma(x,y))
end.
```

Se edita y compila el programa SUMATP.ASM para obtener el programa SUMATP.COM:

```
      ; SUMATP.ASM

public suma
alfa    segment
        assume cs:alfa
  suma  proc near
        push bp
        mov  bp,sp
        mov  ax,[bp+6]
        add  ax,[bp+4]
        pop  bp
        ret 6
  suma  endp
alfa    ends
        end
```

Nota Al hacer llamadas a subrutinas externas el compilador de Turbo Pascal primero apila un parámetro que representa el resultado de la función (function result). Así, en el presente caso se deben desapilar 6 bytes al momento de retornar.

Si el valor de la función es de tipo integer el resultado de la función se entrega en el registro AX, conforme se hace en el programa precedente.

F) dBASE III plus

dBASE III plus permite efectuar llamadas a subrutinas escritas en lenguaje ensamblador. Tales programas, también denominados módulos, son archivos de tipo binario que deben ser almacenados en memoria con la orden:

LOAD nombre de módulo

y pueden ser invocados con:

CALL nombre de archivo WITH expresión

que ejecuta dicho módulo (presente en memoria) pasando como dato (o parámetro) el valor de la expresión

A continuación se desarrolla una subrutina o módulo DIASEM.BIN que convierte un carácter de dígito "1",...,"7", que representa un número de día de semana, en el correspondiente nombre "DOMINGO",...,"SABADO".

Por ejemplo:

```
LOAD DIASEM.BIN
D="5"
CALL DIASEM WITH D
```

y D recibirá el valor "JUEVES":

```
? D
JUEVES
```

Para liberar el módulo DIASEM.BIN de la memoria:

```
RELEASE MODULE DIASEM.BIN
```

El programa DIASEM.ASM es el siguiente:

```
            ;  PROGRAMA DIASEM.ASM

PROGRAMA SEGMENT
         ASSUME CS:PROGRAMA
NDIA     PROC FAR
         MOV AL,[BX]            ; carácter de dígito está en DS:BX
         SUB AL,"1"             ; convertir a número entre 0 y 6
         MOV CX,10              ; diez en CX
         MUL CL                 ; AX =0,10,20,30,40,50,60: entrada en TABLA
         PUSH BX                ; salva BX en la pila
         SUB BX,BX
         MOV BX,AX              ; índice en BX
         LEA SI,TABLA[BX]       ; ubica nombre de día
         POP BX                 ; restaura BX

LAZO:                           ; CX=10, número de caracteres a copiar
         MOV AL,DS:[SI]         ; mueve carácter a registro AL
         MOV  BX [AL]           ; y lo pone en DS:BX
         INC SI                 ; siguiente ...
         INC BX
         LOOP LAZO              ; repite mientras CX > 0
         RET                    ; retorno
NDIA     ENDP

TABLA    db  "DOMINGO  ",0      ; cada nombre ocupa 10 bytes
         db  "LUNES    ",0      ;
         db  "MARTES   " 0
         db  "MIERCOLES",0
         db  "JUEVES   ",0
         db  "VIERNES  ",0
         db  "SABADO   ",0

PROGRAMA ENDS
         END          ;  fin de programa  DIASEM.ASM
```

Proceso para obtener DIASEM.BIN

1) MASM DIASEM; [se produce DIASEM.OBJ]

2) LINK DIASEM; [se produce DIASEM.EXE]

3) EXE2BIN DIASEM [se produce DIASEM.BIN]

8.2. DEPURACION DE SUBRUTINAS DE ENLACE

Frecuentemente una subrutina en lenguaje ensamblador enlazada con un programa escrito en lenguaje de alto nivel no proporciona los resultados deseados, por ejemplo un parámetro retorna con un valor distinto del esperado o simplemente la subrutina no retorna al lugar de llamada. En estos casos es preciso "depurar" la subrutina, es decir determinar los errores y corregirlos.

Para depurar una subrutina, se suele correr el programa de manera que acceda a la subrutina y allí ejecutar las instrucciones de ésta paso a paso, examinando los valores de los registros, los parámetros pasados a través de la pila, etc.

Esta tarea puede ser realizada con el programa depurador SYMDEB.EXE que ofrece adecuados procedimientos con dicho propósito, los cuales se exponen en la correspondiente guía de uso.

También empleando SYMDEB es posible aplicar un método muy simple para depurar una subrutina, que consiste en fijar en ella puntos de parada o de detención con la interrupción 3, a partir de los cuales, luego de ser eliminados o sustituídos por la instrucción NOP, que no hace nada y ocupa el mismo espacio (un octeto), se pueden ejecutar las siguientes instrucciones paso a paso con los comandos T o P de SYMDEB. Este procedimiento se describe en forma detallada así:

(1) En la subrutina S se incluye la interrupción 3:
INT 3 que detiene el programa.

(2) Proceso de depuración con SYMDEB:

```
SYMDEB<enter>        [correr SYMDEB]

-N NP<enter>         [nombrar programa ejecutable; NP es
                      el nombre completo del programa
                      (incluye extensión .EXE o  .COM) ]

-L<enter>            [cargarlo]

-G<enter>            [ejecutar NP bajo SYMDEB]
```

Entonces el programa se detiene en la primera ocurrencia de INT 3. Se pueden observar los registros, las siguientes instrucciones (con el comando U), los valores de la pila (con DB SS:SP), etc.

Para eliminar INT 3 se reemplaza ésta por NOP:

```
- A <enter>   [o A IP <enter> ;  IP tiene la dirección del punto
                                 de parada dado por INT 3]

  XXXX:YYYY   NOP<enter>         [ensamblar NOP]
  XXXX:UUUU   <enter>
```

Ahora se pueden ejecutar los comandos T o P y examinar el desarrollo de la subrutina.

8.3. EJERCICIOS

1. Escriba un programa en un lenguaje de alto nivel que utilice una subrutina en lenguaje ensamblador para obtener el mayor de dos enteros que se pasan como parámetros.

2. Utilice el método de depuración indicado en la sección precedentes para examinar la ejecución de las instrucciones de la subrutina del ejercicio 1.

3. Escriba un programa en un lenguaje de alto nivel que use una subrutina en lenguaje ensamblador que convierta las letras minúsculas a mayúsculas de una cadena que se pasa como parámetro.

4. Un programa ejecutable puede ser ejecutado ingresando su nombre seguido por un texto o cadena de caracteres:

   NP TEXTO<enter>

   Este texto se halla en la dirección S:81h y el número de caracteres que lo componen se halla en el octeto de la dirección S:80h, en donde S es la dirección del llamado segmento prefijo de programa, cuyo valor es puesto por el sistema operativo DOS en los registros DS ó ES al iniciarse la ejecución del programa (y por tanto pueden ser accedidos antes de cambiar sus valores por otros segmentos).

   COPIAR NF ND<enter>

   haga una copia del archivo fuente NF en el archivo destino ND, siendo NF y ND especificaciones arbitrarias de archivos, es decir incluyen posibles caminos de subdirectorios y nombres.

5. El sistema operativo DOS proporciona una función para obtener la fecha del sistema:

Se ingresa AH=2Ah y se ejecuta con INT 21h, resultando los siguientes valores enteros:

AL= día de semana (0=domingo - 6=sábado)
CX= año (1980 - 2099)
DL= día de mes (1 - 31)
DH= mes (1 - 12)

Escriba una subrutina en lenguaje ensamblador que obtenga estos datos y úsela en un programa en lenguaje de alto nivel para imprimir la fecha del sistema, por ejemplo en la forma:

Domingo, 2 de abril de 1989

A P L I C A C I O N E S

En la presente sección se desarrollan en forma completa varios programas en lenguaje ensamblador. Estos programas, después de compilarlos, pueden ser usados directamente como programas ejecutables. También, luego de efectuar ligeras modificaciones para convertirlos en subrutinas, podrán ser incluídos o enlazados con otros programas en lenguaje ensamblador o en lenguaje de alto nivel, en forma individual o extraídos de librerías que los contengan.

9.1. PROGRAMA: Determina si impresora está preparada

El siguiente programa escrito en lenguaje ensamblador se utiliza para determinar si la impresora está preparada o nó para imprimir.

El programa utiliza una subrutina cercana PRUEBA_IMP que retorna con uno de los valores: uno (1=impresora lista) o cero (0=impresora no lista) en el registro AX.

La subrutina PRUEBA_IMP emplea la interrupción 17h con el registro AH=2 (solicitud de estado), que al ejecutarse devuelve en AH el estado de la impresora, la cual no está lista si al menos uno de los bits 3 y 5 de AH toma el valor 1:

```
bit 3 = valor 1 si hay error de entrada/salida
bit 5 = valor 1 si no hay papel
```

El programa es el siguiente:

```
IMPRIME     MACRO CADENA        ; macro para imprimir cadenas
            MOV AX, SEG CADENA; DS:DX= dirección de CADENA
            MOV DS, AX
            MOV DX, OFFSET CADENA
            MOV AH,9            ; función de impresión de cadena en pantalla
            INT 21h
IMPRIME     ENDM

PILA SEGMENT STACK
     DB 10h DUP(?)
PILA ENDS

DATOS SEGMENT
     MLISTA DB "Impresora preparada!",13,10,"$"
     MERROR DB "Impresora no preparada!",13,10,"$"
DATOS ENDS

CODIGO SEGMENT
            ASSUME CS:CODIGO

COMIENZO:   CALL PRUEBA_IMP     ; llamada a subrutina
            CMP AX,1
            JNE NOLISTA
            IMPRIME MLISTA
            JMP FIN
NOLISTA:    IMPRIME MERROR
FIN:        MOV AH,4Ch          ; fin de proceso
            INT 21h

PRUEBA_IMP  PROC NEAR   ; === subrutina  prueba estado de impresora ===

            MOV AH,2            ; solicitud de estado
            MOV DH,0            ; usar primera impresora
            INT 17h             ; ejecutar: AH regresa con bits de estado
                                ; de impresora
            TEST AH,00101000b ; patrón de prueba de estado en binario:
                                ; bit 3 (1=error de entrada/salida)
                                ; bit 5 (1=error de falta de papel)
                                ; indicador de cero ZF se establece (error)
                                ; si y sólo si al menos uno de éstos bits
                                ; es 1
            JNZ IMPNO           ; si no es cero saltar a impresora no lista
            MOV AX,1            ; impresora lista
            JMP FIN_PRUEBA
IMPNO:      MOV AX,0            ; impresora no lista
FIN_PRUEBA:
            RET                 ; AX retorna con información sobre impresora

PRUEBA_IMP  ENDP                ; === fin de subrutina ===

CODIGO      ENDS
            END COMIENZO
```

9.2. PROGRAMA: Redefine teclado

El siguiente programa permite redefinir las teclas de función F1, F2, ..., F40. El programa ejecutable resultante es FUNCION.EXE. Si por ejemplo se desea redefinir la tecla F5 de modo que cada vez que se presione, desde el sistema operativo, imprima DIR B:

basta ejecutar este programa ingresando: FUNCION 5 DIR B: <enter>

9.2.1 Requisito ANSI.SYS

Para que este programa tenga efecto se requiere que el sistema DOS haya sido instalado con el dispositivo ANSI.SYS (el archivo CONFIG.SYS debe contener DEVICE=ANSI.SYS)

9.2.2 Códigos de teclas de funciones

Los códigos de las teclas de funciones F1 ,..., F10 son dados por los pares de números 0 59,..., 0 68, respectivamente. Las otras teclas de funciones F11,...,F40 se describen así:

TECLA	CODIGO	INGRESO
F11 - F20	0 84 - 0 93	Shift F1 - Shift F10
F21 - F30	0 94 - 0 103	Ctrl F1 - Ctrl F10
F31 - F40	0 104 - 0 113	Alt F1 - Alt F10

9.2.3 Reasignación de tecla

Para asignar el texto DIR B: a la tecla F5, de código 0 63, es preciso que se escriba en el archivo asociado al teclado, con descriptor 1, la cadena de octetos:

```
ESC      0;63;"DIR B:  ";8p
--       ----  -----
|          |     | cadena de reasignación
|          |
|          └──→ código de F5
|
└──→ código de escape = 27
```

que se puede almacenar en una variable TECLA_F5 mediante:

```
TECLA_F5   DB  27, '[0;63;"DIR B: ";8p'
```

en donde el texto DIR B: se encierra entre comillas (el carácter 8 que precede a p se usa como delimitador, pero su efecto es el de borrar el último carácter del texto y por ello a éste se le ha añadido un carácter de blanco)

9.2.4 Acceso a las palabras ingresadas al iniciar la ejecución de un programa

Un programa ejecutable NP (de tipo .EXE o .COM) puede ser ejecutado en la forma:

```
NP  PAR<enter>
```

en donde PAR es una cadena, llamada cadena de parámetros de entrada, que está formada por todos los caracteres que siguen al nombre del programa, cuyas palabras componentes son datos que pueden ser empleados directamente en el programa.

Por ejemplo, al ejecutar el programa FUNCION.EXE en la forma:

```
FUNCION 5 DIR:B <enter>
```

la cadena de parámetros será "5 DIR:B" y el programa deberá acceder a ésta y obtener sus palabras componentes "5" y "DIR:B" para poder usarlas (en este caso será necesario que a partir de "5" se derive el segundo código de la tecla F5, esto es la cadena "63", por ejemplo, mediante la siguiente secuencia:

cadena ingresada		número		código F5 (numérico)		código F5 (cadena)
"5"	⟶	5	⟶	58+5=63	⟶	"63"

Nótese que los códigos de las teclas Fi, i=1, ..., 10, se obtienen sumando 58+i; en cambio para las teclas F11 ,..., F40, éstos resultan de sumar 83+i, i=11 ,..., 40

La cadena de parámetros PAR se localiza a partir de la dirección S:81h, y el número de caracteres que la componen se halla en el octeto de la dirección S:80h, en donde S es el valor que tienen los registro DS ó ES, al momento de iniciarse la ejecución del programa. Estos registros contienen la dirección de un bloque de 100h octetos, construído por el sistema operativo, que se llama bloque o segmento prefijo del programa y está construído por información relativa al programa ejecutable. Las instrucciones del programa son cargadas a continuación de este bloque.

9.2.5. Desarrollo del programa FUNCION.ASM

El programa realiza las siguientes tareas:

(1) Mediante la subrutina descodificar se accede a la cadena de parámetros y se obtienen las dos palabras componentes. La primera es empleada para obtener el código (cadena) y la segunda representa el texto que se reasignará a la tecla. Estos datos se copian en un área, a la que llamamos PATRON, constituída además por los otros caracteres que exige la sucesión de escape para la redefinición de una tecla.

(2) Se imprime la cadena PATRON en el archivo asociado al teclado, cuyo descriptor es igual a 1.

(3) Si se detecta un error (primera cadena no es numérica, no hay texto, o simplemente la cadena de parámetros es vacía) no se hace ninguna reasignación.

El programa es el siguiente:

```
BLANCO      equ 32              ; código de carácter de blanco
TECLALIM    equ 40              ; código de tecla de función límite

pila segment stack
     db 20 dup(?)
pila ends

datos  segment
      ;  área de cadena a imprimir según patrón:
      ;              27, '[0;x;"nnn... ";8p'
      ;          x = (cadena) segundo código de tecla de función
      ;   nnn ...  = (cadena) nuevo valor de la tecla

       PATRON     db  27,'[0;'
       FCOD2      db  ?,?,?          ; segundo código de tecla de función
                  db  ';"'           ; punto y coma y un par de comillas
       NVALORT    db  80 dup(?)      ; nuevo valor de la tecla= segunda
                                     ; palabra de cadena de parámetro
                                     ; y otros caracteres de patrón: '8p'

      ; otras variables
       VALOR_COD  db  ?              ; variable para controlar la subrutina
                                     ; descodificar: hay error si ésta
                                     ; retorna con 0
       ULTIMO     dw  ?              ; posición de último carácter de PATRON
       DIEZ       db  10             ; variables para dividir
       CIEN       db  100

datos ends

codigo segment
       assume cs:codigo, ds:datos

@comienzo:
       mov ax, datos
       mov ds, ax
       mov VALOR_COD,0              ; se inicia     con 0=error

       call  descodificar
       cmp  VALOR_COD, 0
       je   @fin

      ;  imprimir  PATRON en archivo teclado

       mov dx, offset PATRON        ; comienzo de PATRON
       mov cx,ULTIMO                ; fin de PATRON
       sub cx,dx                    ; número de caracteres -1
       inc cx
       mov bx, 1                    ; archivo de teclado
       mov ah, 40h                  ; imprimir cx octetos
       int 21h

@fin:  mov ah, 4Ch                  ; fin de proceso
       int 21h
```

```
descodificar proc near

      mov bx, 80h              ; acceso a la cadena de parámetros
      sub cx, cx
      mov cl, es:[bx]          ; es = segmento prefijo de programa
                               ; cx = número de caracteres
      inc bx                   ; apuntar al primer carácter ...

      ; ignorar blancos que preceden a cadena de número

@blancos1:  cmp cx,0
            jne  @sigue1
            ret                ; retornar no hay caracteres
@sigue1:    cmp  byte ptr es:[bx],BLANCO
            jne  @cad_num
            inc  bx
            dec  cx
            jmp  @blancos1

      ; acceder a cadena numérica y convertirla a número
      ; valor resultante en ax

@cad_num:   mov ax,0           ; inicia valor de número
@otrodig:
            cmp cx,0
            jne @sigue2
            ret                ; cx es cero: retornar, sólo blancos o
                               ; sólo número (no hay segunda palabra)

@sigue2:    mov dl, es:[bx]    ; carácter en dl
            cmp dl, '0'        ; probar si es carácter de dígito
            jb  @sigue3
            cmp dl, '9'
            ja  @sigue3
            sub dl, '0'        ; es carácter de dígito: convertirlo a
                               ; número
            mul DIEZ           ; multiplicar al por diez y agregarle dígito
            add al,dl
            dec cx
            inc bx
            cmp ax, TECLALIM   ; límite de tecla de funciones
            jbe @otrodig
            ret                ; retornar: número ingresado es mayor
                               ; que TECLALIM=40

@sigue3:    cmp ax,0           ; no hay valor de tecla de función
            ja  @sigue4
            ret                ; retornar: no se ingresó cadena numérica

@sigue4:    cmp dl, BLANCO     ; compara si siguiente carácter es blanco
            je @blancos2
            ret                ; retornar: no hay segunda palabra
```

```
        ; ignorar siguientes blancos

@blancos2:  cmp  cx, 0

            jne  @sigue5
            ret                 ; retornar: no hay segunda palabra

 sigue5:    cmp  byte ptr es:[bx],BLANCO
            jne  @palabra2
            inc  bx
            dec  cx
            jmp  @blancos2

@palabra2:
            mov VALOR_COD,al  ; salvar código (numérico)

        ; copiar palabra2 en NVALORT

            mov ax, offset NVALORT
            mov si,ax           ; ds:si=dirección de NVALORT
@otrocar:                       ; es:bx=dirección de palabra2
            mov dl, es:[bx]
            mov [si],dl
            inc bx
            inc si
            loop @otrocar       ; copia cx caracteres

        ; añadir resto de patrón

            mov byte ptr [si],BLANCO
            inc si
            mov byte ptr [si],'"'       ;  un par de comillas
            inc si
            mov byte ptr [si],';'       ;  punto y coma
            inc si
            mov byte ptr [si],'8'       ; ccarácter de retroceso
            inc si
            mov byte ptr [si],'p'       ;  último carácter de patrón


        ; salvar posición de último carácter

            mov ax,si
            mov ULTIMO,ax


        ; falta poner la cadena de código de la tecla en FCOD2
            ; obtener código numérico en ax
            sub  ax,ax
            CMP  VALOR_COD,10
            ja  @grupo2
            mov  al,58          ; grupo1: teclas F1 ... F10
            add  al,VALOR_COD   ;       ax    =  59 ...  68
            jmp  @num_a_cad
```

```
@grupo2:                        ; grupo2: teclas F11 ... F40
          mov al,83
          add al,VALOR_COD      ;       ax    =  84  ... 113

     ; convierte código numérico de función a cadena en FCOD2

@num_a_cad:

          mov bx,offset FCOD2 ; ds:bx=dirección de FCOD2
          div CIEN            ; dígito de centenas en al
                              ; resto en ah
          add al,'0'          ; a carácter de dígito
          mov [bx],al         ; moverlo a ds:bx
          mov  al,ah          ; resto en ax
          sub  ah,ah
          div  DIEZ           ; dígito de decenas en al
                              ; dígito de unidades en ah
          add  al,'0'
          inc  bx
          mov [bx],al         ; carácter de decenas en siguiente pos
          add  ah,'0'
          inc  bx
          mov  [bx],ah        ; carácter de unidades
          ret

descodificar  endp

codigo  ends
        end  @comienzo
```

9.3. PROGRAMA: Determina tamaño de espacio libre en disco

El siguiente programa DLIBRE.EXE imprime el número de K bytes libres en la unidad de disco que se especifique al momento de ejecutar el programa. Por ejemplo:

```
DLIBRE A <enter>
128 345  bytes disponibles
```

imprime el espacio libre en la unidad de discos A.

Para hallar el espacio libre de un disco se ingresa:

```
AH = 36h            ; función que obtiene el espacio disponible
                    ; en disco
DL = NumDisco       ; número de unidad de disco:
                    ; 0 = disco por defecto, 1 = A,  2 = B, ...
```

y se ejecuta con INT 21h, dando como resultado los siguientes valores:

BX = número disponible de unidades de alojamiento (U.A.) o "clusters"
AX = número de sectores por U.A.
CX = número de bytes por sector

NOTA: En caso de error AX toma el valor 0FFFFh

Por tanto, espacio libre en bytes es el producto EspLib = AX * BX * CX.

A fin de imprimir N por medio de sus dígitos decimales recurriendo a la multiplicación de 16 bits que provee el procesador, aplicamos el siguiente procedimiento:

(a) Se calcula el número de millares y se imprime
(b) Se calcula el resto, que es un número menor que 1000, y se imprime a continuación del anterior.

Teniendo en cuenta las siguientes relaciones:

(1) `AX * BX = Q1 * MIL + R1` ; Q1 = cociente entero y R1 = resto de la ; división de AX * BX entre MIL

(2) `R1 * CX = Q2 MIL + R2` ; Q2 = cociente entero y R2 = resto de la ; división de R1 * CX entre MIL

```
N = (AX * BX) * CX  =  (Q1 * MIL + R1) * CX
                    =  Q1 * MIL * CX + R1 * CX
                    =  Q1 * CX * MIL + Q2 * MIL + R2
                    =  (Q1 * CX + Q2) * MIL + R2
```

se deduce:

(3) millares de EspLib = Q1 * CX + Q2

(4) resto de Esplib = R2

en donde Q1, R1, Q2 y R2 se determinan según (1) y (2)

El programa se desarrolla a continuación.

```
; ****    PROGRAMA  DLIBRE.ASM    ****

BLANCO EQU 32                   ; carácter de blanco

PILA SEGMENT STACK
     DB 30 DUP (?)
PILA ENDS


DATOS SEGMENT

     NumDisco  db  ?            ; variable para número de disco
                                ; variables para resultado de espacio libre

     NumMil    dw  ?            ; número de millares de EspLib
     NumResto  dw  ?            ; resto de EspLib igual a R2

                                ; variables de cálculos auxiliares

     Q1        dw  ?
     R1        dw  ?
     Q2        dw  ?
     MIL       dw  1000
     DIEZ      dw  10

     CADENA    db  "  bytes disponibles ",13,13,"$"

DATOS ENDS

CODIGO SEGMENT
      ASSUME CS: CODIGO, DS:DATOS
@Comienzo:
         MOV AX, DATOS
         MOV DS, AX
         CALL  HALLA_DISCO      ; NumDisco retorna con número de disco
         JC    @Fin             ; hay acarreo : error
         CALL  HALLA_ESPACIO    ; NumLibre = número de K bytes
         JC    @Fin             ; hay acarreo : error
         CMP   NumMil, 0        ; comprobar si NumMil es cero
         JE    @Resto           ; sí:  no imprimirlo y pasar a @Resto
         MOV   AX, NumMil       ; dato se pasa en AX
         CALL  IMPRIME_NUMERO
         MOV   AH,2             ; imprimir coma para separar millares
         MOV   DL, ","
         INT   21h

         CMP   NumResto, 100    ; imprimir ceros si hay millares y
         JAE   @Resto           ; NumResto tienen menos de 3 dígitos
         MOV   DL,"0"           ; cero centenas
         INT   21 h
         CMP   NumResto, 10     ; cero decenas
         JAE   @Resto
         INT   21h
```

```
@Resto:   MOV   AX, NumResto     ; imprimir NumResto
          CALL  IMPRIME_NUMERO
          MOV   DX, OFFSET CADENA ; imprimir CADENA
          MOV   AH, 9
          INT   21h
@Fin:
          MOV   AH, 4Ch
          INT   21h


HALLA_DISCO PROC NEAR
          SUB   CX, CX
          MOV   SI, 80h          ; dirección de área de parámetros
          MOV   CL, ES:[SI]      ; número de caracteres leídos
          INC   SI

@Blanco_Ini:                     ; eliminar blancos iniciales
          CMP CX, 0
          JA   @HayCar
          MOV  AL, 0             ; AL=0=disco actual (por defecto)
          JMP  @Disco_Listo
    @HayCar:
          CMP  byte ptr ES:[SI], BLANCO
          JNE  @Blanco_Fin
          DEC  CX
          INC  SI
          JMP  @Blanco_Ini

@Blanco_Fin:                     ; eliminar blancos finales
          MOV  BX, CX
          DEC  BX
    @OtroBlanco:
          MOV  AL, ES:[SI+BX]
          CMP  AL, BLANCO
          JNE  @Cadena
          DEC  BX
          DEC  CX
          JMP  @OtroBlanco

@Cadena:                         ; cadena debe tener sólo un carácter:
          CMP  CX, 1             ; letra de disco  en ES:SI
          JE   @Cad1
          JMP  @Disco_Error
  @Cad1:
          MOV AL, ES:[SI]
          CMP AL, "a"
          JB  @Cad2
          SUB AL, 32             ; conversión a mayúscula
  @Cad2:
          SUB AL, "A"            ; reducción a 1 (A), 2 (B) , ...
          INC AL
          CMP AL, 0
          JA  @Disco_Listo
          JMP @Disco_Error
```

```
@Disco_Listo:
            MOV NumDisco, AL
            CLC                 ; anula acarreo:  correcto
            RET

@Disco_Error:                   ; establece acarreo: error
            STC
            RET

HALLA_DISCO ENDP


HALLA_ESPACIO PROC NEAR

            MOV AH, 36h
            MOV DL, NumDisco
            INT 21h
            CMP AX, 0FFFFh      ; probar si hay error de disco (no existe)
            JNE @Espacio
            STC                 ; establecer acarreo: error
            RET

  @Espacio:
            MUL BX              ; AX*BX  en par DX AX
                                ; AX*BX = Q1 * MIL + R1
                                ; hallar  Q1 y R1
            DIV MIL             ; divide DX AX entre MIL
                                ; AX=cociente,  DX=resto
            MOV Q1, AX          ; salvar cociente en Q1
            MOV R1, DX          ; salvar resto en R1

                                ; Hallar R1 * CX
            MOV AX, R1
            MUL CX
                                ; R1 * CX = Q2 * MIL + R2
                                ; hallar Q2=AX  y  R2=DX
            DIV MIL
            MOV Q2, AX

            MOV NumResto, DX    ; salvar resto de millares
            MOV AX, Q1          ; Hallar  Q1*CX+Q2  en  DX AX
            MUL CX
            ADD AX, Q2
            MOV NumMil, AX      ; salvar número de millares
            CLC                 ; anular acarreo: correcto
            RET

HALLA_ESPACIO ENDP
```

```
IMPRIME_NUMERO PROC NEAR          ; AX viene con valor a imprimir
           SUB CX, CX             ; contador de dígitos

 @Digito:
           MOV DX, 0
           DIV DIEZ
           ADD DX, "0"
           PUSH DX
           INC  CX
           CMP  AX, 0
           JE   @Imprimir_Car
           JMP  @Digito

@Imprimir_Car:

           MOV AH, 2              ; función imprime carácter

@Siguiente_Car:

           POP DX
           INT 21h
           LOOP @Siguiente_Car ; repetir CX veces
           RET

IMPRIME_NUMERO ENDP


CODIGO ENDS
      END  @Comienzo

      ;  ****** Fin de programa  DLIBRE.ASM  ******
```

9.4. PROGRAMA: Programas residentes

El sistema operativo DOS permite dejar un programa residente en memoria de manera que puede ser llamado posteriormente por cualquier otro programa que se ejecute sin que sea necesario volver a leerlo del disco. El programa residente permanece en memoria hasta que se vuelva a iniciar el sistema.

9.4.1 Llamadas a programas residentes

Para que otro programa pueda llamar a un programa residente se debe conocer la dirección absoluta (4 bytes) en donde éste se halla actualmente en memoria. Dicha dirección se salva antes de dejar residente al programa en otra dirección previamente elegida por el usuario. De esta manera, para llamar al programa residente basta acceder a la dirección fijada de antemano y efectuar una llamada lejana e indirecta según el contenido de ella.

DOS ofrece al usuario un área de 16 bytes, llamada área de comunicación entre las aplicaciones, que se localiza en la región 0:4F0h - 0:4FFh, en donde se puede salvar la dirección absoluta (4 bytes) del programa a residir.

Ejemplo

Si se asume que en las direcciones 0:4F0h y 0:4FFh se encuentra la dirección (desplazamiento y segmento, respectivamente) del programa residente, por ejemplo, el programa externo podrá llamarlo con las instrucciones:

```
;  salvar DS y hacer DS:BX=0:4F0h
;  de modo que DS:[BX]=desplazamiento de programa residente
;              y DS:[BX+2]= segmento de programa residente

   PUSH DS
   MOV AX, 0
   MOV DS, AX
   MOV BX, 4F0h

;  llamada lejana indirecta a programa residente
   CALL DWORD PTR [BX]  ;  Nota: por defecto [BX] = DS:[BX]

;  restaurar DS
   POP DS   ;
```

o también empleando otros registros o variables de memoria.

Nota

Más adelante se tratan las subrutinas de interrupción definidas por el usuario. Estas también pueden ser instaladas como programas residentes: Basta reemplazar la dirección de la subrutina antigua por la nueva. En estos casos las llamadas a estas subrutinas residentes se realizan cada vez que se genere una interrupción del tipo correspondiente.

9.4.2 Instalación de un programa residente

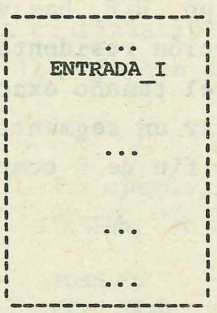
Si se desea que un programa R resida en memoria se debe preparar un programa instalador o cargador, al que denominaremos IR, que contenga como parte inicial a R y una segunda parte, a la que llamaremos I, compuesta por instrucciones cuyo propósito es salvar la dirección de entrada al programa R y establecer la porción que debe quedar residente. Al ejecutarse el programa IR todo su contenido es cargado en memoria, sólo las instrucciones de I se ejecutan y cuando IR termina se libera el área que corresponde a esta última parte.

Si la parte I es bastante pequeña, en muchos casos suele dejarse residente la totalidad del programa.

La estructura del programa IR es la siguiente:

```
                                ;  *** programa residente R ***
          ...
SubrER  PROC FAR                ; ← subrutina de entrada a programa resi-
          ...                   ;    dente  (en un segmento de código
                                ;    para hacer llamadas lejanas  desde
          RET                   ;    otros programas
          ...                   ;
SubrER ENDP
          ...

                                ; *** lugar para marcar fin de residente ***
```




```
                ;  *** programa I ***
                ;  <-  subrutina de entrada a programa IR
                ;          (en un segmento de código).
                ;
                ;          *** TAREAS PRINCIPALES ***
                ;
                ;      1)  Salvar dirección de SubrER en lugar
                ;          seleccionado
                ;      2)  Establecer tamaño en registro DX
                ;      3)  Terminar dejando residente a R
                ;
END  XXXX       ;      punto de entrada a IR en etiqueta XXXX
```

Para establecer el tamaño y dejar residente el programa R se dispone de dos métodos:

(1) Se ingresa:

AH = 31h
DX = tamaño de parágrafos (1 parágrafo = 16 bytes)

y se ejecuta con INT 21h

Con este método -el más recomendado- se pueden instalar programas residentes de tamaño mayor que 64K bytes.

(2) Se ingresa:

DX = tamaño en bytes

y se ejecuta con INT 27h

Este método tiene varias restricciones, por ejemplo, el tamaño no puede exceder a 64K bytes.

Debe tenerse en cuenta que la porción residente comprende al segmento prefijo de programa (256 bytes = 100h) además de las instrucciones del programa R.

Si el programa ejecutable IR será de tipo .COM, el tamaño exacto en bytes puede ser obtenido así:

a) se establece una etiqueta al final de R :

```
FIN_RES LABEL WORD      ; marcar fin de residente
```

y b) el tamaño en bytes es dado por OFFSET FIN_RES.

Si el programa ejecutable será de tipo .EXE, hay varias maneras de determinar el tamaño de la porción residente. En este caso un método simple para calcular el tamaño exacto en parágrafos consiste simplemente en insertar un segmento de programa vacío que sirva como marcador de fin de R como se muestra en la siguiente figura:

```
; *** PARTE 1 : Programa R a residir

DATOS_R    SEGMENT          ; opcional : área de datos de R
           ...
DATOS_R    ENDS

CODIGO_R   SEGMENT
           ASSUME CS: CODIGO_R , ...

  SubrER     PROC FAR       ; ← subrutina de entrada a programa R
             ...
             RET
  SubrER     ENDP

             ...

CODIGO_R   ENDS             ;  *** FIN de R ***

; *** PARTE 2 : Segmento vacío usado como marcador de programa residente
FIN_RES  SEGMENT
FIN_RES  ENDS

; *** PARTE 3 : Instrucciones de instalación
DATOS_I    SEGMENT          ; opcional: área de datos para instalación
           ...
DATOS_I    ENDS

CODIGO_I   SEGMENT          ; código de instalación
           ASSUME CS: CODIGO_I, ...

COMIENZO:
           ...
CODIGO_I   ENDS

PILA_I   SEGMENT STACK      ; segmento de pila usado solamente
         DB N DUP(?)        ;      durante el proceso de instalación
PILA_I   ENDS
         END  COMIENZO
```

A continuación indicamos cómo se pueden escribir las tareas 1) - 3) que deben realizarse en la parte de instalación I:

1) Por ejemplo, para salvar la dirección de SubrER en la dirección 0:4F0h, se tienen las instrucciones:

```
PUSH ES          ; salvar ES
MOV  AX,0        ; hacer  ES:BX = 0:4F0h
MOV  ES,AX
MOV  BX,4F0h

                 ; mover dirección de SubrER a 0:4F0h

MOV WORD PTR ES:[BX], OFFSET SubrER
MOV WORD PTR ES:[BX+2], SEG  SubrER ; *** NOTA: Si programa es de
                      tipo .COM  usar aquí:  MOV ES:[BX+2], CS

POP  ES          ; restaurar ES
```

2) Poner tamaño en DX:

Para programas de tipo .COM usando la etiqueta FIN_RES

```
MOV DX, OFFSET FIN_RES  ;  tamano en bytes
```

y si se desea en parágrafos hay que dividir este valor entre 16, lo que equivale a desplazarlo 4 bits a la derecha, se puede añadir:

```
MOV  CL,4
SHR  DX,CL       ; desplazar CL = 4 bits a la derecha
INC  DX          ; aumentar 1 parágrafo pues resto puede ser
                 ; mayor que 0
```

Otra forma consiste en efectuar directamente la división entre 16.

Nota: En muchos casos, cuando todo el programa IR permanezca residente, estos valores pueden ser ingresados manualmente por el programador, para lo cual se escribe:

```
MOV DX, 0        ;  0 es un valor temporal
```

y luego se compila el programa hasta obtener la versión ejecutable cuyo tamaño en bytes puede ser observado con la orden DIR del DOS. (Si el programa es de tipo .EXE hay que agregarle 256). Finalmente, se retorna al programa fuente IR.ASM a fin de reemplazar el valor temporal 0 por el tamaño en bytes o en parágrafos según sea el caso.

3) Finalizar y dejar residente ejecutando INT 27h (tamaño en bytes) o bien con INT 21h y AH = 31h (tamaño en parágrafos).

Advertencia

Una vez iniciado el sistema, el programa instalador debe ser ejecutado sólo una vez, pues de volver a hacerlo se añadirán nuevas copias del programa residente reduciéndose por consiguiente la memoria disponible para las aplicaciones.

El siguiente programa IR1.ASM instala como programa residente a la sección inicial delimitada por la etiqueta FIN_RES. La tarea del programa residente es muy simple: imprime el mensaje "Programa residente!" cada vez que se realice una llamada desde otro programa. La dirección de llamada es dada por la subrutina lejana ENTR_RES.

El programa instalador será de tipo COM y por lo tanto el segmento de ENTR_RES será dado por el registro CS. Después de IR1.ASM se desarrolla un programa para efectuar llamadas al programa residente.

```
;  ***  programa IR1.ASM
;       genera programa instalador residente IR1.COM

                              ; ***  parte a residir  ***
CODIGO SEGMENT                ; área de datos
       ASSUME CS: CODIGO
       ORG 100h               ; establecer origen en desplazamiento 100h
                              ; para programas de tipo  .COM
COMIENZO:                     ; entrada a programa IR1.COM
       JMP INSTALAR
ENTR_RES  PROC FAC            ; ← entrada a subrutina residente
       PUSH DS                ; salva registro DS
       MOV  AX, CS            ; hace DS:DX = dirección de MENSAJE
       MOV  DS, AX
       MOV  DX, OFFSET MENSAJE
       MOV  AH, 9             ; imprime
       INT  21h
       POP  DS                ; restaura DS
       RET                    ; y retorno lejano
ENTR_RES  ENDP
MENSAJE  db "Programa residente!",13,10,"$"
FIN_RES  LABEL WORD           ; *** etiqueta : marca fin de residente ***
```

```
INSTALAR PROC NEAR

      PUSH DS                 ; salva DS
      MOV AX, 0               ; hace DS:BX=0:4F0h
      MOV DS, AX
      MOV BX, 4F0h
                              ; salva dirección de ENTR_RES
      MOV WORD PTR [BX], OFFSET ENTR_RES;   0:[4F0]= desplazamiento
      MOV [BX+2], CS          ;           0:[4F2]= segmento de ENTR_RES
      POP DS                  ; restaura DS

      MOV DX, OFFSET FIN_RES   ;  calcula tamaño de parágrafos
      MOV CL, 4
      SHR DX
      INC DX
      MOV AH, 31H             ; función para terminar dejando residente
      INT 21h                 ; bloque inicial

INSTALAR ENDP

CODIGO  ENDS
        END  COMIENZO         ; fin de IR1.ASM
```

Este programa puede ser compilado hasta obtener la versión ejecutable IR1.COM:

```
MASM IR1; <enter>
LINK IR1; <enter>
EXE2BIN IR1.EXE IR1.COM <enter>
```

Al correr IR1.COM quedará residente toda la parte inicial hasta el lugar indicado por la etiqueta FIN_RES.

Ahora mostramos un programa que al ejecutarse llame al programa residente que se acaba de construir.

```
; **** programa LLAM-RES.ASM ****
; llama programa residente cuya dirección se halla en 0:4F0h (4 bytes)

CODIGO SEGMENT
       ASSUME CS:CODIGO
COMIENZO:

       MOV AX, 0                ; Hacer DS:BX = 0:4F0h = dirección
       MOV DS, AX               ; donde se halla la dirección (4 bytes:
       MOV BX,4F0h              ; desplazamiento y segmento) de la sub-
                                ; rutina residente

       CALL DWORD PTR [BX]      ; Llamada lejana a subrutina residente
                                ; [BX]=DS:[BX] = dirección (palabra doble)

       MOV AH,4Ch               ; fin de proceso
       INT 21h
CODIGO ENDS
       END  COMIENZO            ; fin de programa LLAM-RES


PILA SEGMENT STACK              ; pila de LLAM-RES
       DB 80 DUP(?)             ; tamaño adecuado para realizar llamadas
PILA ENDS                       ; (incluyendo las que se hacen dentro del
                                ; programa residente)
```

Compile este programa hasta obtener el programa ejecutable LLAM-RES.EXE. Entonces cada vez que se ejecute LLAM-RES se imprimirá el mensaje:

Programa residente!

Por supuesto esto será realizado sólo si el programa residente ha sido instalado antes.

Finalizamos esta sección presentando una versión de tipo .EXE de un programa instalador similar a IR1.ASM. El programa fuente será denominado IR2.ASM y deberá ser compilado hasta obtener IR2.EXE.

```
; *** programa IR2.ASM ***
     genere programa instalador de residente IR2.EXE


                          ; ***  parte a residir  ***

DATOSR SEGMENT            ; área de datos
   MENSAJE  db "Programa residente!",13,10,"$"
DATOSR ENDS

CODIGOR SEGMENT
        ASSUME  CS: CODIGOR, DS: DATOSR
ENT_RES  PROC FAR         ; ← subrutina de entrada a programa residente

        PUSH DS           ; salvar registro  DS
        MOV  AX, DATOSR   ; para hacer DS:DX=dirección de MENSAJE
        MOV  DS, AX
        MOV  DX, OFFSET MENSAJE
        MOV  AH, 9        ; imprimirlo
        INT  21h
        POP  DS           ; restaurar  DS
        RET               ; y retornar (tipo FAR)
ENT_RES  ENDP

CODIGOR  ENDS

FIN_RES SEGMENT           ; segmento marcador de fin de residente
FIN_RES ENDS


; *** programa I ***
; contiene instrucciones para instalar parte anterior

CODIGOI  SEGMENT          ; segmento de código de I
        ASSUME CS: CODIGOI

COMIENZO_I:

      PUSH DS             ; DS =ES =comienzo de programa =segmento
                          ; prefijo de programa (SPP)
      MOV AX, 0           ; hacer DS:BX=0:4F0h
      MOV DS, AX          ; para salvar allí dirección de ENT_RES
      MOV BX, 4F0h        ;            (desplazamiento y segmento)
      MOV WORD PTR [BX], OFFSET ENT_RES ;       0:[4F0] =desplazamiento
      MOV WORD PTR [BX+2], SEG  ENT_RES ;       0:[4F2] =segmento
      POP DS

      MOV  DX, FIN_RES    ; mover segmento de FIN_RES a DX
      MOV  AX, DS         ; mover DS=SPP a AX
      SUB  DX, AX         ; DX=FIN_RES-SPP=tamaño (en parágrafos).
      SUB  AL, AL         ; código de retorno =0
      MOV AH, 31h         ; función para dejar residente bloque inicial
      INT 21h

CODIGOI   ENDS

PILAI   SEGMENT STACK     ; pila de programa IR (se usa sólo en I)
        DB 80 DUP(?)
PILAI   ENDS
        END  COMIENZO_I   ; fin de módulo IR y entrada en COMIENZO_I
```

Nótese que en este caso el segmento FIN_RES coincide con CODIGOI, de manera que podría haberse empleado directamente este último en lugar de aquél.

Recordemos que el valor inicial de DS (y ES) es igual al del segmento SPP. (También puede usarse la función AH = 62h con INT 21h para obtener este valor en BX).

).5 SUBRUTINAS DE INTERRUPCION

).5.1 Descripción

Una interrupción es una señal que realiza la unidad central de procesamiento (UCP) para que se suspenda el proceso o tarea que actualmente está ejecutando, pase a ejecutar un programa especial denominado subrutina o manejador de interrupción (interrupt handler) y cuando éste concluya usualmente se pueda continuar con el programa suspendido.

Las interrupciones pueden ser de dos clases:

a) de hardware: producidas por los dispositivos físicos, por ejemplo: generada en la UCP por una división entre cero (interna), o por un dispositivo periférico (externa).

b) de software: generadas en los programas con la instrucción INT n

Las instrucciones de clase hardware externas pueden ser a su vez enmascarables (aparecen en la línea INTR del procesador) o no enmascarables (aparecen en la línea NMI), estas últimas causadas por situaciones graves como por ejemplo un error en la paridad de memoria.

El sistema operativo DOS reserva los primeros 1024 bytes (= 1Kbytes) de la memoria para salvar allí una tabla de 256 direcciones absolutas, esto es, cada una de 4 bytes, de un conjunto de subrutinas de interrupción.

Las interrupciones son designadas o numeradas por los números 0 ,..., 255 (0FFh), a los cuales se les llama tipos de las mismas, y sus direcciones se suelen denominar vectores de interrupción.

Una interrupción de tipo N tiene su vector de interrupción almacenado en la dirección absoluta 0:4*N en la forma:

```
desplazamiento de subrutina en 0:4*N   (2 bytes)
segmento de subrutina  en     0:4*N+2  (2 bytes)
```

Por ejemplo, la interrupción de tipo 21h tiene su vector de interrupción en 0:84h (pues 4*21h = 84h) y accediendo a este lugar se puede encontrar que:

```
palabra en 0000:0084 = XXXX     (desplazamiento)
palabra en 0000:0086 = YYYY     (segmento)
```

de modo que YYYY:XXXX representará la dirección absoluta o vector de interrupción de la subrutina de interrupción de tipo 21h.

9.5.2 Procesamiento de una interrupción

Cuando se produce una interrupción de tipo N, el procesador salva en la pila el indicador de estados (flags) desactiva el indicador de interrupciones y efectúa una llamada indirecta y lejana a la subrutina de servicio asociada, como con CALL DWORD PTR 0:[4*N]. Esto significa que al ingresar a esta subrutina se hallan apilados: el indicador de estados y la dirección absoluta cs0:ip0 de la próxima instrucción a ejecutarse en el programa suspendido. Así, en la pila se encuentran:

```
SS:SP → ip0   (desplazamiento = 2 bytes)
        cs0   (segmento       = 2 bytes)
        f     (flags)
              (← cima de pila antes de interrupción)
```

La interrupción termina generalmente con una instrucción con la cual se retorna de un modo lejano y se desapila el indicador de estados de la pila en el registro indicador de estados: IRET

Dentro de una subrutina de interrupción pueden activarse o desactivarse nuevas interrupciones (externas y enmascarables) con las instrucciones:

```
STI  ;  set interrupt flag   = permite nuevas interrupciones
CLI  ;  clear interrupt flag = desactiva interrupciones
```

que modifican el bit de interrupción del registro indicador de estados.

9.5.3 Algunas interrupciones

Indicamos una lista de interrupciones con tipos y direcciones en el sistema hexadecimal. La columna de direcciones se refiere a la dirección en memoria en donde se localiza el vector de interrupción, es decir allí se ubica la dirección absoluta de la subrutina de interrupción correspondiente.

Tipo	Dirección	Función
0	0-3	división entre cero
3	8-B	instrucción de punto de parada
5	14-17	impresión de pantalla
8	20-23	temporizador
9	24-27	interrupción de teclado
10	40-43	entrada/salida de video (pantalla)
11	44-47	comprobación de equipo
12	48-4B	comprobación de tamaño de memoria
13	4C-4F	entrada/salida de disquete
14	50-53	comunicaciones en serie
16	58-5B	entrada por teclado
17	5C-5F	salida por impresora
19	64-67	iniciador de sistema
1A	68-6B	hora
1B	6C-6F	obtener control sobre Control Break de teclado
1C	70-73	obtener control sobre interrupción de temporizador (contador)
21	84-87	Manejador de funciones del DOS
23	8C-8F	Manejador de Control-C (o Control Break)
25	94-97	Lectura absoluta de disco
26	98-9B	Escritura absoluta de disco
27	9C-9F	Terminar y dejar residente

9.5.4 Subrutinas de interrupción definidas por el usuario

El usuario puede construir sus propias subrutinas de interrupción simplemente reemplazando el vector de interrupción de la subrutina original por la dirección de su subrutina. De esta manera cuando se genere una interrupción del mismo tipo, la transferencia se realizará a la subrutina del usuario - se dice que esta subrutina "intercepta la llamada" a la original -. La nueva subrutina puede hacer varias tareas, entre ellas llamar o transferir el proceso a la antigua hasta devolver el control al programa suspendido.

La mayor parte de los programas residentes se instalan como manejadores de interrupciones de modo que pueden ser activados por acciones periódicas (por ejemplo el temporizador del reloj), o por otras (por ejemplo, al presionar una cierta tecla), etc. Muchas subrutinas de interrupción no tiene uso alguno: consisten sólo de la instrucción IRET.

Instalación de subrutinas de interrupción

El procedimiento para instalar subrutinas de interrupción definidas por el usuario es bastante sencillo: Basta sustituir el vector de interrupción original por la dirección absoluta de la nueva subrutina.

Si se intenta usar la subrutina de interrupción original es necesario salvar antes su vector de interrupción.

Las funciones del DOS para hacer estas tareas son:

a) Obtener vector: AH=35h, AL=n y se ejecuta con INT 21h.
Resultado: ES=segmento y BX=desplazamiento de la subrutina de interrupción actual.

b) Establecer vector: AH=25h, AL=n, DS=segmento y DX=desplazamiento de la nueva subrutina y se ejecuta con INT 21h.
Resultado: los valores de DX y DS se almacenan a partir de la dirección 0:4*n, respectivamente.

Aquí n es el tipo de interrupción.

Otro método consiste en acceder directamente a la dirección 0:4*n para extraer su contenido o modificarlo. En el último caso es necesario utilizar: CLI antes y STI después de la modificación.

Si no se desea que la subrutina de interrupción quede residente se debe restaurar, empleando el mismo procedimiento, el vector original.

Proceso en la subrutina de interrupción

Ya se ha indicado que las subrutinas de interrupción son activadas cuando se genera una interrupción del mismo tipo.

Al ingresar a la subrutina se hallan apilados (de la cima al fondo de la pila):

1) la dirección de retorno (desplazamiento y segmento)
2) el registro indicador de estados

Es necesario asegurarse que los registros mantengan sus valores originales al regresar al programa suspendido. (Algunas interrupciones modifican los valores de ciertos registros incluyendo el registro indicador de estados). Por lo tanto deben ser salvados, por ejemplo en pila, si han de ser modificados, y recuperados antes de retornar.

Se puede llamar a la subrutina de interrupción original mediante:

```
PUSHF              ;  salva indicador de estados
CALL DWORD PTR        dirección de subrutina original
```

de esta manera cuando finalice ésta se retornará a la subrutina de interrupción actual (a causa de la instrucción IRET).

Otra forma consiste en retornar al programa suspendido saliendo a través de la subrutina de interrupción antigua empleando un salto lejano indirecto:

```
JMP DWORD PTR        dirección de subrutina original
```

siempre que el estado de la pila sea idéntico al que tenía al ingresar a la subrutina de interrupción. Así, la instrucción IRET de la subrutina original producirá el retorno al programa suspendido.

9.5.5 Ejemplo de una subrutina de interrupción no residente

El siguiente programa muestra una subrutina de interrupción que intercepta a la interrupción de teclado (de número 9). La subrutina tiene por propósito convertir a mayúsculas los caracteres de letras minúsculas que se ingresen por el teclado.

Esta interrupción es activada cada vez que se presiona una tecla. El sistema DOS mantiene un área de 16 octetos dobles llamada buffer del teclado, que se extiende en las direcciones absolutas 40:1E hasta 40:3C, en donde se guardan los caracteres recientemente ingresados (cada uno ocupa un octeto doble, es decir se representa por su código extendido con su valor ASCII en el octeto inferior).

Las 4 direcciones anteriores al buffer contienen las posiciones (desplazamiento respecto del segmento 40h) de la cabeza (head) y cola (tail) del buffer, respectivamente. La zona de almacenamiento es tratada con un buffer circular o enrrollado (esto es, a la dirección final 40:3C le "sigue" la posición inicial 40:1E).

La cola siempre apunta a la posición siguiente al último carácter ingresado. Si éste se halla en la última posición 40:3C, entonces la cola apunta a la posición inicial 40:1E. El buffer se considera vacío (no hay caracteres) si la cabeza y la cola coinciden.

Cada vez que se presione una tecla se genera una interrupción de tipo 9 y por lo tanto se ejecutará la nueva interrupción, a la que hemos designado por NUEVA9. La primera acción de esta subrutina es llama a la original cuya dirección ha sido previamente salvada en la variable de 4 octetos VECTOR9_ANTIGUO a fin de que procese la entrada de caracteres por el teclado. Al retornar de la subrutina original, si el buffer no está vacío, se accede al último carácter ingresado y en caso de que éste sea una letra minúscula se convierte a mayúscula. Finalmente se produce el retorno de la interrupción.

En este caso se muestra el uso de la nueva interrupción empleando la función de lectura de cadenas AH=10 con INT 21h.

Al finalizar el programa se restaura el vector original.

El programa se ha escrito para ser compilado a uno de tipo .COM:

```
MASM INT9;
LINK INT9;
EXE2BIN INT9.EXE INT9.COM
```

El texto del programa es:

```
NUM_INTR    9h              ; número de interrupción

SEG_BUF     EQU 40h         ; segmento de buffer del teclado
INI_BUF     EQU 1Eh         ;    inicio de buffer
FIN_BUF     EQU 3Ch         ;    fin de buffer

CODIGO SEGMENT
       ASSUME CS:CODIGO, DS: CODIGO
       ORG 100h
PRINCIPAL PROC NEAR

       MOV AX, CS          ; hacer DS=CS
       MOV DS, AX

  ; 1) Obtener vector de interrupción actual y salvarlo en
                           ; variable de 4 octetos VECTOR9_ANTIGUO
       MOV BX, OFFSET VECTOR9_ANTIGUO  ;  DS:BX apunta a variable

       MOV AX,0            ; ES:SI=0:4*NUM_INTR apunta a vector antiguo
                           ; de interrupción
       MOV ES,AX
       MOV SI, 4*NUM_INTR;

       MOV AX,ES:[SI]      ; pasar desplazamiento
       MOV [BX],AX

       MOV AX,ES:[SI+2]    ; pasar desplazamiento
       MOV [BX+2],AX

  ; 2) Establecer nuevo vector de interrupción = dirección de NUEVA9

       CLI                 ; desactiva interrupciones

       MOV AX, OFFSET NUEVA 9
       MOV ES:[SI],AX
       MOV AX, CS          ; CS = segmento de NUEVA9
       MOV ES:[SI+2],AX

       STI                 ; activa interrupciones
```

```
  ; 3) Efectuar una llamada a la función 10 INT 21h para leer el teclado

      MOV DX, OFFSET CADENA
      MOV AH, 10
      INT 21h


  ; 4) Restaurar vector de interrupción original
       ; ES:SI  sigue siendo 0:4*9

      CLI
      MOV BX, OFFSET VECTOR9_ANTIGUO
      MOV AX, [BX]
      MOV ES:[SI],AX
      MOV AX,[BX+2]
      MOV ES:[SI+2],AX
      STI

  ; 5) Fin de programa

      MOV AH,4Ch
      INT 21h

PRINCIPAL ENDP
  ; *** área de datos ***

      MENSAJE DB        "ingrese texto  $"
      CADENA            DB 30,0
                        DB 31 DUP (?)
      VECTOR9_ANTIGUO   DD ?              ; variable para dirección (4 bytes)

NUEVA9  PROC NEAR
                                          ; salvar registros
      PUSH AX
      PUSH BX
      PUSH CX
      PUSH DX
      PUSH SI
      PUSH DI
      PUSH DS
      PUSH ES

      PUSH CS                             ; hacer DS=CS
      POP DS

      PUSHF                               ; llamar subrutina antigua
      CALL VECTOR9_ANTIGUO
```

```
; Acceder al buffer para tratar carácter ingresado

      MOV AX,SEG_BUF          ; hacer ES:SI=SEG_BUF:INI_BUF
      MOV ES,AX
      MOV SI, INI_BUF         ;
                              ;   cabeza = ES:[SI-4],  cola = ES:[SI-2]
      MOV BX, ES: SI-2        ; obtener cola en BX
      CMP ES: SI-4 , BX       ; comparar cabeza con cola
      JE  FIN9                ; son iguales: cola vacía, ir a final

                              ;  apuntar con ES:BX a último carácter: se ha
                              ;  lla "antes" de la cola

      CMP BX,INI_BUF
      JA  SEGUIR
      MOV BX,FIN_BUF          ; cola = comienzo: último carácter
                              ; está al final del buffer
      JMP SHORT LISTO
SEGUIR:
      SUB BX,2                ; cola está después del comienzo del buffer
                              ; hay que restarle 2 direcciones para  que
                              ; apunte a carácter

LISTO:
      MOV AX,ES:[BX]          ; AX tiene carácter ingresado
      CMP AL,0                ;   ; comprobar si es tecla especial
      JE  FIN9
      CMP AL, "a"             ; comprobar si es letra minúscula
      JB  FIN9
      CMP AL, "z"
      JA  FIN9
      SUB AL,32               ;   ; es letra minúscula: convertir
      MOV ES:[BX],AX          ; y reemplazar en buffer
FIN9:
                              ;   restaurar registros

      POP ES
      POP DS
      POP DI
      POP SI
      POP DX
      POP CX
      POP BX
      POP AX
      IRET                    ; retornar de interrupción
NUEVA9 ENDP

CODIGO ENDS
      END PRINCIPAL           ; fin de programa INT9.ASM
```

9.5.6 Ejemplo de una subrutina de interrupción residente

A continuación mostramos una subrutina residente que intercepta a las interrupciones del teclado(9) y a la interrupción 1Ch del contador de tiempo del reloj. Una vez que se instale el programa ejecutable como programa residente, presionando alternadamente la tecla Alt H se mostrará o nó la hora del sistema en la fila 0 columna 70 de la pantalla.

En el programa, la interrupción 9 es intervenida como en el ejemplo anterior, pero esta vez para comprobar si el carácter ingresado coincide con la tecla referida y en caso de que así sea se modifica el valor de una variable de control MOSTRAR : se intercambian los valores 1 y 0.

La interrupción 1Ch es activada por el ciclo ("tick") del temporizador: cerca de 18.2065 ticks por segundo = 1092 por minuto = 65443 por hora.

El sistema mantiene en la dirección absoluta 40:6Ch un contador de ciclos como un número de 32 bits (parte inferior en 40:6C y parte superior en 40:6E).

La subrutina creada intercepta esta interrupción, comprueba el valor de la variable MOSTRAR: si es 1, accede al contador, convierte su valor en hora ASCII o texto de la forma hh:mm:ss y lo imprime, y si es 0 simplemente retorna.

Cálculo de la hora según el contador

Si N es el valor del contador entonces:

(1) número de horas = N/65543

ó puesto que el denominador excede el rango de los números de 16 bits (sin signo) y teniendo en cuenta que 32771.5*2:

se divide N entre 32771 y el cuociente resultante entre 2.

Sin embargo, al aplicar este método el resto obtenido puede ser distinto del correcto lo cual ocurre precisamente si el primer cuociente es impar, en cuyo caso la corrección consiste en sumar 32771 al resto ya calculado.

(2) número de minutos = resto de (1) dividido entre 1092

(3) número de segundos= resto de (2) dividido entre 18.

La impresión de la hora se realiza escribiendo directamente en la dirección del buffer de video para lo cual se usa la subrutina DETERMINA_PANTALLA en la parte de instalción con el propósito de determinar el segmento de video: 0B000h ó 0B800h según que el adaptador sea monocromático o de color. Puesto que cada carácter en el buffer de video se representa por 2 octetos (atributo y código ASCII) el área en donde se almacena el texto de la hora, dada por las variables HORAS, MINUTOS y SEGUNDOS, ha sido iniciada con un atributo definido por la constante MODO.

En esta ocasión se han empleado las funciones del DOS (35h y 25h, INT 21h) para obtener y establecer vectores de interrupción, respectivamente.

El texto del programa es el siguiente:

```
;  *** PROGRAMA HORA.ASM  ***
;  la versión ejecutable HORA.EXE se instala como programa residente

SEG_BUF          EQU 40h        ; dirección de buffer de teclado
INICIO_BUF       EQU 1Eh        ; inicio de buffer
FIN_BUF          EQU 3Ch        ; fin de buffer
TECLA_ACCION     EQU 2300h      ; tecla acción Alt H

MONO             EQU 0B000h     ; segmento de pantalla según adaptador
COLOR            EQU 0B800h     ; de video

CTE_SI           EQU 1          ; constantes para variable MOSTRAR
CTE_NO           EQU 0

POS_PANTALLA     EQU 2*70       ; posición de lugar de impresión
                                ; de hora : fila,0, columna 70
MODO             EQU 70h        ; atributo de impresión de hora
SEG_CONTADOR     EQU 40h        ; dirección de contador de tiempo
DESP_CONTADOR    EQU 6Ch
```

```
CODIGO SEGMENT
       ASSUME CS:CODIGO, DS :CODIGO

    ; *** área de datos ***

    VECTOR9_ANTIGUO  DD ?            ; variable para dirección (4 bytes)
    MOSTRAR          DW CTE_NU       ; variable para controlar impresión
                                     ; de hora
    SEG_PANTALLA     DW ?            ; para segmento de pantalla

    ; área de HORA en ASCII

    DIEZ             DB  10          ; divisores para efectuar conversiones
    CONV_HORAS       DW  32771
    CONV_MINUTOS     DW  1092
    CONV_SEGUNDOS    DB  18
    HORAS            DB  0,MODO,0,MODO, ":", MODO
    MINUTOS          DB  0,MODO,0,MODO, ":", MODO
    SEGUNDOS         DB  0,MODO,0,MODO
    LONGITUD         EQU 8           ; longitud de texto de hora en
                                     ; palabras dobles

NUEVA9  PROC NEAR
                                     ; salvar registros
        PUSH AX
        PUSH BX
        PUSH CX
        PUSH DX
        PUSH SI
        PUSH DI
        PUSH DS
        PUSH ES

        PUSH CS                      ; hacer DS = CS
        POP DS

        PUSHF                        ; llamar subrutina antigua
        CALL VECTOR9_ANTIGUO


                ; acceder al buffer para tratar carácter ingresado
        MOV AX,SEG_BUF
        MOV ES,AX

        MOV SI, INICIO_BUF           ; ES:SI apunta al comienzo del buffer
        MOV BX, ES:[SI-2]            ;   obtener cola en BX
        CMP ES:[SI-4], BX            ;   comparar cabeza con cola
        JE  FIN9                     ;   son iguales: cola vacía, ir a final
```

```
                ; apuntar a último carácter

        CMP BX,INICIO_BUF
        JA  SEGUIR
        MOV BX,FIN_BUF              ; cola apunta al comienzo: último
                                    ; carácter está al final del buffer
        JMP SHORT LISTO
SEGUIR:
        SUB BX,2                    ; cola está después del comienzo del
                                    ; buffer hay que restarle 2 para que
                                    ; apunte a carácter

LISTO:
        MOV AX,ES:[BX]              ; AX tiene carácter ingresado
        CMP AX,TECLA_ACCION
        JNE FIN9
        XOR MOSTRAR,CTE_SI          ; cambiar
        MOV ES:[BX-2],BX            ; eliminar carácter ingresado redu-
                                    ;    ciendo cola
FIN9:
                                    ; restaurar registros
        POP ES
        POP DS
        POP DI
        POP SI
        POP DX
        POP CS
        POP BX
        POP AX
        IRET                        ; retornar de interrupción
NUEVA9 ENDP

NUEVA1C PROC
        CLI
        PUSH AX
        PUSH BX
        PUSH CX
        PUSH DX
        PUSH SI
        PUSH DI
        PUSH DS
        PUSH ES

        PUSH CS                     ; hacer DS =CS
        POP DS
        CMP MOSTRAR, CTE_NU
        JE  FIN1C
        CALL ESCRIBIR_HORA
```

```
FIN1C:
                                ; restaurar registros
        POP ES
        POP DS
        POP DI
        POP SI
        POP DX
        POP CX
        POP BX
        POP AX
        STI
        IRET                    ; retornar de interrupción

NUEVA1C ENDP

ESCRIBIR_HORA  PROC NEAR

        ; acceder al contador de tiempo

        MOV AX,SEG_CONTADOR
        MOV ES,AX
        MOV SI,DESP_CONTADOR
        MOV AX,ES:[SI]          ; contador en DS AX (32 bits)
        MOV DX,ES:[SI+2]

        ; obtener horas y resto de contador en AX

        DIV CONV_HORAS          ; AX=hora*2; DX=resto
        SHR AX,1                ; dividir entre 2: AL contiene hora

                        ; corregir resto si AX ha sido impar : hay acarreo

        JNC SIGUE_ESCR          ; si bit salido es cero resto es correct
        ADD DX,CONV_HORAS       ; de otra manera actualizar resto

SIGUE_ESCR:

        DIV DIEZ                ; pasar hora a decimal : cuociente=AL
                                ; =unidades y  resto=AH=decenas
        XCHG AL,AH              ; AH=decenas AL=unidades
        OR   AX,3030h           ; convertir a dígitos ASCII
        MOV  HORAS[0],AH
        MOV  HORAS[2],AL
        MOV  AX, DX             ; resto en AX

        ; obtener minutos y resto de contador en AX

        MOV DX,0                ; extender a 32 bits
        DIV CONV_MINUTOS        ; AX=minutos, DX=resto
        DIV DIEZ                ; pasar a  minutos a decimal
        XCHG AL,AH              ; AH=decenas  AL=unidades
        OR  AX,3030h            ; en dígitos ASCII
        MOV MINUTOS[0],AH
        MOV MINUTOS[2],AL
        MOV AX,DX               ; resto en AX
```

```
        ; obtener segundos
        DIV CONV_SEGUNDOS          ; AL=segundos y AH=resto
        MOV AH,0                   ; AX=segundos
        DIV DIEZ
        XCHG AL,AH
        OR AX,3030h
        MOV SEGUNDOS[0],AH
        MOV SEGUNDOS[2],AL

        ; mover área de hora ASCII (16 bytes) a SEG_PANTALLA:POS_PANTALLA
        MOV SI,OFFSET HORAS        ; DS:SI apunta al área
        MOV AX,SEG_PANTALLA
        MOV ES,AX                  ; ES:DI apunta a posición de pantalla
        MOV DI, POS_PANTALLA
        MOV CX,LONGITUD            ; número de palabras a transferir
OTRO:   MOV AX,[SI]
        MOV ES:[DI],AX
        INC SI
        INC SI
        INC DI
        INC DI
        LOOP OTRO
        RET
ESCRIBIR_HORA  ENDP
   FIN_RES LABEL WORD              ; etiqueta de fin de residente
INSTALAR PROC NEAR
        MOV AX, CS                 ; hacer DS=CS
        MOV DS,AX
        CALL DETERMINA_PANTALLA
        MOV AH,35h             ; obtener antiguo vector de interrupción 9
        MOV AL,9                   ; en ES:BX
        INT 21h
                ; y salvarlo en VECTOR9_ANTIGUO
        MOV SI, OFFSET VECTOR9_ANTIGUO
        MOV [SI],BX
        MOV AX, ES
        MOV [SI+2],AX
        MOV AH,25h             ; establecer nueva interrupción 9 en NUEVA9
        MOV AL,9
        MOV DS,OFFSET NUEVA9       ; DS=CS=segmento
        INT 21h
        MOV AH,25h       ; establecer nueva interrupción 1Ch en NUEVA1C
        MOV AL,1Ch
        MOV DX, OFFSET NUEVA1C
        INT 21h
```

```
        ; finalizar dejando residente hasta marca  FIN_RES

        MOV DX,OFFSET FIN_RES
        ADD DX,100h                 ; añadir segmento de prefijo de
                                    ;     programa
        MOV CL,4                    ; convertir a parágrafos
        SHR DX,CL                   ;
        INC DX                      ; DX = tamaño en parágrafos

        MOV AH,31h                  ; terminar y dejar residente
                                    ;     hasta FIN_RES
        INT 21h


INSTALAR ENDP


DETERMINA_PANTALLA PROC NEAR
        MOV AH,0Fh
        INT 10h
        CMP AL,7                    ; compara si adaptador es monocromá
                                    ; tico
        JE  ES_MONO
        MOV SEG_PANTALLA, COLOR
        JMP SHORT FIN_DET

ES_MONO:
        MOV SEG_PANTALLA, MONO

FIN_DET:
        RET

DETERMINA_PANTALLA  ENDP

CODIGO ENDS


PILA SEGMENT STACK

    DB 128 DUP (?)

PILA ENDS


        END INSTALAR                ; *** FIN DE PROGRAMA ***
```

9.5.7 Ejemplo de subrutina residente usando lenguaje de programación C (Turbo C)

El siguiente programa es similar al del ejemplo anterior pero ha sido escrito usando el compilador de Turbo C.

```
#define tecla_accion 0x2300     /* Alt H  */
#define cte_si 1                /* valores de variable mostrar */
#define cte_no 0
#define mono   0xb000           /* valores de segmento de pantalla */
#define color  0xb800
#define modo   0x70
#define cursor 140              /* posición del cursor 0,70: 2*70 */

#define seg_bios       0x40     /* segmento de bios */
#define comienzo_buf   0x1e     /* definen buffer de teclado */
#define fin_buf        0x3c

#define desp_contador 0x6c      /* desplazamiento de contador de reloj en
                                   seg_bios */

/* variables */

char mostrar=cte_no;            /* control sobre impresión de hora */
unsigned far *buffer,far *pcar; /*punteros a buffer de teclado
unsigned cola;

long unsigned far *pcontador;     /* puntero a contador de reloj */
long unsigned temp;
unsigned horas, minutos, segundos, resto;
unsigned far *punt_pantalla     /* punteros a zona de pantalla en memoria */
unsigned far *pos_pantalla;

                                /* área para hora expresada como texto */
char hora_texto[18] = {0,modo, 0,modo, ':', modo, 0,modo, 0,modo,
                       ',' , modo, 0,modo, 0,modo,0,0};


                                /* los dos últimos bytes se hacen nulos para
                                   usarlos como indicadores de final  */

unsigned *punt_hora;            /* puntero a hora_texto */

#include <dos.h>                /* incluye:

          FP_SEG y FP_OFF para hallar segmento y desplazamiento de objetos
                                */

void interrupt nueva9();        */ nueva subrutina para interrupción 9 */
void interrupt nueva1c();       */ nueva subrutina para interrupción 0x1C  */
void interrupt (*antigua9)();   */ puntero antigua9 apunta a subrutina
                                     antigua (*antigua9)() */
```

```
main()          /* instalación de subrutinas de interrupción y terminar
                   dejando residente */

{
      buffer=MK_FP(seg_bios,comienzo_buf);    /* establecer direcciones */
      pcontador=MK_FP(seg_bios, desp_contador);
      obtiene_pantalla();
      antigua9=getvect(9);

      setvect(9,nueva 9);              /* establecer nuevas subrutinas */
      setvect(0x1c,nueva1c);
      keep(0,300);                  /* terminar y dejar residente 300 parágra-
                                       fos=4800 bytes, pues programa .exe con-
                                       siste de cerca de 3100 bytes */
}

obtiene_pantalla()                  /* establece posición de impresión en la
                                       pantalla teniendo en cuenta tipo de adapta-
                                       dor de video  */
{     int valor;
      valor=biosequip();
      valor>>=4;                    /* desplazar 4 bits a la derecha para ubi-
                                       car bits 5 y 4 en posiciones 1 y 0, res
                                       pectivamente */

      valor&=3;                     /* enmascarar los dos bits de orden inferior */
      if (valor==3) pos_pantalla=MK_FP(mono,cursor) ;
      else  pos_pantalla=MK_FP(color,cursor);
}
void interrupt nueva9()
{
      (*antigua9)();                /* llamada a subrutina antigua  */
      cola=buffer[-1];
      if  (buffer[-2]!=cola)
          { if (cola==comienzo_buf) cola=fin_buf;  else cola-=2;
               pcar=MK_FP(seg_bios,cola);
               if (*pcar==tecla_accion)
                 {  buffer -1 =cola; mostrar=mostrar^1; }  /* alternar valor
                                                              de mostrar */
          }
}

void interrupt nueva1c()

{
      disable();
      if (mostrar)
        { temp=*pcontador;
          horas      =temp /65543;  resto= temp % 65543;
          minutos    = resto/1092;  resto= resto% 1092;
          segundos   = resto/18;
```

```
            /* pasar hora en números a texto */

            hora_texto[ 0 ] = horas/10 + '0' ;
            hora_texto[ 2 ] = horas%10 + '0' ;
            hora_texto[ 6 ] = minutos/10 + '0';
            hora_texto[ 8 ] = minutos%10 + '0';
            hora_texto[ 12] = segundos/10 +'0';
            hora_texto[ 14] = segundos%10 +'0';
            punt_hora=(unsigned *) hora_texto;
            punt_pantalla  = pos_pantalla;


            /* mover datos de hora_texto a pos_pantalla */

            while (*punt_hora) *punt_pantalla++=*punt_hora++;
          }
        enable();
}
```

LISTA DE PROGRAMAS

INDICE DE INTERRUPCIONES Y FUNCIONES DEL DOS

INDICE ALFABETICO

Lenguaje ensamblador Macro Assembler de Maynard Kong se terminó de imprimir el mes de setiembre de 1989, en los talleres de Editorial e Imprenta Desa (Reg. Ind. 16521) General Varela 1577, Lima 5, Perú. La edición estuvo al cuidado de *Miguel Angel Rodríguez Rea.* Se hicieron mil quinientos ejemplares.

BIBLIOTECA UNIVERSITARIA DEL FONDO EDITORIAL DE LA PONTIFICIA UNIVERSIDAD CATOLICA DEL PERU

MARCIAL RUBIO CORREA

El sistema jurídico (Introducción al Derecho). 4a. ed. (Colección de Textos Jurídicos, 1).

HUGO MEDINA GUZMAN

Física básica; teoría y problemas. (Agotado).

CARLOS BLANCAS y MARCIAL RUBIO CORREA

Derecho constitucional general. 2a. ed. (Colección de Textos Jurídicos, 2).

MAYNARD KONG

Basic

Lenguaje de programación Pascal. 3a. ed.

Lenguaje de programación C

Lenguaje ensamblador Macro Assembler

Cálculo diferencial

Cálculo integral