Cuadernos de prácticas de Informática Industrial

MÓDULO 1

Enunciados de prácticas de programación en Ensamblador

Fco. Javier Martínez de Pisón Ascacíbar
Joaquín Ordieres Meré
Manuel Castejón Limas
Fco. Javier de Cos Juez
Montserrat Gil Martínez

UNIVERSIDAD DE LA RIOJA
1992-2002 / DÉCIMO ANIVERSARIO

didáctica · didáctica · didáctica · didáctica · didáctica

CUADERNOS DE PRÁCTICAS
DE
INFORMÁTICA INDUSTRIAL

MÓDULO 1:

ENUNCIADOS DE PRÁCTICAS DE

PROGRAMACIÓN EN ENSAMBLADOR

MATERIAL DIDÁCTICO

Ingenierías

nº 20

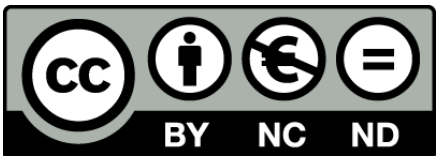
Francisco Javier Martínez de Pisón Ascacíbar
Joaquín Ordieres Meré
Manuel Castejón Limas
Francisco Javier de Cos Juez
Montserrat Gil Martínez

CUADERNOS DE PRÁCTICAS DE INFORMÁTICA INDUSTRIAL

MÓDULO 1:

ENUNCIADOS DE PRÁCTICAS DE PROGRAMACIÓN EN ENSAMBLADOR

UNIVERSIDAD DE LA RIOJA
SERVICIO DE PUBLICACIONES
2001



Cuadernos de prácticas de informática industrial.

Módulo 1: enunciados de prácticas de programación en ensamblador

publicaciones.unirioja.es

E-mail: publicaciones@unirioja.es

ISBN: 978-84-694-0879-7

INTRODUCCIÓN

La programación en ensamblador, aunque en nuestros días es ignorada y apartada de muchos cursos docentes debido a su dificultad, permite que el alumno adquiera conocimientos prácticos del funcionamiento a bajo nivel de un ordenador que con otros lenguajes no se obtiene tan fácilmente. Conceptos como: los modos de direccionamiento de la memoria, tipos de registros, funcionamiento de la Pila, los *flags*, modos de funcionamiento de la CPU, puertos de entrada/salida, programación de dispositivos hardware, manejo de la memoria, tipos de instrucciones de un microprocesador, manejo de interrupciones, la BIOS, programas residentes, métodos de parametrización y paso de información entre subrutinas, funcionamiento de la pantalla en modo texto y gráfico, conversión de datos, aritmética entera y en coma flotante, etc., son conocimientos básicos que todo Ingeniero Electrónico o Informático debe conocer. Además, este tipo de aprendizaje, aunque es una ardua tarea, crea una base sólida para comprender la programación de cualquier otro microprocesador, microcontrolador, DSP, sistema de desarrollo, etc., que pueda utilizarse en el futuro.

Este primer volumen forma parte de un conjunto de prácticas de Ensamblador que se han ido realizado durante varios años como parte inicial de la asignatura de Informática Industrial de la carrera de Ingeniería Técnica Industrial Electrónica en la Universidad de La Rioja. Este módulo tiene como objetivo principal que el alumno adquiera todos los conceptos arriba expuestos de una forma lo más práctica y didáctica posible. Se pretende, por lo tanto, que el estudiante mediante una serie de ejercicios propuestos que deba resolver, vaya asimilando progresivamente todos estos conceptos. Por supuesto, estos conocimientos tienen que completarse con otros contenidos como por ejemplo: metodologías de programación, lenguajes de alto nivel, sistemas operativos, etc., que sirvan para perfeccionar la formación del Ingeniero Electrónico en el campo del desarrollo de software del sector Eléctrico/Electrónico.

Por último destacar, que los ejercicios de ejemplo que se plantea en cada uno de los capítulos de este libro, se han probado y realizado con la versión 2.0 del compilador Turbo Assembler® de la Compañía Borland, aunque son fácilmente convertibles a otros compiladores (Microsoft Assembler, etc.).

Práctica 1

1. Manejo de Ensamblador, Linker y Debugger

1.1. Compilación y ejecución paso a paso de un programa ejemplo

Entender el funcionamiento del siguiente programa "PRACT1A.ASM", ensamblarlo y ejecutarlo paso a paso viendo como se modifican los registros y los *flags*:

```
;PROGRAMA EJEMPLO: PRACT1A.ASM
;
;PROGRAMA EJEMPLO: ESCRIBIR UN TEXTO EN PANTALLA USANDO LA FUNCIÓN 09H
DEL DOS
;-------------------------------------------------------------
CR     EQU   13        ;Retorno de carro
LF     EQU   10        ;Salto de línea
;
;-------------------------------------------------------------
;
;Segmento de Datos
;---------
;
DATOS SEGMENT         ;Comienzo segmento DATOS
TEXTO DB 'ESTE ES UN TEXTO DE PRUEBA',CR,LF ;TEXTO A IMPRIMIR
       DB '$'          ;Delimitador de fin de texto
DATOS ENDS
;
;-------------------------------------------------
;
;Segmento de Pila
;-------------------------------------------------
;
PILA SEGMENT STACK  ;Comienzo segmento PILA
     DB 128 DUP('PILA')    ;Inicialización PILA
PILA ENDS
;
;-------------------------------------------------
;
;Segmento de Código
;------------------
;
CODIGO    SEGMENT     ;Comienzo segmento CODIGO
EJEMPLO   PROC FAR
          ASSUME   CS:CODIGO,DS:DATOS,SS:PILA
;
          PUSH DS     ;Guarda Segmento en pila (DS:AX dirección retorno)
          SUB AX,AX   ;Borrar registro AX=0
          PUSH AX     ;Guarda en Pila AX (IP=0)
;
          MOV AX,DATOS     ;AX=DATOS (SEGMENTO DE DIRECCION DATOS)
          MOV DS,AX        ;DS=AX
;
```

```
          LEA DX,TEXTO     ;DX=DESPLAZAMIENTO DE TEXTO
          CALL ESCRIBIR    ;SUBRUTINA DE ESCRIBIR TEXTO
          RET               ;RETORNA
;
EJEMPLO   ENDP              ;FIN DE PROCEDIMIENTO
;
ESCRIBIR PROC              ;PROCEDIMIENTO 'ESCRIBIR'
          PUSH AX          ;GUARDA EN PILA AX
          MOV AH,9         ;AH=9  FUNCION NUMERO 9 'SALIDA DE CARACTERES'
          INT 21H          ;LLAMADA A INTERRUPCION DEL DOS, CON FUNCION 9
          POP AX           ;RECUPERA EL REGISTRO AX
          RET              ;RETORNAR
ESCRIBIR ENDP              ;FIN DE PROCEDIMIENTO ESCRIBIR
CODIGO    ENDS             ;FIN DE SEGMENTO DE CODIGO
          END EJEMPLO      ;FIN DE PROGRAMA EJEMPLO
```

Figura 1-1. Programa que escribe una cadena de caracteres en pantalla.

1.2. Ensamblador (TASM), Linkador (TLINK) y Depurador (TD)

El programa de la Figura 1-1 se puede escribir con cualquier editor de textos, por ejemplo el editor del DOS "*EDIT.COM*".

Una vez tenemos el archivo "*Nombre.ASM*" lo ensamblamos con "***TASM.EXE***" de la siguiente forma:

TASM –zi Nombre.ASM

que nos generará otro archivo con extensión OBJ: "*Nombre.OBJ*". El comando *"-zi"* incluye las etiquetas del código fuente para el depurador.

Usando "***TLINK.EXE***" se *linkan* ("une") todos los OBJ que necesitemos (en este caso sólo tenemos uno) y nos generará un ejecutable ".EXE".

TLINK –v Nombre.OBJ

que nos generará un archivo "*Nombre.exe*", que podemos ejecutar y comprobar. El comando *"-v"* incluirá la tabla de contenidos con las etiquetas en el ejecutable.

Para depurar usaremos el programa "***TD.EXE***".

Ejemplo:

EDIT PRACT1.ASM	*;EDITAMOS PRACT1.ASM*
TASM –zi PRACT1.ASM	*;ENSAMBLAMOS PRACT1.ASM*
TLINK –v PRACT1.OBJ	*;LINKAMOS PRACT1.ASM*
PRACT1	*;EJECUTAMOS PRACT1.ASM*
TD PRACT1.EXE	*;EJECUTAMOS CON TD (PARA DEPURAR)*

1.3. PROGRAMA EJEMPLO "PRACT1B.ASM". SITÚA TEXTO EN UNA POSICIÓN DETERMINADA

Escribir, ejecutar paso a paso el programa de la Figura 1-2 y comprender su funcionamiento. Este programa sitúa en una posición determinada de la pantalla un texto.

```
;PROGRAMA PRACT1B.ASM
;
;ESCRIBIR EN UNA POSICIÓN DE LA PANTALLA
;---------------------------------------
;
CR          EQU  13         ;Retorno de carro
LF          EQU  10         ;Salto de línea
POSX        EQU  5          ;Posicion X (0 A 79)
POSY        EQU  5          ;Posicion Y (0 A 24)
;
;-------------------------------------------------------
;
;Segmento de Datos
;---------
;
DATOS SEGMENT               ;Comienzo segmento DATOS
TEXTO DB 'ESTO ESTA ESCRITO EN LA POSICION (5,5)',CR,LF
                            ;TEXTO A IMPRIMIR
      DB '$'                ;Delimitador de fin de texto
DATOS ENDS
;-------------------------------------------------------
;
;Segmento de Pila
;----------------
;
PILA SEGMENT STACK  ;Comienzo segmento PILA
     DB 128 DUP('PILA')     ;Inicialización PILA
PILA ENDS
;
;-------------------------------------------------------
;
;Segmento de Código
;------------------
;
CODIGO   SEGMENT    ;Comienzo segmento CODIGO
EJEMPLO1B PROC FAR
          ASSUME   CS:CODIGO,DS:DATOS,SS:PILA

          PUSH DS       ;Guarda Segmento en pila (DS:AX dirección retorno)
          SUB AX,AX     ;Borrar registro AX=0
          PUSH AX       ;Guarda en Pila AX (IP=0)

          MOV DL,POSX   ;MUEVO EN DL LA POSICION X
          MOV DH,POSY   ;MUEVO EN DH LA POSICION Y

          CALL POSICION ;SUBRUTINA PARA POSICIONAR EL CURSOR

          MOV AX,DATOS
          MOV DS,AX
          LEA DX,TEXTO
          CALL ESCRIBIR ;SUBRUTINA DE ESCRIBIR TEXTO
```

```
          RET              ;RETORNA
EJEMPLO1B ENDP            ;FIN DE PROCEDIMIENTO

;          ##############################################
;          # PROCEDIMIENTO POSICIONAR CURSOR 'POSICION' #
;          ##############################################

POSICION PROC             ;PROCEDIMIENTO 'POSICION'
          PUSH DX          ;GUARDO REGISTRO DX EN PILA
          PUSH BX           ;GUARDO REGISTRO BX
          PUSH AX           ;GUARDO REGISTRO AX


          CMP DL,0          ;COMPARA DL CON 0
          JL FIN            ;SI ES MENOR (LOW) SALTA A 'FIN'
          CMP DL,79         ;COMPARA DL CON 79
          JG FIN            ;SI ES MAYOR (GREATER) SALTA A 'FIN'
          CMP DH,0          ;COMPARA DH CON 0
          JL FIN            ;SI ES MENOR (LOW) SALTA A 'FIN'
          CMP DH,24         ;COMPARA DH CON 24
          JG FIN            ;SI ES MAYOR SALTA A 'FIN'

          MOV BH,0          ;PµGINA 0 DE LA PANTALLA EN MODO TEXTO
          MOV AH,2          ;FUNCION 2 POSICIONAR CURSOR DE LA BIOS
          INT 10H           ;LLAMAR A LA INTERRUPCION BIOS

FIN:      POP AX            ;RECUPERA REGISTROS
          POP BX
          POP DX
          RET
POSICION ENDP

;          ################################################
;          # PROCEDIMIENTO ESCRIBIR CARACTERES EN PANTALLA #
;          ################################################
;
ESCRIBIR PROC             ;PROCEDIMIENTO 'ESCRIBIR'
          PUSH AX           ;GUARDA EN PILA AX
          MOV AH,9          ;AH=9 FUN. NUM. 9 'SALIDA DE CARAC.'
          INT 21H           ;LLAMADA A INTERRUPCION DEL DOS, CON FUNCION 9
          POP AX            ;RECUPERA EL REGISTRO AX
          RET               ;RETORNAR
ESCRIBIR ENDP             ;FIN DE PROCEDIMIENTO ESCRIBIR
CODIGO   ENDS             ;FIN DE SEGMENTO DE CODIGO
          END EJEMPLO1B    ;FIN DE PROGRAMA EJEMPLO
```

Figura 1-2. Programa que escribe una cadena de caracteres en una posición determinada.

1.4. PROGRAMAS PROPUESTOS PARA PROFUNDIZAR EN EL MANEJO DEL COMPILADOR Y DEPURADOR. USO DE INSTRUCCIONES BÁSICAS EN ENSAMBLADOR, MANEJO DE PANTALLA.

1.4.1. *Programa A.1*

Realizar un programa que nos genere en pantalla el menú siguiente:

```
MENU (Introduce opción)

        OPCIÓN 1
        OPCIÓN 2
        OPCIÓN 3

          SALIR
```

Figura 1-3. Menú con tres opciones.

1.4.2. *Programa A.2*

Realizar un programa que nos dibuje en pantalla una ventana. La ventana la realizaremos con el uso de los caracteres ASCII especiales para dibujar marcos (ver Figura 1-5).

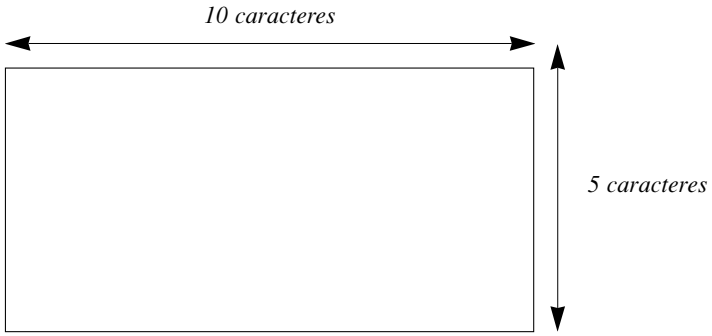


Figura 1-4. Ventana ejemplo con texto en su interior.

1.4.3. *Programa A.3*

Realizar un programa que llene la pantalla de espacios.

1.4.4. *Programa A.4*

Modificar el programa A.2 para que nos ponga la ventana en la posición que queramos de la pantalla.

1.4.5. *Programa A.5*

Realizar un programa que mueva un asterisco por los bordes de la pantalla

1.4.6. *Programa A.6*

Modificar el programa A.5 para que nos mueva un texto por la pantalla de izquierda a derecha y de derecha a izquierda sucesivamente.

1.4.7. *Programa A.7*

Realizar un programa que mueva la ventana de derecha a izquierda y de izquierda a derecha, todo lo rápido que se pueda.

1.4.8. *Programa A.8*

Realizar un programa que llene la pantalla con todos los códigos ASCII.

1.4.9. *Programa A.9*

Realizar un programa que mueva un asterisco en pantalla en diagonal y "rebote" en los bordes de la pantalla.

1.4.10. *Programa A.10*

Realizar un programa que mueva un texto en pantalla en diagonal y "rebote" en los bordes de la pantalla.

850 Multilingüe (Latín 1)

0		32		64	@	96	`	128	Ç	160	á	192	└	224	Ó
1	☺	33	!	65	A	97	a	129	ü	161	í	193	┴	225	ß
2	☻	34	"	66	B	98	b	130	é	162	ó	194	┬	226	Ô
3	♥	35	#	67	C	99	c	131	â	163	ú	195	├	227	Ò
4	♦	36	$	68	D	100	d	132	ä	164	ñ	196	─	228	õ
5	♣	37	%	69	E	101	e	133	à	165	Ñ	197	┼	229	Õ
6	♠	38	&	70	F	102	f	134	å	166	ª	198	ã	230	µ
7	•	39	'	71	G	103	g	135	ç	167	º	199	Ã	231	þ
8	◘	40	(	72	H	104	h	136	ê	168	¿	200	╚	232	Þ
9	○	41	)	73	I	105	i	137	ë	169	®	201	╔	233	Ú
10	◙	42	*	74	J	106	j	138	è	170	¬	202	╩	234	Û
11	♂	43	+	75	K	107	k	139	ï	171	½	203	╦	235	Ù
12	♀	44	,	76	L	108	l	140	î	172	¼	204	╠	236	ý
13	♪	45	-	77	M	109	m	141	ì	173	¡	205	═	237	Ý
14	♫	46	.	78	N	110	n	142	Ä	174	«	206	╬	238	¯
15	☼	47	/	79	O	111	o	143	Å	175	»	207	¤	239	´
16	►	48	0	80	P	112	p	144	É	176	░	208	ð	240	
17	◄	49	1	81	Q	113	q	145	æ	177	▒	209	Ð	241	±
18	↕	50	2	82	R	114	r	146	Æ	178	▓	210	Ê	242	‗
19	‼	51	3	83	S	115	s	147	ô	179	│	211	Ë	243	¾
20	¶	52	4	84	T	116	t	148	ö	180	┤	212	È	244	¶
21	§	53	5	85	U	117	u	149	ò	181	Á	213	ı	245	§
22	▬	54	6	86	V	118	v	150	û	182	Â	214	Í	246	÷
23	↨	55	7	87	W	119	w	151	ù	183	À	215	Î	247	¸
24	↑	56	8	88	X	120	x	152	ÿ	184	©	216	Ï	248	°
25	↓	57	9	89	Y	121	y	153	Ö	185	╣	217	┘	249	¨
26	→	58	:	90	Z	122	z	154	Ü	186	║	218	┌	250	·
27	←	59	;	91	[	123	{	155	ø	187	╗	219	█	251	¹
28	∟	60	<	92	\	124	¦	156	£	188	╝	220	▄	252	³
29	↔	61	=	93	]	125	}	157	Ø	189	¢	221	¦	253	²
30	▲	62	>	94	^	126	~	158	×	190	¥	222	Ì	254	■
31	▼	63	?	95	_	127	⌂	159	ƒ	191	┐	223	▀	255	

Figura 1-5. Tabla de códigos 850 de MSDOS para lenguaje Multilingüe
(Esta tabla junto con la 437 son las más utilizadas del MSDOS en España).

Práctica 2

2. USO DE MACROS

Una macro es una serie de funciones reunidas bajo un nombre y que tiene unos parámetros de salida y de entrada. El uso de la macro facilita el no tener que repetir la introducción de una serie de instrucciones que se repiten en diversas partes del programa. Usando la macro, **EL ENSAMBLADOR A LA HORA DE ENSAMBLAR, SUSTITUYE LA PALABRA DE LA MACRO POR EL CÓDIGO DE ÉSTA EN TODOS LOS SITIOS DONDE APARECE**.

```
; MACRO PARA ESCRIBIR EN UNA POSICION DETERMINADA
; ENTRADAS:
;       POSICION X,Y
;       TEXTO=ETIQUETA DEL TEXTO A IMPRIMIR

PONTEXTO    MACRO X,Y,TEXTO
            MOV DL,X
            MOV DH,Y
            MOV AH,2
            MOV BH,0
            INT 10H         ;LLAMAMOS A LA BIOS PARA SITUAR EL CURSOR EN (X,Y)
            LEA DX,TEXTO    ;MUEVE DESPLAZAMIENTO A DX
            MOV AH,9
            INT 21H         ;LLAMAMOS AL DOS PARA IMPRIMIR EL TEXTO
            ENDM
```

Figura 2-1. Macro para imprimir un texto en una posición determinada

```
; MACRO PARA REALIZAR UN BUCLE DE PAUSA
; ENTRADAS:
;           NUMBUCLE1, NUMBUCLE2=VALORES DE CONTEO DE LOS DOS BUCLES

BUCLE       MACRO NUMBUCLE1, NUMBUCLE2
            LOCAL BUC1, BUC2

            MOV CX, NUMBUCLE1
BUC2:       PUSH CX
            MOV CX, NUMBUCLE2
BUC1:       NOP
            LOOP BUC1
            POP CX
            LOOP BUC2
            ENDM
```

Figura 2-2. Macro para realizar un bucle de pausa

1.1. COMPILACIÓN Y EJECUCIÓN PASO A PASO DE UN PROGRAMA EJEMPLO.

Comprender el funcionamiento del programa de la Figura 2-3, ensamblarlo y ejecutarlo paso a paso.

```
;PROGRAMA EJEMPLO: PRACT2A.ASM
;          MUEVE UNA VENTANA CON TEXTO DE IZQ-DER,DER-IZQ

; CONSTANTES
TIEMP1  EQU 100
TIEMP2  EQU 1000


; MACRO PARA ESCRIBIR EN UNA POSICIÓN DETERMINADA

PONTEXTO MACRO X,Y,TEXTO
          PUSH BX
          MOV DL,X
          MOV DH,Y
          CALL POSICION    ;SUBRUTINA PARA POSICIONAR EL CURSOR
          LEA DX,TEXTO     ;MUEVE DESPLAZAMIENTO A DX
          CALL ESCRIBIR    ;SUBRUTINA DE ESCRIBIR TEXTO
          POP BX
          ENDM

BUCLE     MACRO NUMBUCLE1, NUMBUCLE2
          LOCAL BUC1,BUC2
          MOV CX,NUMBUCLE1
BUC1:     PUSH CX
          MOV CX,NUMBUCLE2
BUC2:     NOP
          LOOP BUC2
          POP CX
          LOOP BUC1
          ENDM
;
;PROGRAMA EJEMPLO2: ESCRIBIR EN UNA POSICION DE LA PANTALLA
;-------------------------------------------------------------
;
CR          EQU  13        ;Retorno de carro
LF          EQU  10        ;Salto de línea
POSX        EQU  5         ;Posicion X (0 A 79)
POSY        EQU  5         ;Posicion Y (0 A 24)
;
;-----------------------------------------------------
;
;Segmento de Datos
;-----------------
;
DATOS SEGMENT          ;Comienzo segmento DATOS
TEXT1 DB '             $'
TEXT2 DB '             $'
TEXT3 DB '    HOLA     $'
TEXT4 DB '             $'
TEXT5 DB '             $'   ;TEXTO A IMPRIMIR
VENTX DB 5
```

```
VENTY DB 5
DATOS ENDS
;
;-------------------------------------------------
;
;Segmento de Pila
;----------------
;
PILA SEGMENT STACK         ;Comienzo segmento PILA
     DB 128 DUP('PILA')    ;Inicialización PILA
PILA ENDS
;
;-------------------------------------------------
;
;Segmento de Código
;------------------
;
CODIGO   SEGMENT       ;Comienzo segmento CODIGO
EJEMPLO  PROC FAR
          ASSUME   CS:CODIGO,DS:DATOS,SS:PILA

          PUSH DS      ;Guarda Segmento en pila (DS:AX dirección retorno)
          SUB AX,AX    ;Borrar registro AX=0
          PUSH AX      ;Guarda en Pila AX (IP=0)
          MOV AX,DATOS ;MUEVE EL SEGMENTO DE DATOS A AX
          MOV DS,AX    ;MUEVE AX A DS

          MOV [VENTX],0
          MOV [VENTY],5

          MOV CX,6     ;NÚMERO DE VECES QUE LA VENTANA
MOVER:    PUSH CX      ;SE MUEVE DE IZQ-DER,DER-IZQ

IZQUIER:
          CALL PONE_VENT

          INC [VENTX]
          CMP [VENTX],60
          JNZ IZQUIER
DEREC:
          CALL PONE_VENT

          DEC [VENTX]
          CMP [VENTX],0
          JNZ DEREC

          POP CX
          DEC CX
          CMP CX,00
          JZ FIN2
          JMP MOVER

FIN2:     RET          ;RETORNA
EJEMPLO   ENDP         ;FIN DE PROCEDIMIENTO
```

```
;          ################################################
;          # PROCEDIMIENTO PONE LA VENTANA EN LA PANTALLA #
;          ################################################

PONE_VENT PROC
           PUSH BX
           MOV BH,[VENTY]
           PONTEXTO [VENTX],BH,TEXT1       ;PONE PRIMERA LINEA
           INC BH
           PONTEXTO [VENTX],BH,TEXT2       ;PONE SEGUNA LINEA
           INC BH
           PONTEXTO [VENTX],BH,TEXT3       ;PONE TERCERA LINEA
           INC BH
           PONTEXTO [VENTX],BH,TEXT4       ;PONE CUARTA LINEA
           INC BH
           PONTEXTO [VENTX],BH,TEXT5       ;PONE QUINTA LINEA

           BUCLE TIEMP1,TIEMP2
           POP BX
           RET
PONE_VENT ENDP

;          ###############################################
;          # PROCEDIMIENTO POSICIONAR CURSOR 'POSICION' #
;          ###############################################

POSICION PROC              ;PROCEDIMIENTO 'POSICION'
          PUSH DX          ;GUARDO REGISTRO DX EN PILA
          PUSH BX          ;GUARDO REGISTRO BX
          PUSH AX          ;GUARDO REGISTRO AX

;          MOV DL,POSX     ;MUEVO EN DL LA POSICION X
;          MOV DH,POSY     ;MUEVO EN DH LA POSICION Y

          CMP DL,0         ;COMPARA DL CON 0
          JL FIN           ;SI ES MENOR (LOW) SALTA A 'FIN'
          CMP DL,79        ;COMPARA DL CON 79
          JG FIN           ;SI ES MAYOR (GREATER) SALTA A 'FIN'
          CMP DH,0         ;COMPARA DH CON 0
          JL FIN           ;SI ES MENOR (LOW) SALTA A 'FIN'
          CMP DH,24        ;COMPARA DH CON 24
          JG FIN           ;SI ES MAYOR SALTA A 'FIN'

          MOV BH,0         ;PÁGINA 0 DE LA PANTALLA EN MODO TEXTO
          MOV AH,2         ;FUNCION 2 POSICIONAR CURSOR DE LA BIOS
          INT 10H          ;LLAMAR A LA INTERRUPCIÓN BIOS

FIN:      POP AX           ;RECUPERA REGISTROS
          POP BX
          POP DX
          RET
POSICION ENDP

;          ################################################
;          # PROCEDIMIENTO ESCRIBIR CARACTERES CON BIOS #
;          ################################################
```

```
ESCRIBIR    PROC            ;PROCEDIMIENTO 'ESCRIBIR' USANDO LA BIOS

            PUSH AX         ;GUARDA EN PILA AX
            PUSH BX
            MOV BX,DX       ;BX=DX
BUCLE1:     MOV AL,DS:[BX]  ;MUEVE A AL=EL VALOR DE LA DIRECCION DS:DX
            CMP AL,'$'      ;COMPARA AL CON EL SIMBOLO '$'
            JE SALIDA       ;SI ES IGUAL SALTA A 'SALIDA'

            PUSH BX         ;GUARDA BX
            MOV BX,0        ;PAGINA 0
            MOV AH,0EH      ;FUNCION IMPRIMIR CARACTER DE LA BIOS
            INT 10H         ;LLAMADA A LA BIOS
            POP BX

            INC BX          ;INCREMENTA BX, SIGUIENTE CARµCTER
            JMP BUCLE1

SALIDA:     POP BX          ;RECUPERA EL REGISTRO BX
            POP AX          ;RECUPERA EL REGISTRO AX
            RET             ;RETORNAR
ESCRIBIR    ENDP            ;FIN DE PROCEDIMIENTO ESCRIBIR
CODIGO      ENDS            ;FIN DE SEGMENTO DE CODIGO
            END EJEMPLO     ;FIN DE PROGRAMA EJEMPLO
```

Figura 2-3. Programa ejemplo que dibuja una ventana y la desliza en pantalla de forma horizontal.

2.2. Entrada de datos por teclado. Realización de un INPUT mediante las funciones del DOS.

Comprender el funcionamiento del programa de la Figura 2-4.

```
;
;PROGRAMA EJEMPLO: EJEMPLO2B.ASM
;REALIZA UN INPUT EN PANTALLA MEDIANTE LA FUNCIÓN 0Ah del DOS
;-------------------------------------------------
CR     EQU   13         ;Retorno de carro
LF     EQU   10         ;Salto de línea
;
;#################################################
;
;Segmento de Datos
;-----------------
;
DATOS SEGMENT          ;Comienzo segmento DATOS
CADENA DB 53 DUP('x')     ;DEFINE 50 CARACTERES DE ENTRADA + 3 NECESARIOS
        DB '$'            ;PARA LA RUTINA INPUT DEL DOS
TEXTO1 DB 'INTRODUCE CADENA:','$'
CAMBIF DB CR,LF,'$'
DATOS ENDS
;
;#################################################
;
;Segmento de Pila
;----------------
```

```
;
PILA SEGMENT STACK          ;Comienzo segmento PILA
     DB 128 DUP('PILA')     ;Inicialización PILA
PILA ENDS


;
;          ########################################
;          #     MACRO REALIZA UN INPUT           #
;          ########################################

INPUT      MACRO DESTINO, NUM_MAX_CARAC
           LEA DX,DESTINO ;COGE EL OFFSET DEL DESTINO
           MOV AL,NUM_MAX_CARAC
           MOV [DESTINO],AL     ;PRIMER BYTE CON EL NÚMERO MÁXIMO DE
CARAC.
           MOV AH,0AH           ;FUNCION DEL DOS DE ENTRADA DE CARACTERES
           INT 21H
           MOV AL,[DESTINO+1]       ;COGEMOS EL NUM. CARACTERES METIDOS
           MOV AH,0                 ;PARTE ALTA DE AH CON CERO
           MOV DI,AX                ;DI=VALOR DE AL
           MOV AL,'$'
           MOV DESTINO+2[DI],AL     ;PONEMOS AL FINAL UN '$'
           ENDM
;
;          ########################################
;          #     MACRO REALIZA UN PRINT           #
;          ########################################

PRINT      MACRO FUENTE
           LEA DX,FUENTE    ;COGE EL OFFSET DE FUENTE
           MOV AH,09H       ;AH=9  FUNCION NUMERO 9 'SALIDA DE CARACTERES'
           INT 21H          ;LLAMADA A INTERRUPCION DEL DOS, CON FUNCION 9
           ENDM

;-----------------------------------------------
;
;          ########################################
;          #     PROCEDIMIENTO INICIAL            #
;          ########################################
;
;Segmento de Código
;------------------
;
CODIGO    SEGMENT       ;Comienzo segmento CODIGO
EJEMPLO   PROC FAR
          ASSUME   CS:CODIGO,DS:DATOS,SS:PILA
;
          PUSH DS       ;Guarda Segmento en pila (DS:AX dirección retorno)
          SUB AX,AX     ;Borrar registro AX=0
          PUSH AX       ;Guarda en Pila AX (IP=0)
;
          MOV AX,DATOS      ;AX=DATOS (SEGMENTO DE DIRECCION DATOS)
          MOV DS,AX         ;DS=AX
;
          PRINT TEXTO1
          INPUT CADENA,50       ;HACE UN INPUT EN CADENA
```

```
          PRINT CAMBIF
          PRINT CADENA+2        ;IMPRIME CADENA
          PRINT CAMBIF
          RET                ;RETORNA
EJEMPLO   ENDP              ;FIN DE PROCEDIMIENTO

CODIGO    ENDS              ;FIN DE SEGMENTO DE CODIGO
          END EJEMPLO
```

Figura 2-4. Programa ejemplo que realiza un INPUT mediante la función 0Ah del DOS.

2.3. Programas propuestos para profundizar en el manejo de las macros. Uso de instrucciones básicas en ensamblador. Realización de la entrada y salida de datos.

2.3.1. *Programa B.1*

Realizar un programa que nos pida el Nombre, DNI, Dirección y Edad; y nos lo visualice en pantalla de la siguiente forma:

```
       NOMBRE: --------------------------  EDAD: ----
       DIRECCIÓN: -----------------------------------
       NIF: -------------------
```

Figura 2-5. Salida del Programa B.1.

2.3.2. *Programa B.2*

Realizar una macro que nos limpie la pantalla.

2.3.3. *Programa B.3*

Realizar una función que pida que se introduzca una cadena de caracteres y nos devuelva el número de vocales y de consonantes. Además, nos devolverá el número de palabras que hay dentro (se considera el espacio, la coma, el punto, el punto y coma, como separadores de palabras).

2.3.4. *Programa B.4*

Realizar una función que visualice en pantalla un contador que vaya incrementándose, lo más rápidamente posible, desde el 0000 hasta el 9999.

2.3.5. *Programa B.5*

Realizar un programa que pida que se le introduzca una cadena de caracteres y nos la escriba al revés.

2.3.6. *Programa B.6*

Realizar una subrutina que pida una cadena y la imprima en pantalla en vertical.

2.3.7. *Programa B.7*

Realizar una macro con los siguientes parámetros de entrada:

```
VENTANA        MACRO X, Y, ANCH, ALTU, TEXTO
               ----
               ----
               ENDM
```

Figura 2-6. Parámetros de entrada de una macro que dibuja una ventana con sus dimensiones definidas.

que dibuje una ventana de *ANCH* caracteres de ancho y *ALTU* caracteres de altura; en la posición *X,Y*; y con un texto centrado en la misma.

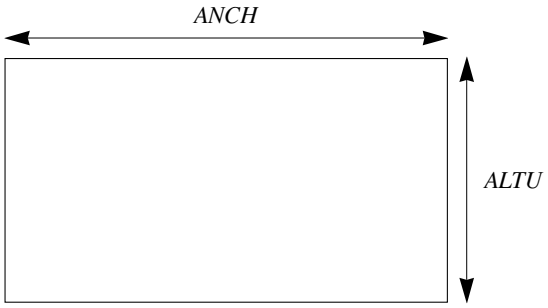


Figura 2-7. Ventana de anchura y altura variable

2.3.8. *Programa B.8*

Desarrollar un programa que **rellene** la pantalla con un asterisco "*" en espiral según las agujas del reloj, desde la esquina superior izquierda hasta el centro de la pantalla y que después **borre** la pantalla moviéndose en sentido contrario. Será necesario ralentizar el movimiento del cursor.

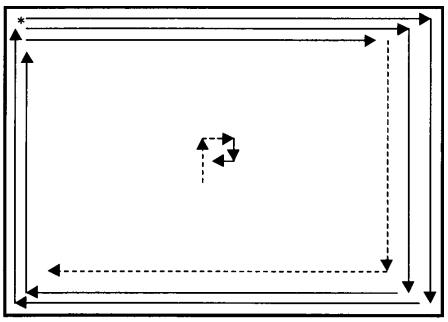


Figura 2-8. Trayectoria del cursor.

2.3.9. *Programa B.9*

Realizar un programa, usando la macro del programa B.7, que dibuje una ventana que va aumentando desde el centro de la pantalla hasta llenarla completamente.

2.3.10. *Programa B.10*

Realizar un programa que nos pida que le introduzcamos una cadena, y nos visualice cada palabra por separado y en orden alfabético.

PRÁCTICA 3

3. Manejo del Teclado

3.1. Funciones BIOS asociadas

Cada vez que se pulsa una tecla, se genera una interrupción Hardware (la interrupción número 9) que ejecuta una rutina de interrupciones que se encarga de leer la tecla pulsada y almacenarla en un *buffer*. Esta rutina almacena dos bytes: El primero es el código de la tecla (ver Figura 3-3) y el segundo el código ASCII de ésta (si tiene).

BIT	KB_FLAG	KB_FLAG_1
	Dirección BIOS 40h:17h = 417h	Dirección BIOS 40h:18h = 418h
7	Insert (1=Activo, 0=Inactivo)	Insert (1=Pulsado, 0=No Pulsado)
6	Caps-Lock (1=Activo, 0=Inactivo)	Caps-Lock (1=Pulsado, 0=No Pulsado)
5	Num-Lock (1=Activo, 0=Inactivo)	Num-Lock (1=Pulsado, 0=No Pulsado)
4	Scroll-Lock (1=Activo, 0=Inactivo)	Scroll-Lock (1=Pulsado, 0=No Pulsado)
3	Alt (1=Pulsado, 0=No Pulsado)	Ctrl Num-Lock (1=Pulsado, 0=No Pulsado)
2	Ctrl. (1=Pulsado, 0=No Pulsado)	No se usa
1	May. Izquierda (1=Pulsado, 0=No Pulsado)	No se usa
0	May. Derecha (1=Pulsado, 0=No Pulsado)	No se usa

Figura 3-1. Variables de la BIOS asociadas al estado de ciertas teclas especiales.

Las funciones BIOS de uso de teclado, corresponden a la interrupción 16H de la BIOS y son:

- **Función AH=00h**: Lee del *buffer* de teclado los códigos asociados a una tecla o combinación de teclas y avanza el puntero del *buffer* al carácter siguiente. Si el *buffer* está vacío, espera que se pulse una tecla. Devuelve:

  En AH = Código de la tecla pulsada

  En AL = Código ASCII del carácter.

- **Función AH=01h**: Devuelve el estado del *buffer* de teclado.

  Si el *buffer* está vacío, ZF=1.

  Si no está vacío:

  ZF = 0

  AH = Código de la tecla

  AL = Código del carácter

Nota: Esta rutina no avanza el puntero de la tecla leída, de forma que si se pulsan varias teclas, sólo nos devolverá el código de la primera tecla.

- **Función AH=02h**: Devuelve en AL el byte de estado del teclado (KB_FLAG). Esta función nos permite conocer el estado de teclas especiales como el *Bloqueo de Mayúsculas*, *Insertar, Num-Lock, etc* (ver la Figura 3-1).

Existen otras rutinas de lectura de teclado del DOS (interrupción 21h), que es conveniente conocer.

3.2. Programa ejemplo. Ejecutar un programa mientras no se pulse la tecla ESCAPE

Muchas veces es interesante dentro de un bucle, tener una subrutina que verifique la pulsación de una tecla pero sin parar la ejecución del mismo. Esto nos permite poder salir de un bucle infinito cuando queramos, sin tener que resetear el ordenador. La rutina estudia el estado del buffer de teclado. Si éste está vacío o se ha pulsado una tecla distinta de "ESCAPE" devuelve el flag de carry a cero (CF=0), si se ha pulsado la tecla "ESCAPE" (código 01h) devuelve el carry a uno (CF=1).

```
; ################################################################
; #        TECLA:          FUNCION QUE MIRA SI SE PULSA ESC          #
; ################################################################

TECLA      PROC
           MOV AH,01        ; FUNCION 01H, MIRA ESTADO DEL BUFFER
           INT 16H          ; ¿VACIO?  ZF=1 SI, ZF=0 NO
           JZ SALTEC        ; SI NO SE HA PULSADO TECLA SALTA A "SALTEC"
           XOR AX,AX        ; FUNCION AH=0
           INT 16H          ; COGE EL CARÁCTER PULSADO
           CMP AH,01        ; ES LA TECLA ESCAPE?
           JNE SALTEC       ; SI NO ES LA TECLA "esc" SALTA A "SALTEC"
           STC              ; PON EL FLAG DE CARRY A UNO CF=1
           RET              ; VUELVE
SALTEC:    CLC              ; PON EL FLAG DE CARRY A CERO CF=0
           RET              ; VUELVE
TECLA      ENDP
```

Figura 3-2. Programa que devuelve CF=1 si se pulsa la tecla 'Escape', en caso contrario devuelve CF=0.

Número	Tecla	Normal	Mayúsculas	Control.	Alt	Alt-Ctrl
1	Esc	01 1B	01 1B	01 13		
2	1 ¡	02 31	02 AD		78 00	78 00
3	2 ¿	03 32	03 AS	03 00	79 00	03 40
4	3 #	04 33	04 23		7A 00	7A 00
5	4 $	05 34	05 24		7B 00	7B 00
6	5 %	06 35	06 25		7C 00	7C 00
7	6 /	07 36	07 2F	07 1E	7D 00	7D 00
8	7 &	08 37	08 26		7E 00	7E 00

Número	Tecla	Normal	Mayúsculas	Control.	Alt	Alt-Ctrl
9	8 *	09 38	09 2A		7F 00	7F 00
10	9 (	0A 39	0A 28		80 00	80 00
11	0)	0B 30	0B 29		81 00	81 00
12	- _	0C 2D	0C 5F	0C IF	82 00	82 00
13	= +	0D 3D	0D 2B		83 00	83 00
14	Retroceso	0E 08	0E 08	0E 7F		
15	Tabulador	0F 09	OF 00			
16	Q	10 71	10 71	10 11	10 00	10 00
17	W	11 77	11 77	11 17	11 00	11 00
18	E	12 65	12 65	12 05	12 00	12 00
19	R	13 72	13 72	13 52	13 00	13 00
20	T	14 74	14 74	14 54	14 00	14 00
21	Y	15 79	15 79	15 59	15 00	15 00
22	U	16 75	16 75	16 55	16 00	16 00
23	I	17 69	17 69	17 49	17 00	17 00
24	O	18 6F	18 6F	18 4F	18 00	18 00
25	P	19 70	19 70	19 50	19 00	19 00
26	á ä	1A ..	1A FE			1A 5B
27	à â	1B 60	1B 5E			1B 5D
28	Enter	1C 0D	1C 0D	IC OA		
29	Ctrl	1D ..				
30	A	1E 61	1E 61	1E 01	1E 00	1E 00
31	S	1F 73	1F 73	1F 13	1F 00	1F 00
32	D	20 64	20 64	20 04	20 00	20 00
33	F	21 66	21 66	21 06	21 00	21 00
34	G	22 67	22 67	22 07	22 00	22 00
35	H	23 68	23 68	23 08	23 00	23 00
36	J	24 6A	24 6A	24 0A	24 00	24 00
37	K	25 6B	25 6B	25 0B	25 00	25 00
38	L	26 6C	26 6C	26 0C	26 00	26 00
39	Ñ	27 A4	27 A4			
40	; :	28 3B	28 3A			
41	Ç	29 87	29 87			

Número	Tecla	Normal	Mayúsculas	Control.	Alt	Alt-Ctrl
42	Mayúscula	2A ..				
43	< >	2B 3C	2B 3E	213 IC		2B 5C
44	Z	2C 7A	2C 7A	2C IA	2C 00	2C 00
45	X	2D 78	2D 78	2D 18	2D 00	2D 00
46	C	2E 63	2E 63	2E 03	2E 00	2E 00
47	V	2F 76	2F 76	2F 16	2F 00	2F 00
48	B	30 62	30 62	30 02	30 00	30 00
49	N	31 6E	31 6E	31 0E	31 00	31 00
50	M	32 6D	32 6D	32 0D	32 00	32 00
51	, ?	33 2C	33 317			
52	. !	34 2E	34 21			
53	‘ “	35 27	35 22			
54	Mayúscula	36 ..				
55	^ PrtSc	37 5E		72 00		
56	Alt	38 ..				
57	Espacio	39 20	39 20	39 20	39 20	
58	Caps Lock	3A ..				
59	F1	3B 00	54 00	5E 00	68 00	68 00
60	F2	3C 00	55 00	5F 00	69 00	69 00
61	F3	3D 00	56 00	60 00	6A 00	6A 00
62	F4	3E 00	57 00	61 00	6B 00	6B 00
63	F5	3F 00	58 00	62 00	6C 00	6C 00
64	F6	40 00	59 00	63 00	6D 00	6D 00
65	F7	41 00	5A 00	64 00	6E 00	6E 00
66	F8	42 00	5B 00	65 00	6F 00	6F 00
67	F9	43 00	5C 00	66 00	70 00	70 00
68	F10	44 00	5D 00	67 00	71 00	71 00
69	Num Lock	45 ..				37 00
70	Scroll Lock	46 ..				38 00
71	Home 7	47 37	47 37	77 00	00 07	00 07
72	Flecha arr. 8	48 38	48 38		00 08	00 08
73	PgUp 9	49 39	49 39	84 00	00 09	00 09
74	–	4A 2D	4A 2D			

Número	Tecla	Normal	Mayúsculas	Control.	Alt	Alt-Ctrl
75	Flecha izq. 4	4B 34	4B 34	73 00	00 04	00 04
76	5	4C 35	4C 35		00 05	00 05
77	Flecha der. 6	4D 36	4D 36	74 00	00 06	00 06
78	+	4E 2B	4E 2B			
79	End 1	4F 31	4F 31	75 00	00 01	00 01
80	Flecha aba. 2	50 32	50 32		00 02	00 02
81	PgDn 3	51 33	51 33	76 00	00 03	00 03
82	Ins 0	52 30	52 30			
83	Del	53 2E	53 2E			Carga Sistema

Figura 3-3. Tabla de códigos de teclas en el teclado español.
El primer número de cada columna es el valor de la tecla y el segundo su código ASCII.
(Extraída del libro titulado "8088-8086/8087 Programación ENSAMBLADOR en entorno MS-DOS" de la Editorial ANAYA Multimedia escrita por Miguel Angel Rodríguez-Roselló).

3.3. Transformación de números "Binario-ASCII", "ASCII-Binario".

Muchas veces nos encontramos con uno de los siguientes problemas:

1. Existen casos donde **es necesario coger el valor binario que tenemos en un registro y convertirlo a caracteres ASCII**. Ejemplos típicos son estos: se tiene un valor en un registro o posición de memoria (se designará como número en binario ya que está almacenado de esa forma en la memoria) y se quiere visualizar en una pantalla; o dada una tarjeta de control con un microcontrolador y unos pocos displays se desea visualizar ciertos parámetros; o el envío de un número carácter a carácter por el puerto serie, etc.
2. Un ejemplo muy típico en el que es necesario realizar el proceso inverso, puede aparecer cuando en un teclado numérico se introduce números carácter a carácter y hay que pasarlo a un registro para operar con él. En este caso se habla **de conversión ASCII a binario.**

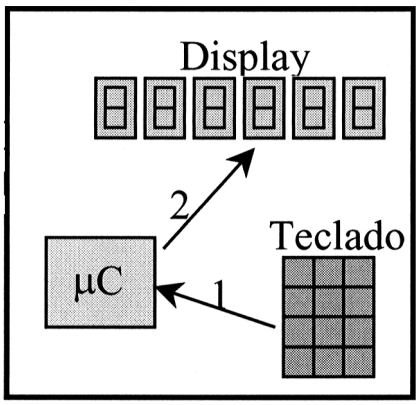


Figura 3-4. Tarjeta con un microcontrolador donde es necesario convertir de ASCII a binario los números introducidos (flecha 1) para poder operar con ellos y después los números de binario a ASCII para poder visualizarlos en el display (flecha 2).

3.3.1. *Conversión binario a ASCII*

Vamos a ver dos métodos para convertir un número que tenemos almacenado en un registro a ASCII.

El primer método consiste en dividir por 10 elevado al número de dígitos máximo posible del número a convertir y seguir con el resto dividiéndolo como se ve en la Figura 3-5. La ventaja que tiene este método es que el cociente de la división nos va dando el dígito más significativo.

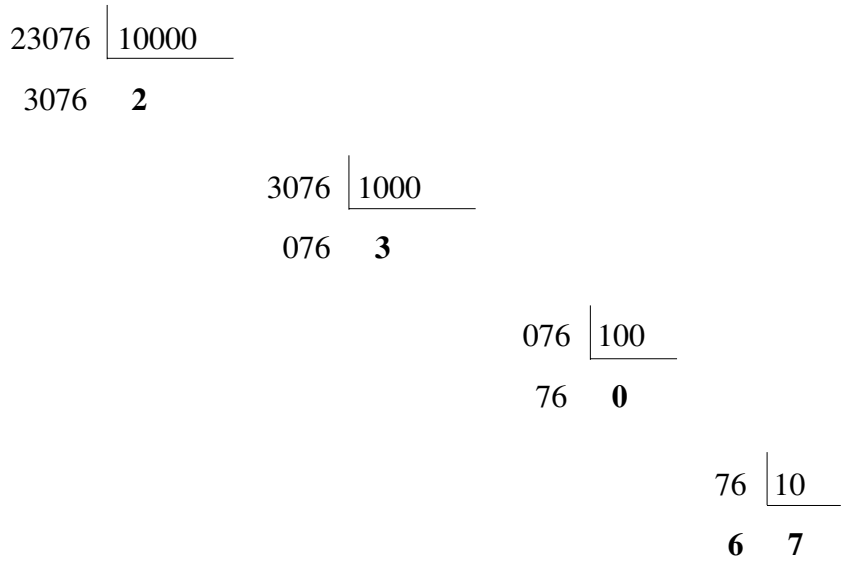


Figura 3-5. División del número por 10000, 1000, 100, 10.

El segundo método consiste en dividir por 10 el cociente contínuamente como en la Figura 3-6. La desventaja que tiene este método es que los dígitos los obtenemos al revés, es decir, empezando por el menos significativo. Tiene la ventaja de que es más fácil de implementar en un bucle ya que siempre se divide por el mismo número.

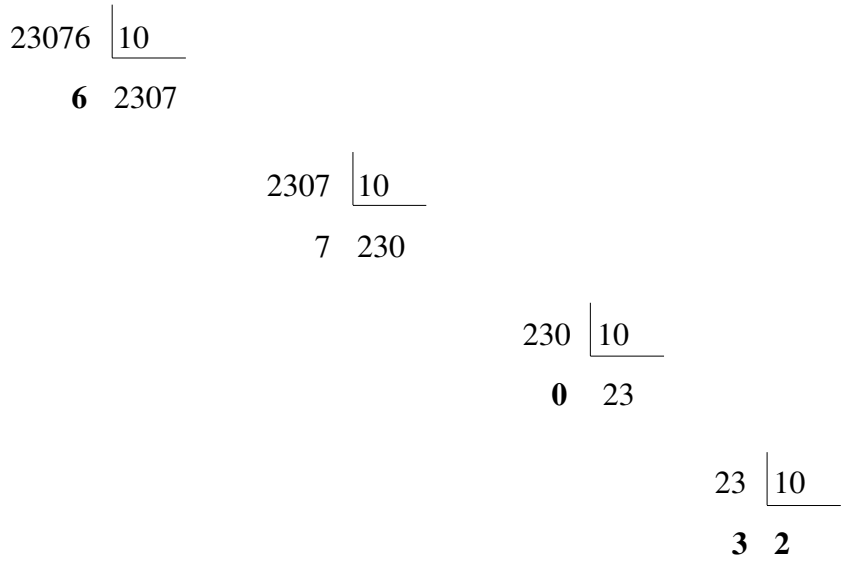


Figura 3-6. División por 10.

Una vez tenemos los números desempaquetados (es decir, cada dígito en un byte), le sumamos a cada uno de ellos, antes de visualizarlo, el código ASCII del número "0" (valor 48) para obtener su correspondiente código ASCII. Por ejemplo el código ASCII del número '3' es 48+3=51.

3.3.2. *Conversión ASCII a binario*

Supongamos que tenemos el número 23076 formado por los códigos ASCII: "2", "3", "0", "7", "6". El algoritmo de conversión más práctico es el siguiente:

1. Inicializamos una variable donde vamos a almacenar el resultado. Ejemplo: SUMA=0.
2. Cogemos el primer dígito (en este caso el "2"). Ejemplo: VALOR= "2" = 50.
3. Multiplicamos SUMA por 10. Ejemplo: SUMA*10=0
4. Convertimos el dígito de ASCII a número desempaquetado restándole 48 (valor del "0"). Ejemplo: VALOR-48=2
5. Se lo sumamos a la variable SUMA. Ejemplo: SUMA=SUMA+VALOR=2
6. Obtenemos el siguiente dígito y repetimos los pasos 3, 4 y 5; hasta que terminamos con todos los dígitos. Es decir, multiplicamos SUMA por 10, (SUMA=20), cogemos el segundo ("3") le restamos 48 a su valor ASCII (51-48=3) y se lo sumamos a la variable SUMA (SUMA=20+3=23). Y así, con todos los dígitos...

3.4. Programas propuestos para profundizar en el: Manejo del teclado y la transformación de números "Binario-ASCII" y "ASCII-Binario"

3.4.1. *Programa C.1*

Añadir al programa B.4 de la práctica anterior, el código necesario para que la ejecución se pueda interrumpir cuando se pulse la tecla 'ESC'.

3.4.2. *Programa C.2*

Realizar un programa que sitúe un asterisco en el centro de la pantalla y permita moverlo con los cursores.

3.4.3. *Programa C.3*

Desarrollar un programa que visualice en pantalla lo más rápidamente posible un contador de 10 dígitos que vaya incrementándose desde el 0000000000 hasta el 9999999999.

El programa se puede interrumpir en cualquier momento con la tecla 'ESCAPE' y nos dirá cuanto tiempo en segundos y décimas de segundo ha tardado el ordenador en incrementar ese contador desde 0 hasta el número que hemos parado.

Pista para calcular el tiempo: Se usará la interrupción 1Ah de la BIOS que incrementa un contador 18.2 veces por segundo.

```
INTERRUPCIÓN    1Ah (BIOS)
FUNCIÓN         00h
ENTRADA:        AH=00h
SALIDA:         CX=Hi-Word del contador de Tiempo
                DX=Lo-Word del contador de Tiempo
                AL=0 Si desde la última lectura del contador han
                transcurrido menos de 24 Horas
```

Figura 3-7. Manejo de la interrupción 1Ah de la BIOS para conteo del tiempo.

3.4.4. *Programa C.4*

Realizar un programa que:

1. Pida que le introduzcamos un número (no mayor de 32000).
2. Multiplique por 5 el número introducido.
3. Después divida el resultado por 2.
4. El resultado de la división lo imprimirá en pantalla.

3.4.5. *Programa C.5*

Realizar un programa que a medida que pulsemos teclas nos saque por pantalla en hexadecimal el código de la tecla pulsada, su código ASCII correspondiente (si lo tiene) y el carácter ASCII (si lo tiene).

3.4.6. *Programa C.6*

Realizar un programa que tenga el siguiente menú:

```
              Menu

          1.- Binario-Decimal
          2.- Decimal-Binario
          3.- Hexadecimal-Binario
          4.- Binario-Hexadecimal

          0.- Salir
```

Figura 3-8. Salida del programa C.6.

De forma que si pulsamos el ‘1’ nos pedirá que le introduzcamos un número binario y nos lo convertirá en decimal; si pulsamos el ‘2’ nos pedirá que le introduzcamos un número en decimal y nos lo convertirán en binario; igualmente con las opciones ‘3’ y ‘4’.

3.4.7. *Programa C.7*

El programa nos pedirá que introduzcamos una dirección en hexadecimal, nos representará en pantalla el contenido de la memoria en esa dirección de dos formas: en Hexadecimal y en ASCII (los símbolos que pueda).

Ejemplo:

```
Introduce dirección inicial: 0C75:0100
ASCII           DIRECCIÓN       +0 +1 +2 +3 +4 +5 +6 +7 +8 +9 +A +B +C +D +E +F
.............t.< 0C75:0100     F9 EB F6 BE 81 00 E8 F8-CC AC E8 FC CC 74 0A 3C
.t.:...u.N..4.d. 0C75:0110     0D 74 06 3A 06 D5 9A 75-F0 4E C3 BF 34 00 64 0C
......<.u.&..... 0C75:0120     FE C9 AC AA FE C1 3C 0D-75 F8 26 88 0E 80 00 C3
.............X. 0C75:0130     C6 06 91 8E 01 BA 93 8E-A3 93 8E C3 B8 00 58 CD
!....X.!.....$.. 0C75:0140     21 8A D8 B8 02 58 CD 21-8A F8 93 D0 C0 24 01 D0
................ 0C75:0150     E4 0A C4 C3 1E E8 E4 FF-8E 1E D1 9A A2 13 06 80
......_..m.....H 0C75:0160     0E 13 06 80 1F E8 5F FF-E8 6D FF C3 A1 1F A1 48
......_......... 0C75:0170     BB 0B 00 F7 E3 BF 5F 9E-8B F7 83 C6 0B 8B C8 FC
```

Figura 3-9. Salida del programa C.7.

Se podrá avanzar y retroceder en la memoria mediante el uso de las teclas “Avance Página” y “Retrocede Página”. Se podrá salir con “Escape”.

Advertencia: Tener cuidado al pasar de un segmento a otro.

PRÁCTICA 4

4. Funciones Externas y Creación de Librerías

4.1. Paso de parámetros a una Subrutina

El paso de argumentos (valores de entrada) a una función, lo podemos realizar de varias formas. Los métodos más útiles son:

1. Usando **Registros**: El programa utiliza el valor de una serie de registro; simplemente hay que poner el valor en estos y llamar a la subrutina.

Ejemplo:

```
                PUSH AX
                MOV AX,0FF0h
                CALL SUBRUTINA      ;LA SUBRUTINA UTILIZA AX
                POP AX
```

Figura 4-1. Ejemplo de paso de datos mediante el uso de registros.

La desventaja mayor es que tenemos que saber qué registros utiliza la función pero, por otro lado, es la más cómoda.

2. Usando **la Memoria**: Se almacena el valor en una posición de memoria y la subrutina recoge el valor de esa dirección. Es poco flexible ya que tenemos que saber dónde está esa posición de memoria.

Ejemplo:

```
    VALOR1:     DW 0
                --
                --
                MOV DX,0FF0h
                MOV [VALOR1],DX
                CALL SUBRUTINA
                --
                --
    SUBRUTINA:  MOV AX,[VALOR1]
```

Figura 4-2. Ejemplo de paso de datos mediante el uso de la memoria.

3. Usando **la Pila**: Se guardan los valores en la pila, en un orden determinado. La subrutina los recogerá de ésta. La ventaja principal es que no hay que saber que registros utiliza, sólo qué datos hay que introducir y en qué orden. Además el número de parámetros de entrada, mientras no se llene la pila, puede ser elevado.

Ejemplo:

```
PRINCIPAL:
            PUSH AX ;PARÁMETRO1
            PUSH BX ;PARÁMETRO2
            PUSH CX ;PARÁMETRO3
            CALL SUBRUTINA
            POP CX              ;SACAMOS LOS PARÁMETROS DE
            POP BX              ;LA PILA PARA DEJARLA COMO
            POP AX              ;ESTABA.

            --
            --
            PUBLIC SUBRUTINA
SUBRUTINA:  PROC FAR
            PUSH BP             ;GUARDA BP
            MOV BP,SP           ;BP=SP PUNTERO DE LA PILA
            MOV DX,[BP+6]       ;DX=PARAMETRO3
            MOV AX,[BP+8]       ;AX=PARÁMETRO2
            MOV CX,[BP+10]      ;CX=PARÁMETRO1
            --
            POP BP
            RET
```

Figura 4-3. Ejemplo de paso de parámetros mediante el uso de la pila.

Después del PUSH BP, tenemos en la pila:

```
➢ En [SS:BP] el valor que tiene BP al entrar en la subrutina.
➢ En [SS:BP+2] el desplazamiento de la dirección de retorno.
➢ En [SS:BP+4] el segmento de la dirección de retorno
➢ En [SS:BP+6] el último valor metido (PARÁMETRO3)
➢ En [SS:BP+8] el penúltimo valor metido (PARÁMETRO2)
➢ En [SS:BP+10] el primer valor metido (PARÁMETRO1)
```

Figura 4-4. Organización de la pila del programa de la Figura 4-3.

La instrucción:

```
RET num
```

recupera de la pila la dirección de retorno, adelanta el puntero de pila *(SP) num* bytes y retorna. De esta forma podemos eliminar los 'POP' del programa principal. Veamos el ejemplo anterior usando esta forma:

Ejemplo:

```
PRINCIPAL:
            PUSH AX ;PARÁMETRO1
            PUSH BX ;PARÁMETRO2
            PUSH CX ;PARÁMETRO3
            CALL SUBRUTINA
            --  ;no hacemos POP AX, ni POP BX, ni POP CX

            --
            --
```

```
                PUBLIC SUBRUTINA
      SUBRUTINA:  PROC FAR
                  PUSH BP     ;GUARDA     BP
                  MOV BP,SP       ;BP=SP PUNTERO DE LA PILA
                  MOV DX,[BP+6]       ;DX=PARAMETRO3
                  MOV AX,[BP+8]       ;AX=PARÁMETRO2
                  MOV CX,[BP+10]      ;CX=PARÁMETRO1
                  ---
                  POP BP
                  RET 6       ;Recupera la dirección de retorno,
                            ;adelanta la pila 6 bytes y retorna
```

Figura 4-5. Programa ejemplo donde se pasan los parámetros en la pila y se destruyen de ésta al salir del programa.

4.2. ENLAZAR VARIOS OBJS. PROGRAMAS "PRACT4A.ASM" Y "PRACT4B.ASM".

Podemos definir nuestras propias funciones, ensamblarlas por separado y después unirlas mediante el "*linkador*" todas juntas. Cuando se realiza un programa de grandes dimensiones es conveniente tener las funciones en "grupos de subrutinas", cada grupo en un archivo OBJ; de esta forma, si hay que modificar alguna función, solo hay que ensamblar el archivo OBJ correspondiente a la misma.

El procedimiento de ensamblado y "*linkado*" es el siguiente.

1º. Se ensambla cada OBJ por separado. Por ejemplo:
```
TASM PRACT4A.ASM
TASM PRACT4B.ASM
```

2º. Se linkan los OBJs usando el operador **'+'**
```
TLINK PRACT4A.OBJ+PRACT4B.OBJ
```

Se generará un archivo "PRACT4A.EXE".

A la hora de definir una función que se va a llamar desde una función de otro OBJ, se define la primera con la Directiva **PUBLIC**.

4.2.1. *Programa ejemplo PRACT4A.ASM*

```
;
;PROGRAMA EJEMPLO4A:
;
;Segmento de Datos
;-----------------
DATOS SEGMENT         ;Comienzo segmento DATOS
TEXTO1 DB 'EN ESTE EJEMPLO USAMOS DOS ARCHIVOS OBJETO, UNO PRINCIPAL Y
OTRO SECUNDARIO CON LA FUNCIÓN PÚBLICA ESCRIBIR','$'
DATOS ENDS

PILA SEGMENT STACK  ;Comienzo segmento PILA
     DB 128 DUP('PILA')   ;Inicialización PILA
PILA ENDS

;Segmento de Código
;--------
```

```
;
          EXTRN ESCRIBIR:FAR
CODIGO     SEGMENT   ;Comienzo segmento CODIGO
EJEMPLO    PROC FAR
           ASSUME    CS:CODIGO,DS:DATOS,SS:PILA
;
           PUSH DS   ;Guarda Segmento en pila (DS:AX dirección retorno)
           SUB AX,AX ;Borrar registro AX=0
           PUSH AX   ;Guarda en Pila AX (IP=0)
;
           MOV AX,DATOS    ;AX=DATOS (SEGMENTO DE DIRECCION DATOS)
           MOV DS,AX       ;DS=AX
;
           LEA DX,TEXTO1   ;DX=DESPLAZAMIENTO DE TEXTO1
           PUSH DS         ;GUARDAMOS DS
           PUSH DX         ;GUARDAMOS DX (PASAMOS EN PILA LA DIRECCIÓN
                            DEL TEXTO)
           CALL ESCRIBIR   ;SUBRUTINA (TIPO FAR) DE ESCRIBIR TEXTO1
           RET             ;RETORNA
EJEMPLO    ENDP            ;FIN DE PROCEDIMIENTO
CODIGO     ENDS            ;FIN DE SEGMENTO DE CÓDIGO
           END EJEMPLO
```

Figura 4-6. Programa ejemplo Pract4A.asm.

4.2.2. *Programa ejemplo PRACT4B.ASM*

```
; PROGRAMA PRACT4B.ASM
; FUNCIONES DE NUESTRA LIBRERIA


;     ##### OTRO SEGMENTO DE CÓDIGO #####
CODIGO2    SEGMENT    ;Comienzo segmento CODIGO2
;-------------------------------------------------------------------------
;          ######################################
;          #     PROCEDIMIENTO 'ESCRIBIR'         #
;          #     ENTRADA: (EN PILA)                #
;          #              DS:DX=DIRECCION_TEXTO    #
;          ######################################
;
          PUBLIC ESCRIBIR
ESCRIBIR PROC FAR          ;PROCEDIMIENTO 'ESCRIBIR' TIPO FAR
          ASSUME   CS:CODIGO2
          PUSH BP          ;GUARDA BP
          MOV BP,SP        ;BP=SP
          MOV AX,SS:[BP+6] ;COGEMOS EL VALOR DEL DESPLAZAMIENTO DX
          MOV DX,AX
          MOV AX,SS:[BP+8] ;COGEMOS EL VALOR DEL SEGMENTO DS
          MOV DS,AX
          PUSH AX          ;GUARDA EN PILA AX
          MOV AH,9         ;AH=9  FUNCION NUMERO 9 'SALIDA DE CARACTERES'
          INT 21H          ;LLAMADA A INTERRUPCION DEL DOS, CON FUNCION 9
          POP AX           ;RECUPERA EL REGISTRO AX

          POP BP           ;RECUPERA BP
          RET 4            ;RETORNAR
```

```
ESCRIBIR    ENDP        ;FIN DE PROCEDIMIENTO ESCRIBIR
CODIGO2     ENDS
            END
```

Figura 4-7. Programa ejemplo Pract4B.asm.

4.3. Creación de Librerías

El programa "TLIB.EXE" permite al usuario generar librerías formadas por un conjunto de funciones que sólo se unirán al programa principal cuando se usen.

Por ejemplo, si tenemos una librería llamada "P.LIB" formada por tres funciones "FUNCION1, FUNCION2, FUNCION3", y desde un objeto propio "PRACT4A.OBJ" sólo llamamos a "FUNCION1", en el proceso de "*linkado*":

```
TLINK PRACT4A.OBJ+P.LIB ;GENERAMOS PRACT4A.EXE
```

El *"linkador"* sólo unirá al objeto inicial la función *"FUNCION1"*, con el consiguiente ahorro de memoria.

El uso de "TLIB.EXE" es muy sencillo. Para añadir una serie de funciones que se tiene en un objeto llamado "PRACT4B.OBJ" a una librería "P.LIB" se escribe:

```
TLIB P.LIB+PRACT4B.OBJ ;AÑADIR LAS FUNCIONES DE PRACT4B.OBJ
```

Si se quiere sustituir las funciones de "PRACT4.OBJ" que ya están en la librería, se eliminan primero con un menos y se añaden las nuevas después:

```
TLIB P.LIB-PRACT4B.OBJ+PRACT4B.OBJ ;SUSTITUIR FUNCIONES
```

Una vez se tiene la librería creada y actualizada, se usa mediante "TLINK.EXE" como en el ejemplo siguiente:

EJEMPLO:

```
TLIB P.LIB-PRACT4B.OBJ+PRACT4B.OBJ  ;SUSTITUIR OBJETO
TLINK PRACT4A.OBJ+P.LIB             ;Creamos un ejecutable con el
                                    ;objeto PRACT4A.OBJ y las
                                    ;funciones de P.LIB que son
                                    ;llamadas por el mismo
```

Figura 4-8. Ejemplo de creación y uso de una librería de funciones.

Se obtendrá un archivo ejecutable llamado: "PRACT4A.EXE".

4.4. Programas propuestos para profundizar en: Uso y creación de librerías y programas externos.

4.4.1. *Programa D.1*

Realizar una función externa en "PRACT4D.ASM" que nos pida que le introduzcamos por teclado una frase y la almacene en la dirección FAR que previamente le habrá sido introducida en la pila desde una función que la llama (incluida en "PRACT4C.ASM").

4.4.2. *Programa D.2*

Realizar una función que nos imprima en pantalla los números que le son introducidos en la pila de la siguiente forma:

```
PUSH Num1
PUSH Num2
PUSH Num3
PUSH ---
PUSH NumN
PUSH Cantidad de Números Introducidos
CALL PONNUMEROS
POP ---
POP ---
POP ---
---
```

La función la meteremos en una librería propia llamada “MI_LIB.LIB”.

4.4.3. *Programa D.3*

Realizar un programa que nos ordene mediante el método de la burbuja los números que le son introducidos en la pila como en el programa D.2. Esta función nos devolverá en las mismas posiciones de la pila los números ordenados. Se guardará en la librería “MI_LIB.LIB”.

4.4.4. *Programa D.4*

Realizar un programa que tenga como parámetros de entrada (en la pila) el número de filas y columnas de una matriz, devolviendo en la pila una dirección FAR que apunte a un buffer lleno de ceros del tamaño de la matriz (de enteros).

4.4.5. *Programa D.5*

Realizar una función que calcule el número primo número NUM que se le habrá pasado en la pila y nos devuelva en la misma posición de la pila este número primo. Por ejemplo, si se introduce en la pila un 1.000 nos encontrará el número primo 1000(empezando por el 1) y nos lo devolverá en la pila en la misma posición.

4.4.6. *Programa D.6*

Un número de 6 dígitos se modifica de la siguiente forma:

Ejemplo:

```
Número a modificar =                    112233

En ASCII (‘0’ es 48..’9’ es 57)=    49,49,50,50,51,51
```

```
          Modificación=              49    Núm. 1º x 1
                                    490    Núm. 2º x 10
                                   5000    Núm. 3º x 100
                                  50000    Núm. 4º x 1000
                                 510000    Núm. 5º x 10000
                           +    5100000    Núm. 6º x 100000
                                5665539

   Número solución =            5665539
```

Figura 4-9. Método de codificación utilizado.

Realizar un programa que buscando todas las combinaciones posibles, encuentre el número de 6 dígitos que corresponde a la suma cuyo número es **5333428** (Pueden salir varios). Sacar en pantalla el tiempo que le cuesta resolverlo.

Realizar esta misma función para que busque la combinación de 6 dígitos que dé como resultado el número que habrá sido enviado dígito a dígito, desde otra función, usando la pila.

4.4.7. *Programa D.7*

Añadir a la librería "MI_LIB.LIB" las funciones siguientes (los parámetros se pasa en la pila y se devuelven en la misma):

1. Convertir un número decimal a binario.
2. Convertir un número binario a decimal.
3. Convertir un número decimal a hexadecimal
4. Convertir un número hexadecimal a binario
5. Convertir un valor numérico a ASCII
6. Convertir un número ASCII a valor numérico.

PRÁCTICA 5

5. Uso de Instrucciones de Manejo de Cadenas

5.1. Programas “PRACT5A.ASM” y “PRACT5B.ASM”.

Escribir los programas siguientes, que utilizan las instrucciones de manejo de cadenas, y comprender su funcionamiento.

5.1.1. *Programa “PRACT5A.ASM”*

```
DATOS           SEGMENT
TEXTO           DB 512 DUP (?)
CARAC           DB ` ` , `,` , `;` , `.` , `:`

DATOS ENDS

PILA            SEGMENT STACK
                DB 128 DUP (`PILA`)
PILA            ENDS

CODIGO          SEGMENT
EJEMPLO         PROC FAR
                ASSUME CS:CODIGO,DS:DATOS,SS:PILA

                PUSH DS
                SUB AX,AX
                PUSH AX

                MOV AX,DATOS
                MOV DS,AX

                MOV [TEXTO],255
                LEA DX,TEXTO
                MOV AH,0Ah
                INT 21h
                XOR AX,AX
                MOV AL,[TEXTO+1]
                MOV DI,AX
                MOV TEXTO[DI+2],'$'

                PUSH DS
                POP ES
                LEA SI,TEXTO+2
                LEA DI,TEXTO+256

BUC1:           LODSB
                CMP AL,'$'
                JE FIN
```

```
                PUSH DI
                LEA DI,CARAC
                MOV CX,5
                REPNE SCASB
                POP DI
                JE BUC1

                STOSB
                JMP BUC1

FIN:                MOV AL,'$'
                STOSB

                MOV AH,0
                MOV AL,3
                INT 10H

                LEA DX,TEXTO+256
                MOV AH,09h
                INT 21h
                RET
EJEMPLO         ENDP

CODIGO          ENDS
                END EJEMPLO
```

Figura 5-1. Programa PRACT5A.ASM que utiliza las instrucciones de cadena.

5.1.2. *Programa "PRACT5B.ASM"*

```
DATOS           SEGMENT
TEXTO           DB 256 DUP (?)
VOCAL           DB 'AaEeIiOoUu'
NUMVOCAL        DW 0
TEXTO2          DB 13,10,13,10,'NUMERO DE VOCALES:$'
TEXTO3          DB 13,10,'$'
DATOS ENDS

PILA            SEGMENT STACK
                DB 128 DUP ('PILA')
PILA            ENDS

CODIGO          SEGMENT
EJEMPLO         PROC FAR
                ASSUME CS:CODIGO,DS:DATOS,SS:PILA

                PUSH DS
                SUB AX,AX
                PUSH AX

                MOV AX,DATOS
                MOV DS,AX

                MOV [TEXTO],253
                LEA DX,TEXTO
```

```
                MOV AH,0Ah
                INT 21h
                XOR AX,AX
                MOV AL,[TEXTO+1]
                MOV DI,AX
                MOV TEXTO[DI+2],'$'

                PUSH DS
                POP ES
                LEA SI,TEXTO+2
                MOV CX,AX
                CLD

BUC1:           LODSB
                PUSH CX
                LEA DI,VOCAL
                MOV CX,10
                REPNE SCASB
                JNZ BUC2
                INC [NUMVOCAL]
BUC2:           POP CX
                LOOP BUC1


                LEA DX,TEXTO2
                MOV AH,09h
                INT 21h
                MOV AX,[NUMVOCAL]
                MOV BL,10
                DIV BL
                ADD AX,3030H
                MOV DX,AX

                MOV AH,0EH
                MOV AL,DL
                MOV BH,0
                MOV BL,47H
                INT 10H
                MOV AH,0EH
                MOV AL,DH
                MOV BH,0
                MOV BL,47H
                INT 10H
                LEA DX,TEXTO3
                MOV AH,09h
                INT 21h


                RET
EJEMPLO         ENDP

CODIGO          ENDS
                END EJEMPLO
```

Figura 5-2. Programa PRACT5B.ASM que utiliza las instrucciones de cadena.

5.2. Programas propuestos para el uso de Instrucciones de Manejo de Cadenas

5.2.1. *Programa E.1*

Usando las instrucciones de cadenas, realizar una función que nos pida que le introduzcamos una cadena de caracteres y **nos cuente y visualice el número de vocales y consonantes que tiene**.

5.2.2. *Programa E.2*

Usando las instrucciones de cadenas, realizar una función que nos pida que le introduzcamos una cadena de caracteres y **cambie todos los caracteres que estén en minúsculas a mayúsculas**.

5.2.3. *Programa E.3*

Crear un buffer de 10.000 words y llenarlo con 0FFFFh de dos formas: con instrucciones de manejo de cadena y con instrucciones simples (MOV, INC y LOOP). Contabilizar y comparar el tiempo en segundos que cuesta llenar 100.000 veces este buffer con uno y otro método, determinar cuál de ellos es más rápido.

5.2.4. *Programa E.4*

Usando las instrucciones de cadenas, realizar una función que nos pida que le introduzcamos una cadena de caracteres y **la imprima al revés (de derecha a izquierda).**

5.2.5. *Programa E.5*

Usando las instrucciones de cadenas, realizar una función que nos pida que le introduzcamos una cadena de caracteres **y nos ordene las palabras de la cadena introducida en orden alfabético visualizándolas por separado.**

5.2.6. *Programa E.6*

Usando las instrucciones de cadenas, realizar una función que nos pida que le introduzcamos una cadena de caracteres **y nos elimine los espacios, comas, puntos y comas, puntos y dos puntos; visualizando la cadena resultante al revés.**

5.2.7. *Programa E.7*

Usando las instrucciones de cadenas, realizar una función que nos pida que le introduzcamos una cadena de caracteres y **nos elimine las palabras que empiecen por vocal visualizando la cadena resultante**.

5.2.8. *Programa E.8*

Usando las instrucciones de cadenas, realizar una función que nos pida que le introduzcamos dos cadenas de caracteres. El programa generará dos nuevas cadenas: la primera estará formada por la 1ª palabra de la cadena 1 + la 2ª palabra de la cadena 2 + la 3ª palabra de la cadena 1 + la 4ª palabra de la cadena 2 + ... y así sucesivamente; la segunda cadena estará formada por la 1ª palabra de la cadena 2 + la 2ª palabra de la cadena 1 + la 3ª palabra de la cadena 2 + la 4ª palabra de la cadena 1 + ... y así sucesivamente.

5.2.9. *Programa E.9*

Realizar un programa que nos pida que le introduzcamos una secuencia de números separados por comas. Ejemplo: 123,23,45,64,22,21,22,234,500,212,932,212,...

El programa nos ordenará de menor a mayor los números introducidos y nos dará la suma resultante de los mismos.

5.2.10. Programa E.10

Realizar un programa que nos pida que le introduzcamos una frase y después una palabra. Éste nos buscará la palabra en la frase y nos indicará en que posición se encuentra. En el proceso de búsqueda se considerarán las mayúsculas y minúsculas iguales entre sí, por ejemplo si le decimos que busque la palabra ‘CaSa’ dentro de una frase ‘El otro día cayó un rayo cerca de mi **casa**’; nos la encontrará. (Las letras mayúsculas y las minúsculas sólo se diferencian en un bit, ¡descubre cual es!).

PRÁCTICA 6

6. La Pantalla en Modo Alfanumérico

6.1. La Pantalla en Modo Alfanumérico

Dentro de los modos de texto que tenemos en el ordenador, se va a utilizar en los ejemplos el modo más común y que generalmente se establece por defecto al arrancar el ordenador.

MODO TEXTO: 80(Columnas)x25(Líneas) siendo la esquina superior izquierda la coordenada (0,0). Para poner este modo de pantalla se utiliza la función 00h de la BIOS de vídeo (INT 10h):

```
Función "Modo" de la Interrupción 10h de la BIOS
(Modos Básicos de Vídeo CGA).

   MOV AH,00h
   MOV AL,Modo ;Modo:  AL=0  40x25, blanco y negro, alfanumérica.
                   ;  AL=1  40x25, color, alfanumérica.
                   ;  AL=2  80x25, blanco y negro, alfanumérica.
                   ;  AL=3  80x25, color, alfanumérica.
                   ;  AL=4  320x200, color, gráfica.
                   ;  AL=5  320x200, blanco y negro, gráfica.
                   ;  AL=6  640x200, blanco y negro, gráfica.
   INT 10h
```

La dirección de la pantalla en modo texto es la **B800:0000h=B8000h** siendo los dos primeros bytes correspondientes al carácter de la esquina superior izquierda, los dos siguientes al segundo carácter de la primera fila, y así, sucesivamente hasta la segunda línea, tercera, etc.

Cada carácter esta formado por dos bytes:

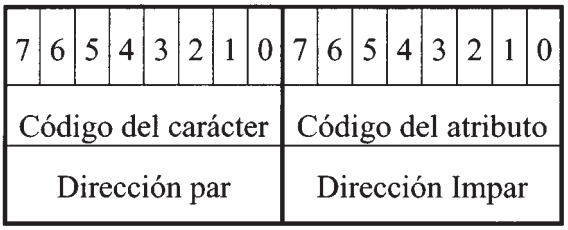


Figura 6-1. Organización de los dos bytes de cada dirección par de pantalla.

El código de atributo es el color del fondo y del carácter, de la siguiente forma:

7	6	5	4	3	2	1	0
Parpadeo	Rojo	Verde	Azul	Intensidad	Rojo	Verde	Azul
	Fondo			Carácter			

Figura 6-2. Significado de los bits del atributo.

Correspondiente cada bit del atributo a:

Bits	*Descripción*
7	Parpadeo (P) del carácter
6	Componente rojo (R) del fondo
5	Componente verde (V) del fondo
4	Componente azul (A) del fondo
3	Intensidad (I) del carácter
2	Componente rojo (R) del carácter
1	Componente verde (V) del carácter
0	Componente azul (A) del carácter

Figura 6-3. Descripción de cada bit del atributo.

Tenemos por tanto 16 posibles colores (4 bits) para el carácter y 8 (3 bits) para el fondo. Los colores que se obtienen son:

Número	*I*	*R*	*V*	*A*	*Color*
0	0	0	0	0	Negro
1	0	0	0	1	Azul
2	0	0	1	0	Verde
3	0	0	1	1	Azul-Verde (cyan)
4	0	1	0	0	Rojo
5	0	1	0	1	Magenta
6	0	1	1	0	Marrón
7	0	1	1	1	Blanco
8	1	0	0	0	Gris
9	1	0	0	1	Azul claro
10	1	0	1	0	Verde claro
11	1	0	1	1	Azul-Verde (cyan) claro
12	1	1	0	0	Rojo claro
13	1	1	0	1	Magenta claro
14	1	1	1	0	Amarillo
15	1	1	1	1	Blanco intenso

Figura 6-4. Combinaciones de colores posibles.

En el modo de 80x25 de la pantalla CGA, tenemos la posibilidad de almacenar cuatro páginas a la vez (cuatro pantallas de texto). Por defecto, la pantalla activa (la que vemos) es la página 0, pero se puede modificar con la función 05h de la BIOS de vídeo:

Función "Seleccionar Página Activa" de la Interrupción 10h de la BIOS.

```
MOV AH,05h
MOV AL,Página
INT 10h
```

Figura 6-5. Uso de la función 05h de la BIOS para cambiar la página activa.

En la figura siguiente podemos ver el valor que hay que sumarle a la dirección B8000h para acceder a cada página.

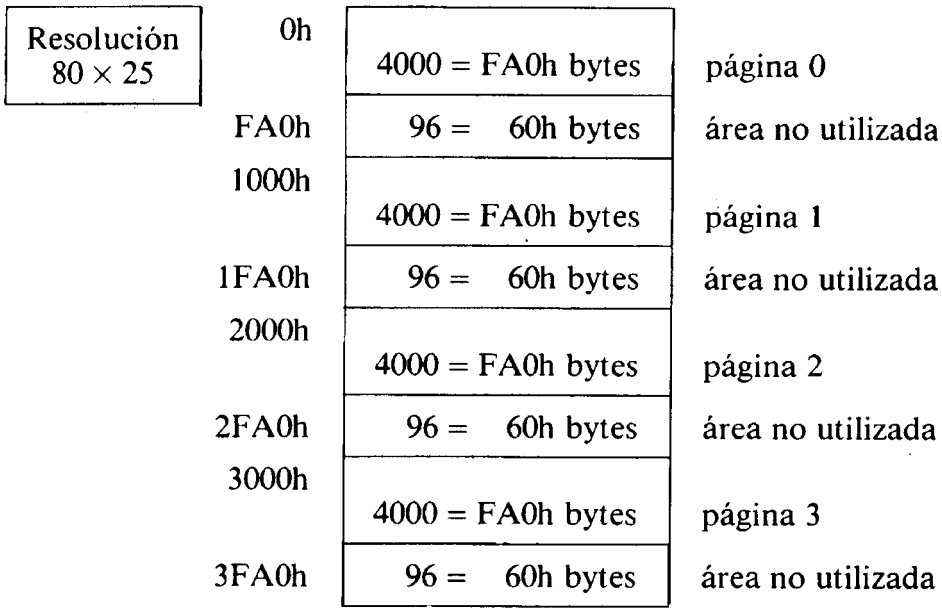


Figura 6-6. Direcciones de las cuatro páginas del modo texto de resolución 80x25.

6.2. EJEMPLOS DE PROGRAMAS QUE USAN EL MODO TEXTO

Escribir y comprender el funcionamiento de los siguientes programas.

6.2.1. *Programa ejemplo "PRACT6A.ASM"*

```
; CONSTANTES Y VARIABLES
DIR_INICIAL     EQU 0B800h      ;SEGMENTO DE DIRECCION DE PANTALLA

PAUSA               MACRO NUMB1,NUMB2
                    LOCAL BP1,BP2
                    PUSH CX
                    MOV CX,NUMB1
BP1:                PUSH CX
                    MOV CX,NUMB2
BP2:                LOOP BP2
                    POP CX
                    LOOP BP1
                    POP CX
                    ENDM
```

```
MUEVE_FILA          MACRO FILA
                    MOV AX,DIR_INICIAL
                    MOV DS,AX
                    MOV AX,DIR_INICIAL
                    MOV ES,AX
                    MOV AX,FILA
                    MOV BX,160
                    MUL BX
                    MOV BX,AX

                ENDM

;
PILA SEGMENT STACK          ;Comienzo segmento PILA
     DB 128 DUP('PILA')     ;Inicialización PILA
PILA ENDS
;

CODIGO     SEGMENT     ;Comienzo segmento CODIGO
EJEMPLO    PROC FAR
           ASSUME   CS:CODIGO,SS:PILA

           PUSH DS     ;Guarda Segmento en pila (DS:AX dirección retorno)
           SUB AX,AX   ;Borrar registro AX=0
           PUSH AX     ;Guarda en Pila AX (IP=0)

           MOV CX,23
VECES:     PUSH CX
           MOV CX,23
BUCLE:     PUSH CX
           MUEVE_FILA CX
           POP CX
           LOOP BUCLE
           POP CX
           PAUSA 300,20000
           LOOP VECES
FINPROG:   RET              ;RETORNA
EJEMPLO    ENDP             ;FIN DE PROCEDIMIENTO
CODIGO     ENDS             ;FIN DE SEGMENTO DE CODIGO
           END EJEMPLO      ;FIN DE PROGRAMA EJEMPLO
```

Figura 6-7. Programa "PRACT6A.ASM"

6.2.2. *Programa ejemplo "PRACT6B.ASM"*

```
; ######################################################
; USO DIRECTO DE LA MEMORIA DE PANTALLA EN MODO TEXTO
; ######################################################

; CONSTANTES Y VARIABLES
DIR_INICIAL    EQU 0B800h      ;SEGMENTO DE DIRECCIÓN DE PANTALLA
MODO           EQU 3           ;MODO DE PANTALLA
X_IZQ          EQU 0
Y_SUP          EQU 0
X_DER          EQU 30
```

```
Y_INF           EQU 20

DATOS           SEGMENT
TEXTO           DB ' TEXTO A IMPRIMIR   '
TEXTO2          DB '                    '
FIL_TEXTO       DB 0
COL_TEXTO       DB 0
DATOS           ENDS



; MACRO DE UN BUCLE DE PAUSA

PAUSA       MACRO
            LOCAL BUC,BUC2

            MOV CX,10
BUC2:       PUSH CX
            MOV CX,0FFFFh
BUC:        NOP
            LOOP BUC
            POP CX
            LOOP BUC2

            ENDM


;-------------------------------------------------
;
;Segmento de Pila
;----------------
;
PILA SEGMENT STACK          ;Comienzo segmento PILA
     DB 128 DUP('PILA')     ;Inicialización PILA
PILA ENDS
;
;-------------------------------------------------
;
;Segmento de Código
;------------------
;
CODIGO      SEGMENT     ;Comienzo segmento CODIGO
EJEMPLO     PROC FAR
            ASSUME      CS:CODIGO,DS:DATOS,SS:PILA

            PUSH DS     ;Guarda Segmento en pila (DS:AX dirección retorno)
            SUB AX,AX   ;Borrar registro AX=0
            PUSH AX ;Guarda en Pila AX (IP=0)

            MOV AL,MODO     ;MODO DE PANTALLA
            INT 10h



BAJA:       CALL P_TXT
            PAUSA
            CALL P_TXT2
```

```
            INC [FIL_TEXTO]
            CMP [FIL_TEXTO],Y_INF
            JL BAJA

DERE:       CALL P_TXT
            PAUSA
            CALL P_TXT2

            CALL TECLA
            JC FINPROG

            INC [COL_TEXTO]
            CMP [COL_TEXTO],X_DER
            JL DERE

SUBE:       CALL P_TXT
            PAUSA
            CALL P_TXT2

            CALL TECLA
            JC FINPROG

            DEC [FIL_TEXTO]
            CMP [FIL_TEXTO],Y_SUP
            JG SUBE

IZQU:       CALL P_TXT
            PAUSA
            CALL P_TXT2

            CALL TECLA
            JC FINPROG

            DEC [COL_TEXTO]
            CMP [COL_TEXTO],X_IZQ
            JG IZQU

FINPROG:    RET              ;RETORNA
EJEMPLO     ENDP             ;FIN DE PROCEDIMIENTO

; ######################################################################
; #        TECLA:          FUNCION QUE SALE AL DOS SI SE PULSA ESC     #
; ######################################################################

TECLA       PROC
            MOV AH,01
            INT 16H
            JZ SALTEC
            XOR AX,AX
            INT 16H
            CMP AH,01
            JNE SALTEC
            STC
            RET
SALTEC:     CLC
            RET
TECLA       ENDP
```

```
; ############################################################
; #        P_TXT:          FUNCION QUE LLAMA A PON_TEXTO      #
; ############################################################
P_TXT2      PROC
            LEA BX,TEXTO2
            JMP ETIQ1
P_TXT2      ENDP

P_TXT       PROC
            LEA BX,TEXTO        ;DIRECCION TEXTO DS:BX

ETIQ1:      MOV AX,DATOS        ;INICIALIZAR SEGMENTO DE DATOS
            MOV DS,AX

            MOV CL,18           ;NUM CARACTERES
            MOV DL,[COL_TEXTO]  ;COLUMNA
            MOV DH,[FIL_TEXTO]  ;FILA
            MOV CH,00001100b    ;ATRIBUTO "PRVAIRVA=00001100b"

            CALL PON_TEXTO
            RET
P_TXT       ENDP

; ############################################################
; #        PON_TEXTO:      FUNCION QUE PONE DIRECTAMENTE UN  #
; #                         TEXTO EN PANTALLA                #
; #        ENTRADAS:       DL=COLUMNA                       #
; #                         DH=FILA                          #
; #                         DS:BX=TEXTO                      #
; #                         CL=NUMERO DE CARACTERES          #
; #                         CH=ATRIBUTO                      #
; ############################################################


PON_TEXTO   PROC
            CMP DL,80       ;SE MIRA SI ESTA DENTRO DE LAS COORDENADAS
            JGE SALIDA      ;DL=[0..79]
            CMP DL,0        ;DH=[0..24]
            JL SALIDA
            CMP DH,25
            JGE SALIDA
            CMP DH,0
            JL SALIDA

            PUSH CX
            PUSH DX

            MOV AX,DIR_INICIAL
            MOV ES,AX       ;INICIALIZAMOS SEGMENTO DE DESTINO

            XOR AX,AX       ;CALCULAMOS DI=(DL*160)+(DH*2)
            MOV AL,DH
            MOV CX,160
            MUL CX

            POP DX
```

```
            MOV DH,0
            SHL DL,1        ;MULTIPLICA BL POR 2
            ADD AX,DX
            MOV DI,AX       ;ES:DI ES EL DESTINO EN MEMORIA DE PANTALLA

            POP CX
            MOV AH,CH
            MOV CH,0
PONCAD:     MOV AL,[BX]     ;MOVER UN BYTE DE LA CADENA DE CARACTERES
            MOV ES:[DI],AL
            INC DI
            MOV ES:[DI],AH      ;MOVER EL ATRIBUTO
            INC DI              ;INCREMENTA LA DIRECCION
            INC BX
            LOOP PONCAD         ;BUCLE HASTA CX=0

SALIDA:     RET
PON_TEXTO   ENDP
CODIGO      ENDS            ;FIN DE SEGMENTO DE CODIGO
            END EJEMPLO     ;FIN DE PROGRAMA EJEMPLO
```

Figura 6-8. Programa "PRACT6B.ASM" que maneja directamente la memoria de video en modo texto.

6.3. Programas propuestos para el uso de la pantalla en modo alfanumérico

6.3.1. *Programa F.1*

Realizar una función que limpie la pantalla accediendo directamente a la memoria de vídeo. Comparar el tiempo que cuesta borrar 100.000 veces la pantalla de esta forma con la función del programa A.3.

6.3.2. *Programa F.2*

Diseñar un programa que cambie los atributos de la pantalla (color de cada carácter): los pares de color verde y los impares de color azul.

6.3.3. *Programa F.3*

Realizar un programa que realice una OR EXCLUSIVA con 01010101b de todos los caracteres de la pantalla.

6.3.4. *Programa F.4*

Realizar un programa que realice los siguientes tipos de borrados:

A. Tipo telón: Realizar un programa que borre la pantalla como un telón, es decir, una columna de color negro que avanza por la izquierda y otra que avanza por la derecha borrando la pantalla.

B. Un cuadrado: Un cuadrado negro que va creciendo desde el centro de la pantalla y va borrando todo la pantalla. Modo texto.

C. Telón diagonal: Un telón que empieza de las cuatro esquinas y se va cerrando hasta el centro de la pantalla.

6.3.5. *Programa F.5*

Realizar una función que imprima directamente en la pantalla el texto enviándole los siguientes datos en la pila:

1. El atributo del texto.
2. La coordenada X del texto.
3. La coordenada Y del texto.
4. La dirección del texto (SEGMENTO).
5. La dirección del texto (DESPLAZAMIENTO).

6.3.6. *Programa F.6*

Realizar una función que realice un telón como el del programa F.4 de forma que se cierre y se abra manteniendo el texto que tenía anteriormente. Para ello se realizará una copia de toda la pantalla en otra página de la memoria de texto.

6.3.7. *Programa F.7*

Realizar un programa que escriba en la esquina superior izquierda un *smile* sonriente :-), si en la pantalla hay una palabra que ponga CONTENTO; o que ponga un *smile* triste :-(, si en la pantalla hay una palabra que ponga TRISTE (pueden estar en mayúsculas o minúsculas).

6.3.8. *Programa F.8*

Realizar cuatro funciones que realicen respectivamente:

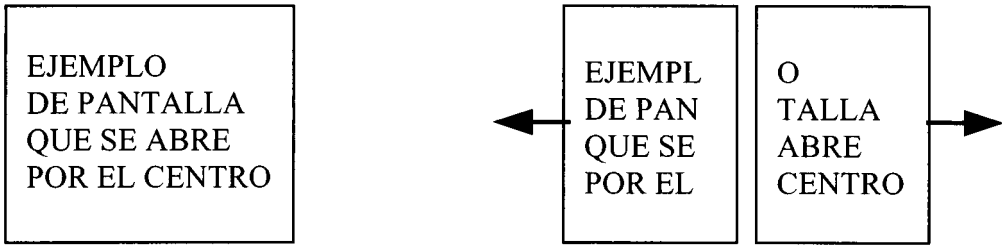
1. Un desplazamiento de todas las filas hacia arriba (*scroll* hacia arriba).
2. Un *scroll* hacia abajo (Mover todas las filas una posición hacia abajo).
3. Un *scroll* hacia la derecha (desplazar las columnas una posición hacia la derecha).
4. Un *scroll* hacia la izquierda (desplazar las columnas una posición hacia la izquierda).

6.3.9. *Programa F.9*

Situar un cursor parpadeante en el centro de la pantalla (usando sólo los atributos) y que se pueda mover con los cursores del teclado.

6.3.10. *Programa F.10*

Realizar un programa que abra la pantalla desde el centro y desplace las columnas a modo de telón.



PRÁCTICA 7

7. Las Interrupciones Hardware. Programas Residentes

7.1. Las interrupciones hardware

Una interrupción hardware ocurre habitualmente ante un evento externo que se ha producido. Generalmente el ordenador tiene que realizar una determinada tarea cuando esto sucede: capturar o enviar una serie de datos, activar una serie de alarmas, etc.

En el programa ejemplo del punto siguiente se muestra un mecanismo muy utilizado de captura de datos en un buffer cíclico con dos punteros: uno de escritura y otro de lectura. De esta forma, a medida que se van produciendo interrupciones externas (en este caso la pulsación de una tecla entre el "0" y el "9"), se van guardando los datos obtenidos.

Por otro lado, existe una función con el último dato al que apunta el puntero de lectura.

7.1.1. *Programa ejemplo "PRACT7A.ASM"*

Entender el funcionamiento del programa siguiente:

```
; Símbolos:
    pantalla       equ  0b800h        ; segmento memoria de pantalla


; Segmento DATOS
; ---------
; variables
; ---------

DATOS   segment
ant_teclado_int      equ this dword
ant_tec_ofs dw 0
ant_tec_seg dw 0
TEXTO_INICIO1 DB 'ESCRIBE=TECLAS "0" AL "9" $'
TEXTO_INICIO2 DB 'LEE="ENTER", SALIR="ESC $'

BUFF_TEC DB 50 DUP ('_')
          DB 0
PUNT_ESC DW 0
PUNT_LEC DW 0
BUFF_INICIO DB 0
DATOS   ends

;Segmento PILA

PILA    segment stack
        DB 128 dup ('PILA')
PILA    ends
```

```
; -------------------
; segmento de código
; -------------------

codigo segment
        assume  cs:codigo, ds:DATOS, ss:PILA
;
; direccionar segmento interrupcs con es
;
instalar proc far
          push ds
          xor ax,ax
          push ax


          mov  es,ax      ;INICIALIZAMOS 'ES' CON CERO

          mov ax,DATOS
          mov ds,ax       ;INICIALIZAMOS 'DS' CON 'DATOS'

          ; IMPRIMO TEXTOS INICIALES
          mov dx,0419h    ;POSICIÓN PANTALLA
          mov ah,2
          mov bh,0
          int 10h
          lea dx,TEXTO_INICIO1
          mov ah,09h
          int 21h

          mov dx,0519h    ;POSICIÓN PANTALLA
          mov ah,2
          mov bh,0
          int 10h
          lea dx,TEXTO_INICIO2
          mov ah,09h
          int 21h
;
; salvar dirección de la rutina de interrupción de teclado actual
;
          mov  ax,es:[09h*4]       ; ax = despl. rutina teclado actual
          mov  [ant_tec_ofs],ax    ; salvar desplazamiento
          mov  ax,es:[09h*4+2]     ; ax = segm.  rutina teclado actual
          mov  [ant_tec_seg],ax    ; salvar segmento
;
; poner dirección de la nueva rutina de interrupción de teclado
;
          cli
          mov ax,offset nue_teclado_int
          mov di,09h*4
          mov es:[di],ax
          mov ax,seg nue_teclado_int
          mov es:[di+2],ax
          sti

;
```

```
;       Entramos en un bucle donde si se pulsa el asterisco se lee el dato
;       al que apunta el puntero de lectura (PUNT_LEC)
;
        push es
BUCF:   in al,60h
        cmp al,1Ch
        jne BUCF2
        call leedato
        mov bx,pantalla
        mov es,bx
        mov di,160*13+80
        mov ah,57h
        mov es:[di],al
        mov es:[di+1],ah
ESPERA: in al,60h
        cmp al,1Ch
        je ESPERA

BUCF2:  in al,60h
        cmp al,1
        jne BUCF
        pop es
        ;DEJAMOS LA INT DE TECLADO COMO ESTABA ...
        cli
        mov di,09h*4
        mov ax,[ant_tec_ofs] ; desplazamiento dir inicial
        mov es:[di],ax
        mov ax,[ant_tec_seg] ; segmento dir inicial
        mov es:[di+2],ax
        sti

        ret

instalar endp

;       Rutina leedato: lee un dato del buffer de captura si hay dato en
el
;       buffer e incrementa el puntero de lectura (SALIDA EN 'AL')
;       si no hay dato devuelve el acarreo a uno (CF=1)
;
leedato proc

        push ax
        mov ax,'  '
        call PONPUNTS  ;Borramos los indicadores de los punteros
        pop ax

        mov di,[PUNT_LEC]
        cmp di,[PUNT_ESC]
        je NO_HAY_DATO   ;SI P.LECTURA=P.ESCRITURA NO HAY DATO QUE DEVOLVER

        mov al,BUFF_TEC[di]   ;DEVOLVEMOS EL CARACTER AL QUE APUNTA EL
PUNT. LECTURA
        inc di
        cmp di,50       ;MIRAMOS SI HA LLEGADO AL FINAL DEL BUFFER
        jne BUC4
```

```
            mov di,00       ;INICIALIZAMOS 'DI' AL PRINCIPIO DEL BUFFER
BUC4:       mov [PUNT_LEC],di       ;GUARDAMOS EL NUEVO PUNT.ESCRITURA
            push ax
            mov ax,'LW'
            call PONPUNTS  ;Borramos los indicadores de los punteros
            pop ax
            clc
            ret
NO_HAY_DATO:
            push ax
            mov ax,'LW'
            call PONPUNTS  ;Borramos los indicadores de los punteros
            pop ax
            mov al,'-'
            stc
            ret
leedato     endp


;--------------------------------------------------------------------
;
; nueva rutina de interrupción de teclado
; ---------------------------------------
;
nue_teclado_int proc near
            assume cs:codigo, ds:DATOS
            cli
            push ax              ; salvar registros
            push bx
            push cx
            push dx
            push di
            push si
            push ds
            push es

            mov ax,40h
            mov es,ax
            mov bx,es:[1Ch]    ;Cogemos el puntero de escritura de la BIOS
            push es
            push bx

            pushf                    ; salvar banderas
            call ant_teclado_int     ; llamada a la rutina de
                                     ; interrupción anterior
            cli
            pop bx
            pop es
            mov es:[1Ch],bx    ;Dejamos el puntero de escritura de la BIOS
igual
            mov al,es:[17h]          ;MIRAMOS SI SE HA PULSADO EL 'ALT'
            and al,00001111b         ; 'CTR', 'MAYÚSCULAS DER o IZQ'
            jnz fin_nue              ;SI ES ASI NO HACE NADA


            in al,60h
            cmp al,2h
```

```
           jl fin_nue              ;MIRAMOS SI SE HA PULSADO UN NéMERO
           cmp al,11               ;SI NO ES ASI SALE
           jg fin_nue


           call proceso            ; salto a proceso
;
; salida de la rutina de interrupción de teclado
;
fin_nue:
           pop  es         ; restaurar registros
           pop  ds
           pop  si
           pop  di
           pop  dx
           pop  cx
           pop  bx
           pop  ax
           sti
           iret            ; una interrupción necesita iret
nue_teclado_int endp
;
;--------------------------------------------------------------------------
; -------
; proceso
; -------
;
proceso    proc
           mov cx,DATOS
           mov ds,cx       ;INICIALIZAMOS DS=DATOS

           push ax
           mov ax,'  '
           call PONPUNTS   ;Borramos los indicadores de los punteros
           pop ax

           mov di,[PUNT_ESC]
           inc di
           cmp di,50
           jne NO_FINAL
           mov di,0
NO_FINAL:
           cmp di,[PUNT_LEC]
           je NO_GUARDA    ;MIRAMOS SI P.LECTURA=P.ESCRITURA NO GUARDA

           cmp al,11               ;¨ES CERO?
           jne NO_CERO
           add al,'0'-11           ;NUM DISTINTO CERO
           jmp CONT
NO_CERO: add al,'1'-2
CONT:      mov si,[PUNT_ESC]
           mov BUFF_TEC[si],al   ;GUARDAMOS EL CARACTER DEL NUMERO PULSADO
           mov [PUNT_ESC],di     ;GUARDAMOS EL NUEVO PUNT.ESCRITURA

NO_GUARDA:
           lea si,BUFF_TEC       ;IMPRIMIMOS LA LÖNEA EN PANTALLA
           mov ax,pantalla
```

```
            mov es,ax
            mov di,160*10+(15*2)     ;SEGUNDA LÍNEA CENTRADA
            mov ah,47h               ;ATRIBUTO=FONDO ROJO, TINTA BLANCA
BUC1:       mov al,ds:[si]
            cmp al,00
            je salir
            mov es:[di],ax
            inc di
            inc di
            inc si
            jmp BUC1
salir:
            mov ax,'LW'
            call PONPUNTS  ;Ponemos los indicadores de los punteros
            ret
proceso     endp
;
            ;Ponemos la 'L' y la 'E' de los punteros de escritura y lectura
            ; o el espacio segfn AL y AH
PONPUNTS proc
            mov bx,pantalla
            mov es,bx
            mov di,160*9+(15*2)
            mov bx,[PUNT_ESC]
            shl bx,1             ;MULTIPLICAMOS POR DOS
            mov es:[di+bx],al    ;Pone el contenido de AL='W' o ' '
            mov di,160*11+(15*2)
            mov bx,[PUNT_LEC]
            shl bx,1             ;MULTIPLICAMOS POR DOS
            mov es:[di+bx],ah ;Pone el contenido de AL='L' o ' '
            ret
PONPUNTS endp
;
codigo      ends
            end  instalar
```

Figura 7-1. Programa ejemplo "PRACT7A.ASM"

7.2. Programas propuestos para el uso de las interrupciones hardware

Antes de realizar los programas propuestos **hay que tener en cuenta que si se cambia una dirección de salto de una interrupción, antes de salir del programa hay que dejar la antigua dirección que tenía.** Esto es debido a que cuando salimos de un programa, éste se elimina de la memoria de forma que, si no recuperamos la dirección antigua de la interrupción, cuando se produzca ésta saltará a una zona de la memoria vacía, obteniéndose resultados inesperados (generalmente se colgará el ordenador).

7.2.1. *Programa G.1*

Realizar un programa, modificando el ejemplo anterior, para que cambie la interrupción hardware de teclado y cada vez que se pulse una tecla ponga el número de la tecla pulsada en hexadecimal en la esquina superior derecha.

7.2.2. *Programa G.2*

Cambiar la interrupción 00h Hardware (*Error de división por cero*), de forma que cuando se realice una división incorrecta nos ponga en el centro de la pantalla en fondo rojo y parpadeante "LA DIVISIÓN QUE HAS EFECTUADO ES INCORRECTA".

NOTA: Cuando la CPU salta a esta interrupción, **la dirección de retorno que almacena es la de la instrucción DIV (la que produce el error de división por cero), de forma que si volvemos de la interrupción, volverá a ejecutarse la instrucción DIV generándola de nuevo.**

7.2.3. *Programa G.3*

Realizar un salto a la interrupción 02h Hardware (NMI) y ver que sucede.

7.2.4. *Programa G.4*

Cambiar la interrupción 03h (*Breakpoint*) para que salte a una rutina donde visualice el contenido de todos los registros en hexadecimal y de los FLAGS cuando el procesador encuentre la instrucción INT (código CCh).

7.2.5. *Programa G.5*

Desviar la interrupción 1Ch para que salte a una rutina que incremente e imprima un contador que vaya desde '00000000' hasta '99999999'. Esta interrupción se activa 18.2 veces con segundo.

7.3. Programas Residentes

Un programa residente es aquel que se queda en memoria después de que el programa principal ha terminado y retorna al DOS. En todo sistema operativo existen programas residentes como el controlador de ratón, controlador de memoria extendida, control del teclado, programas de comunicaciones, *Watch-Dogs*, etc.

7.3.1. *Programa ejemplo "PRACT7B.ASM"*

El programa siguiente se queda residente y espera a que se pulse la combinación de teclas 'ALT-P' para situar un texto en pantalla.

```
; Símbolos:
    pantalla       equ  0b800h         ; segmento memoria de pantalla
    tecla_accion   equ  1900h          ; Alt-P

;
;-----------------------------------------------------------------
;
; segmento principal
```

```
; ---------
;
codigo segment
        assume      cs:codigo
        org         100h        ; origen por ser fichero tipo COM
empezar:
        jmp         instalar    ; bifurca a la rutina de inicialización
;
;-----------------------------------------------------------------
;
; variables
; ---------
;
ant_teclado_int      equ this dword
ant_tec_ofs dw 0
ant_tec_seg dw 0
;-----------------------------------------------------------------
;
; nueva rutina de interrupción de teclado
; ---------------------------------------
;
nue_teclado_int proc near
          assume cs:codigo
          push ax                       ; salvar registros
          push bx
          push cx
          push dx
          push di
          push si
          push ds
          push es
          pushf                         ; salvar banderas
          call ant_teclado_int          ; llamada a la rutina de
          in al,60h                     ; interrupción anterior
          cmp al,19h
          jne fin_nue                   ;MIRAMOS SI SE HA PULSADO LA 'P'

          mov ax,40h
          mov es,ax
          mov al,es:[17h]               ;MIRAMOS SI SE HA PULSADO EL 'ALT'
          and al,00001111b
          cmp al,00001000b
          jne fin_nue

          call proceso                  ; salto a proceso
;
; salida de la rutina de interrupción de teclado
;
fin_nue:
          pop  es       ; restaurar registros
          pop  ds
          pop  si
          pop  di
          pop  dx
          pop  cx
          pop  bx
          pop  ax
```

```
            iret            ; una interrupción necesita iret
nue_teclado_int endp
;
;--------------------------------------------------------------------
;
; proceso
; -------
;
TEXTO_ACT DB 'SE HA PULSADO ALT-P...',0
proceso     proc
            lea si,TEXTO_ACT
            mov ax,pantalla
            mov es,ax
            mov di,0
            mov ah,47h
BUC1:       mov al,cs:[si]
            cmp al,00
            je salir
            mov es:[di],ax
            inc di
            inc di
            inc si
            jmp BUC1
salir:      ret
proceso     endp
;
;--------------------------------------------------------------------
;
; instalación
; -----------
;
instalar proc near
;
; direccionar segmento interrupcs con es
;
            assume es:0
            xor ax,ax
            mov  es,ax
;
; salvar dirección de la rutina de interrupción de teclado actual
;
            mov  ax,es:[09h*4]      ; ax = despl. rutina teclado actual
            mov  [ant_tec_ofs],ax   ; salvar desplazamiento
            mov  ax,es:[09h*4+2]    ; ax = segm.  rutina teclado actual
            mov  [ant_tec_seg],ax   ; salvar segmento
;
; poner dirección de la nueva rutina de interrupción de teclado
;
            cli
            mov ax,offset nue_teclado_int
            mov di,09h*4
            mov es:[di],ax
            mov es:[di+2],cs

            sti
;
; terminar rutina de inicialización, pero dejar residente la rutina
```

```
; de interrupción de teclado
;
           mov  dx,offset instalar  ; dx = última dirección + 1 respecto
                                    ; al segmento de la rutina de
                                    ; interrupción
           int  27h                 ; terminar, pero quedar residente
instalar endp
;
;--------------------------------------------------------------------------
;
codigo     ends
           end  empezar
```

Figura 7-2. Programa "PRACT7B.ASM".

Entender el funcionamiento considerando que:

- Para instalar una rutina se sitúa la dirección de salto a la interrupción del programa residente y se sale del programa de instalación con la interrupción 27h (INT 27h), **indicando en DX la última dirección más uno respecto del segmento de la rutina de interrupción que va a quedar residente**. Esta es la forma con la que obtiene la longitud de la misma.
- Debido al modo especial con que se sale de la rutina de instalación, es necesario **convertir el archivo a ".COM"** con el comando '/t' del *TLINK*. El tamaño máximo de la rutina residente no puede ser mayor de 64k.

```
Rem Programa e2.bat
cls
edit %1.asm
tasm %1.asm
Rem Genera un archivo .COM
tlink %1.obj /t
pause
cls
%1.com
```

Figura 7-3. Fichero ".BAT" ejemplo que sirve para generar un archivo ".COM".

- El programa llama a la anterior rutina de interrupciones, simulando un salto de interrupción; es decir:

```
   pushf                        ; salvar banderas
   call ant_teclado_int         ; llamada a la rutina anterior
```

Figura 7-4. Forma de simular un salto de una interrupción.

- Cada vez que ejecutamos el programa de instalación, se instala en memoria (usar MEM /D para ver los programas residentes en DOS), de forma que si hemos ejecutado 5 veces el mismo programa tendremos 5 programas residentes en memoria que se llaman unos a otros, debido a que el programa llama a la dirección anterior de la interrupción anti-

gua. Para evitar esto, sería necesario **detectar antes de instalar que el programa ya está en memoria** (se puede desviar la interrupción del multiplexor del DOS (int 2Fh) para que nos devuelva un código que nos indique que ya está instalado y la dirección y tamaño del programa para desinstalarlo).

- Otro punto a tener en cuenta es que cuando se produce el salto a un programa residente pueden ocurrir en ese momento sucesos delicados como el acceso a disco o la ejecución de una llamada al DOS (donde se usa tres pilas diferentes según la llamada que se haga). Debido a esto, **la llamada a una interrupción DOS desde una subrutina residente no se hace** a no ser que se detecte que no se está ejecutando ninguna función DOS en ese momento. Si se produce un salto a la rutina residente y luego dentro de esta llamamos a una función del DOS, esto producirá que se sobreescriban los valores de la pila que estaba usando la función del DOS en el momento de producirse la interrupción, con lo que es probable que el ordenador se "cuelgue" al retornar de la rutina de interrupciones.

Debido a estas circunstancias[1] y algunas otras más, la creación de un programa residente se puede complicar bastante más si se quiere llamar a funciones DOS desde el programa residente, detectar que no se está accediendo al disco en ese momento, etc.

Los ejercicios de ejemplo de esta práctica son más sencillos, buscándose fundamentalmente el aspecto didáctico de la construcción y funcionamiento de programas residentes. No se pretende entrar en profundidad en el diseño de programas residentes, sino que el alumno entienda la utilidad que tiene el manejo de estos mecanismos.

7.4. Programas propuestos para el aprendizaje y creación de programas residentes

7.4.1. *Programa G.6*

Realizar un programa residente que, cuando se pulse la combinación de las teclas 'ALT-Q', limpie la pantalla.

7.4.2. *Programa G.7*

Realizar un programa residente que escriba en la esquina superior izquierda un *smile* sonriente :-), si en la pantalla hay una palabra que ponga CONTENTO; o que ponga un *smile* triste :-(, si en la pantalla hay una palabra que ponga TRISTE (pueden estar en mayúsculas o minúsculas).

7.4.3. *Programa G.8*

Realizar una rutina residente que cada vez que la hora sea en punto nos sitúe ésta en pantalla, en la esquina superior derecha, poniéndonos el mensaje: ¡¡¡ ATENCIÓN !!! SON LAS XX:00 Horas

1. En el libro titulado "PC INTERNO" se explica con detalle y con numerosos ejemplos la programación de programas residentes.

7.4.4. *Programa G.9*

Realizar un programa residente que cada vez que se pulse una tecla guarde el valor de la tecla pulsada (lectura del puerto 60h) y del byte de la variable de la BIOS (40:17h) que sirve para detectar la pulsación de las teclas especiales (‘ALT’, ‘CONTROL’, etc.). El programa guardará en un buffer cíclico con un tamaño de 10.000 *words* todas las teclas pulsadas.

Por otro lado, el programa residente tendrá otra interrupción (la 90h por ejemplo) que buscará una combinación de teclas preestablecidas dentro del buffer cíclico donde se han introducido las teclas que se han ido pulsando. Para ello, a la interrupción 90h se le enviará en DS:DX la dirección absoluta donde está la combinación de teclas a buscar y ésta devolverá en DS:DX la posición donde se encuentra la misma si existe dentro del buffer, sino nos devolverá DX:0FFFFh.

Por ejemplo,

```
HOLA_TEC   DB 23h,00h,18h,00h,26h,00h,1Eh,00h  ;Códigos de las teclas
'H','O','L','A'
       DB 0FFh,0FFh    ;Fin de la serie de teclas
       ...
       ...
       LEA DX,HOLA_TEC ; DS:DX apunta a HOLA_TEC
       INT 90h ;Llamamos a la INT 90h donde busca esos mismos códigos
               ; en el buffer de 10.000 words y si lo encuentra nos
devuelve DS:DX
               ; apuntando a la dirección, sino devuelve en DX=0FFFFh
```

7.4.5. *Programa G.10*

Los mecanismos *watch-dog* (o perro guardián), muy utilizados en autómatas, son sistemas *hardware* o *software* que si no son llamados cada cierto tiempo, o el tiempo entre llamadas es excesivo, generan un salto a una rutina de error. Esto sirve para detectar cuando un programa tiene un código que ralentiza el proceso, o que incluso se queda en un bucle infinito, en PLCs donde el tiempo máximo definido de refresco en la captura y salida de las señales de control es sumamente importante. El interés estriba en que si un programa se queda colgado, el *watch-dog* saltará a una rutina de tratamiento de errores.

Realizar un programa residente que incremente un contador 18.2 veces por segundo, de forma que si llega a 2.000 pulsos salte a una rutina donde ponga:

¡¡¡ SE HA EJECUTADO LA RUTINA DE TRATAMIENTO DE ERRORES !!!

La Interrupción tendrá los siguientes comandos:

- Con AH=00h, se pone el contador a cero.
- Con AH=01h, se activa el *watch-dog* y se inicializa el contador a cero.
- Con AH=02h, se para el *watch-dog*
- Con AH=03h, devuelve en AX el valor del contador.

Inicialmente se instalará la rutina del *watch-dog* pero no se incrementará el contador hasta que no lo activemos con la función AH=01h.

Dentro de nuestro programa principal, tendríamos que llamarlo al final de cada ciclo del mismo para inicializar el contador del *watch-dog* a cero.

PRÁCTICA 8

8. La Pantalla en Modo Gráfico

8.1. Breve explicación del Modo Gráfico a utilizar en esta práctica

El modo de vídeo utilizado (modo gráfico de 640x200 en blanco y negro) se selecciona con la función 00h (AH=00h) de la interrupción 10h de la BIOS y con el modo 6 (AL=6).

Cada bit es un pixel, es decir, cada byte corresponde a 8 puntos de la pantalla. En la dirección B800:0000h están las líneas pares (0, 2, 4, 6, 8, ...) y en la BA00:0000h las impares (1, 3, 5, 7, 9, ...). En el orden siguiente:

Los primeros 8 puntos de la línea 0 están en la dirección B800:0000, los siguientes 8 puntos en la B800:0001 hasta el final de la línea 0 continuando con la línea 2 y así sucesivamente. De esta forma, tenemos dividida la pantalla en dos bancos de memoria de 8 Kb cada uno.

Se ha elegido este tipo de pantalla de los múltiples modos que existen debido a su sencillez de manejo. Esto permite que el alumno se adentre en el uso de gráficos de la forma más rápida y didáctica que se pueda.

8.2. Acceso a un punto cualquiera de la pantalla

El programa ejemplo siguiente utiliza las funciones CALC_DIR para calcular la dirección y posición correspondiente de un pixel cuyas coordenadas son (X_PUNTO, Y_PUNTO). Por otro lado, la función PON_PUNTO sitúa el punto en pantalla.

Entender el funcionamiento del programa “PRACT8A.ASM”, el programa trabaja directamente con la memoria de vídeo en el modo 640x200 (CGA), situando un punto en pantalla gráfica que se mueve por todo el borde de la pantalla.

```
; ########################################################
; PRACTICA NUMERO 8:USO DIRECTO DE LA MEMORIA DE PANTALLA
;                       EN MODO GRAFICO MODO CGA
; ########################################################

; CONSTANTES Y VARIABLES
DIR_INICIAL     EQU 0B800h      ;SEGMENTO DE DIRECCION DE PANTALLA
MODO            EQU 6           ;MODO DE PANTALLA MÁXIMO CGA 640*200 B/N
X_IZQ           EQU 0
Y_SUP           EQU 0
X_DER           EQU 639
Y_INF           EQU 199

DATOS           SEGMENT
X_PUNTO         DW 0
```

```
Y_PUNTO         DW 0

DATOS           ENDS



; MACRO DE UN BUCLE DE PAUSA

PAUSA       MACRO
            LOCAL BUC,BUC2

            MOV CX,1
BUC2:       PUSH CX
            MOV CX,0FFFFh
BUC:        NOP
            LOOP BUC
            POP CX
            LOOP BUC2

        ENDM


;-----------------------------------------------------
;
;Segmento de Pila
;----------------
;
PILA    SEGMENT STACK       ;Comienzo segmento PILA
        DB 128 DUP('PILA')  ;Inicialización PILA
PILA    ENDS

;
;Segmento de CÓDIGO
;------------------
;
CODIGO      SEGMENT     ;Comienzo segmento CODIGO
EJEMPLO     PROC FAR
            ASSUME      CS:CODIGO,DS:DATOS,SS:PILA

            PUSH DS     ;Guarda Segmento en pila (DS:AX dirección retorno)
            SUB AX,AX   ;Borrar registro AX=0
            PUSH AX     ;Guarda en Pila AX (IP=0)

            MOV AX,DATOS
            MOV DS,AX       ;INICIALIZAR DS

            MOV AX,MODO     ;MODO DE PANTALLA
            INT 10h

DERE:       CALL PON_PUNTO
            PAUSA
            INC [X_PUNTO]
            CMP [X_PUNTO],X_DER
            JL DERE

BAJA:       CALL PON_PUNTO
```

```
            PAUSA
            INC [Y_PUNTO]
            CMP [Y_PUNTO],Y_INF
            JL BAJA

IZQU:       CALL PON_PUNTO
            PAUSA
            DEC [X_PUNTO]
            CMP [X_PUNTO],X_IZQ
            JG IZQU
SUBE:       CALL PON_PUNTO
            PAUSA
            DEC [Y_PUNTO]
            CMP [Y_PUNTO],Y_SUP
            JG SUBE

            MOV AH,0
            INT 16h     ;ESPERA A QUE SE PULSE UNA TECLA

            MOV AX,3    ;MODO DE PANTALLA TEXTO
            INT 10h
            RET             ;RETORNA
EJEMPLO     ENDP            ;FIN DE PROCEDIMIENTO

; ############################################################
; #  PON_PUNTO:      FUNCION QUE PONE UN PUNTO EN PANTALLA  #
; ############################################################
PON_PUNTO PROC
            CALL CALC_DIR

            MOV CH,10000000b    ;BIT MAS SIGNIFICATIVO A 1
            SHR CH,CL           ;MOVEMOS EL BIT A 1 'CL' VECES A LA DERECHA

            MOV AX,DIR_INICIAL
            MOV ES,AX
            MOV AL,ES:[BX]
            OR AL,CH
            MOV ES:[BX],AL

            RET
PON_PUNTO   ENDP

; ############################################################
; # CALCULA DIRECCION EN PANTALLA SEGUN [X_PUNTO] [Y_PUNTO]  #
; #              DEVUELVE:                                   #
; #                          BX=OFFSET DIRECCION             #
; #                          CL=NUMERO BIT DE ESA POSICIÓN   #
; #                             EMPEZANDO POR LA DERECHA     #
; ############################################################

CALC_DIR    PROC
            XOR BX,BX           ;BORRAMOS BX
            MOV DX,80           ;BYTES POR FILA

            MOV AX,[Y_PUNTO]    ;AX=FILA
            SHR AX,1            ;DIVIDO POR DOS, EL BIT 0 CAE EN EL ACARREO
            JNC ES_PAR          ;SI NO HAY ACARREO ES PAR
```

```
            MOV BX,2000h        ;SI ES IMPAR BX=8192d=2000h
ES_PAR:     MUL DX              ;MULTIPLICA LA FILA POR 80 BYTES POR FILA
            ADD BX,AX           ;SUMASELO A BX

            MOV AX,[X_PUNTO]    ;COGEMOS LA X
            SHR AX,3            ;LA DIVIDIMOS POR 8
            ADD BX,AX           ;EN BX TENEMOS LA DIRECCIàN DEL BYTE

            MOV AX,[X_PUNTO]
            AND AL,00000111b    ;COGEMOS EL RESTO DE DIVIDIR POR 8
            MOV CL,AL           ;EN CL TENEMOS EL DESPLAZAMIENTO DEL PIXEL
            RET
CALC_DIR    ENDP
CODIGO      ENDS                ;FIN DE SEGMENTO DE CODIGO
            END EJEMPLO         ;FIN DE PROGRAMA EJEMPLO
```

Figura 8-1. Programa ejemplo "PRACT8A.ASM".

8.3. Programas propuestos para el uso de la pantalla en modo gráfico

8.3.1. *Programa H.1*

Partiendo del programa anterior "PRACT8A.ASM", realizar un programa que sitúe un gráfico de la letra A en cualquier punto de la pantalla. El gráfico de la letra A estará definido de la siguiente forma:

```
GRAFICO     DB 00000000b
            DB 00011000b
            DB 00100100b
            DB 01000010b
            DB 01000010b
            DB 01000010b
            DB 01111110b
            DB 01000010b
            DB 01000010b
            DB 00000000b
```

Figura 8-2. Gráfico del carácter ASCII "A".

Esta letra A se podrá desplazar de pixel en pixel mediante las teclas de cursor. Si se pulsa la tecla "ESCAPE" se sale del programa, poniéndose la pantalla en modo texto antes de salir.

8.3.2. *Programa H.2*

Realizar una función que dibuje una línea dados sus puntos inicial y final usando el algoritmo de Bresenham (ver apéndice F del libro).

8.3.3. *Programa H.3*

Dibujar una gráfica en pantalla con los ejes de ésta. La gráfica dibujará los datos cualesquiera de un *array* de 640 enteros.

8.3.4. *Programa H.4*

Dibujar en ensamblador la pantalla de un osciloscopio dividida en cuadrículas. Dibujar las líneas de los ejes con 5 "muescas" por cuadrícula.

8.3.5. *Programa H.5*

Crear un gráfico y su máscara como el de la figura 8.3.

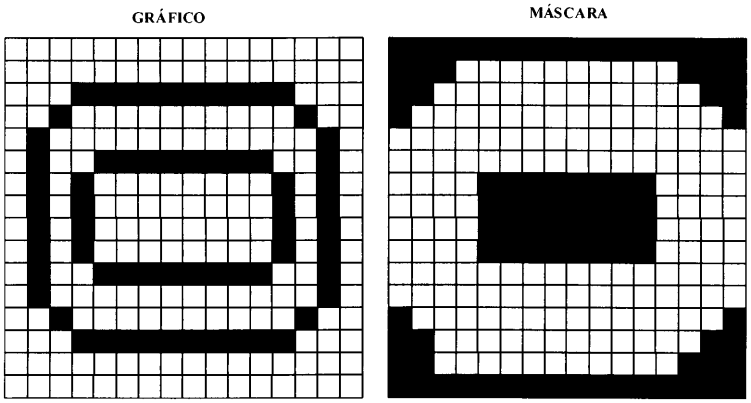


Figura 8-3. Gráfico de un "Donut" y su máscara.

La máscara sirve para hacer un AND con el fondo y después una OR con el gráfico; usando la máscara podemos hacer que el objeto sea "transparente" donde queramos.

El gráfico se **moverá en pantalla pixel a pixel**, mediante el uso de los **cursores**.

La forma de operar es la siguiente:

1º Se coge el cuadrado del fondo de la pantalla donde vamos a poner el gráfico y se almacena en memoria.

2º Se hace una AND de ese fondo con la máscara, queda en blanco los huecos de la máscara.

3º Hacemos una OR del gráfico con el resultado del punto 2, y se imprime en pantalla. Quedará el gráfico superpuesto al fondo de la pantalla, viéndose el fondo a través de las zonas transparentes del gráfico.

4º Para poner el gráfico en otra posición, primero se restaura el fondo original (almacenado en el punto 1º), y se empieza de nuevo desde el principio para esa nueva posición.

8.3.6. *Programa H.6*

Tenemos un sensor analógico de temperatura que nos genera un valor de tensión de 0 a 10 voltios. La tensión la introducimos en una entrada analógica 0 a 10 voltios, con un conversor A/D de 10 bits. Realizar un programa **que nos pregunte un valor digital de 10 bits cualquiera** y nos diga el valor de tensión que es y el valor de temperatura al que corresponde. (Generaremos una tabla con los 2^{10}=1024 valores posibles de temperatura).

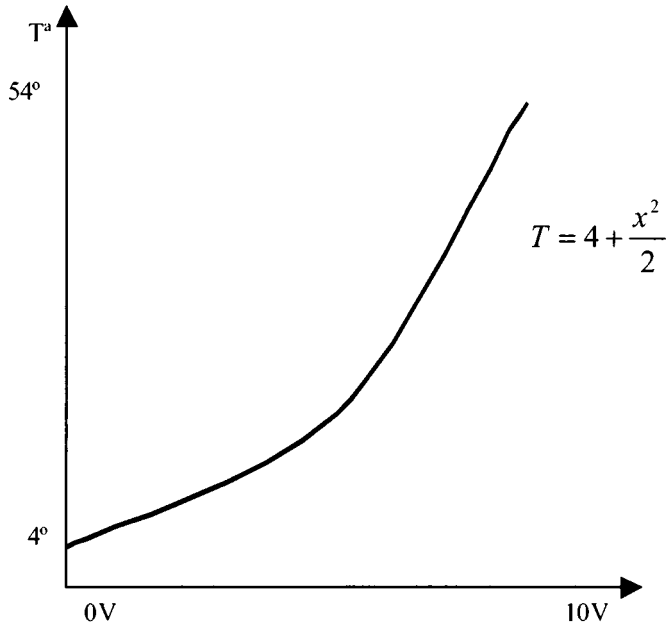


Figura 8-4. Curva tensión-temperatura de un sensor ejemplo.

8.3.7. *Programa H.7*

Realizar lo mismo que en el ejercicio H.6 para un sensor donde sólo tenemos los valores de 10 puntos. Crear la tabla de los 1024 puntos, interpolando según estos 10 puntos.

Puntos:

0 Voltios	5º C
1 Voltio	10º C
2 Voltios	15º C
3 Voltios	25º C
4 Voltios	40º C
5 Voltios	55º C
6 Voltios	65º C
7 Voltios	85º C
8 Voltios	100º C
9 Voltios	130º C
10 Voltios	160º C

Dibujar en una gráfica en pantalla con sus ejes esta tabla. Modo Gráfico.

8.3.8. *Programa H.8*

Dado un *buffer* de 200 bytes de datos cualesquiera, realizar el programa que dibuje a modo de "Analizador Lógico" los bits de cada byte.

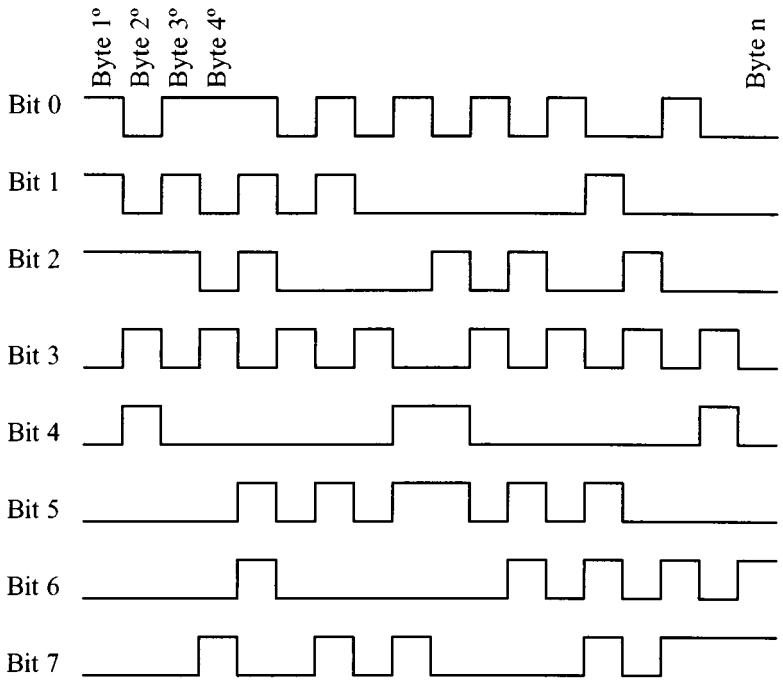


Figura 8-5. Ejemplo tipico de un programa que realiza la función de un analizador lógico.

8.3.9. *Programa H.9*

Dibujar un cubo en tres dimensiones con líneas y girarlo en el espacio lo más suavemente posible. Modo gráfico.

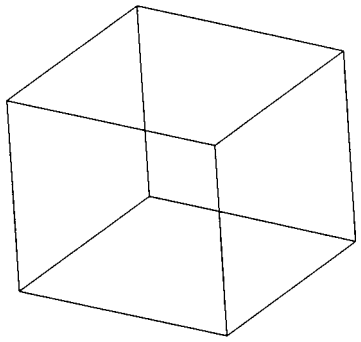


Figura 8-6. Cubo en tres dimensiones.

8.3.10. *Programa H.10*

Realizar un programa que rellene de puntos cualquier figura cerrada, dado un punto de inicio (típico comando de llenado *(fill)*).

PRÁCTICA 9

9. Manejo de Ficheros

9.1. El File Handle

A partir del DOS 2.0, el manejo de archivos es muy sencillo. Simplemente a cada archivo abierto se le designa un número (*File Handle*), de forma que para escribir o leer le indicamos a la función correspondiente el *Handle* del fichero abierto.

9.2. Forma de manejar los ficheros

Lo primero que hay que hacer para manejar un fichero es abrirlo o crearlo si no existe. Para ello existen diferentes funciones, las más sencillas son:

- **Función AH=3Ch**: Crear archivo.
- **Función AH=3Dh**: Abrir archivo.

Luego procedemos a situarnos en la posición que nos interese dentro del archivo (al principio, al final, en una posición concreta, etc.)

- **Función AH=42h**: Situar puntero de lectura/escritura.

Una vez situados en la posición, procedemos a grabar o leer la información, siempre en bloques no superiores al tamaño del buffer.

- **Función AH=3Fh**: Leer fichero.
- **Función AH=40h**: Escribir en un fichero.

Y por último, y muy importante, **cerrar el archivo antes de salir**.

- **Función AH=3Eh**: Cerrar fichero.

Queda la posibilidad de borrar el archivo con la función:

- **Función AH=41h**: Borrar fichero.

9.3. Programa ejemplo PRACT9A.ASM. Abrir un archivo de datos para añadir información

El programa siguiente, abre un archivo de datos y si no existe lo crea. Se sitúa al final del mismo y le añade información. Entender el funcionamiento del mismo.

```
;
;PROGRAMA EJEMPLO PRACT9A.ASM: ABRIR O CREAR UN ARCHIVO Y AÑADIRLE INFOR-
MACIÓN
;-----------------------------------------------
CR     EQU   13        ;Retorno de carro
LF     EQU   10        ;Salto de línea
;




;-----------------------------------------------
;
;Segmento de Datos
;-----------------
;
DATOS SEGMENT         ;Comienzo segmento DATOS
ERROR1 DB 'ERROR AL ABRIR EL ARCHIVO',CR,LF,'$'
ERROR2 DB 'ERROR AL CREAR EL ARCHIVO',CR,LF,'$'
ERROR3 DB 'ERROR AL INTENTAR GRABAR LOS DATOS EN EL ARCHIVO',CR,LF,'$'
ERROR4 DB 'ERROR AL CERRAR EL ARCHIVO',CR,LF,'$'
ERROR5 DB 'ERROR AL POSICIONARSE EN EL ARCHIVO',CR,LF,'$'
MENSAJE_BIEN DB 'SE HAN GRABADO BIEN LOS DATOS',CR,LF,'$'

NAME_A DB 'MI_ARC.TXT',0           ;NOMBRE DEL ARCHIVO
HANDLE DW 0

BUFFER DB ';123456789'             ;DATOS A GUARDAR
DATOS ENDS
;
;-----------------------------------------------
;
;Segmento de Pila
;----------------
;
PILA SEGMENT STACK  ;Comienzo segmento PILA
     DB 128 DUP('PILA')    ;Inicialización PILA
PILA ENDS
;
;
;Segmento de Código
;----------------
;
CODIGO    SEGMENT     ;Comienzo segmento CODIGO
EJEMPLO   PROC FAR
          ASSUME   CS:CODIGO,DS:DATOS,SS:PILA
;
          PUSH DS     ;Guarda Segmento en pila (DS:AX dirección retorno)
          SUB AX,AX   ;Borrar registro AX=0
          PUSH AX     ;Guarda en Pila AX (IP=0)
;
          MOV AX,DATOS     ;AX=DATOS (SEGMENTO DE DIRECCION DATOS)
          MOV DS,AX        ;DS=AX

;         ################################################
;         #    ABRIMOS EL ARCHIVO, SI NO EXISTE SE CREA  #
;         ################################################
```

```
          MOV AH,3Dh       ;FUNCION DE ABRIR ARCHIVO
          MOV AL,02h       ;ABRIR LECTURA Y ESCRITURA
          LEA DX,NAME_A    ;NOMBRE DEL ARCHIVO
          INT 21h

          JNC ABIERTO      ;SI NO HAY ERROR CONTINÚA

          CMP AX,2
          JE NO_EXISTE     ;SALTA A NO_EXISTE
          CMP AX,3
          JE NO_EXISTE     ;SALTA SI NO EXISTE EL DIRECTORIO O ARCHIVO

          LEA DX,ERROR1
          CALL ESCRIBIR
          RET


NO_EXISTE:
          MOV AH,3Ch       ;FUNCION PARA CREAR EL ARCHIVO
          MOV CX,0         ;ATRIBUTOS NORMALES
          LEA DX,NAME_A    ;NOMBRE DEL ARCHIVO A CREAR
          INT 21h

          JNC ABIERTO      ;SE HA CREADO EL ARCHIVO

          LEA DX,ERROR2
          CALL ESCRIBIR
          RET

;         ##########################################
;         #     NOS SITUAMOS AL FINAL DEL ARCHIVO    #
;         ##########################################

ABIERTO:
          MOV [HANDLE],AX       ;GUARDAMOS EL HANDLE DEL ARCHIVO


          MOV AH,42h            ;FUNCION PARA MOVER EL PUNTERO EN EL
ARCHIVO
          MOV AL,02h            ;MOVER PUNTERO A PARTIR DEL FINAL DEL
ARCHIVO
          MOV BX,[HANDLE]       ;BX=AL HANDLE
          XOR CX,CX             ;CX=0
          XOR DX,DX             ;DX=0      CX,DX=DESPLAZAMIENTO
          INT 21h

          JNC POSIC

          LEA DX,ERROR5
          CALL ESCRIBIR
          RET

;         ##########################################
;         #     GUARDAMOS LOS DATOS EN EL ARCHIVO    #
;         ##########################################
```

```
POSIC:      LEA DX,BUFFER       ;DIRECCION DATOS A GUARDAR
            MOV CX,10           ;NUMERO DE DATOS
            MOV AH,40h          ;FUNCION DE GUARDAR DATOS
            MOV BX,[HANDLE]     ;BX=HANDLE DEL ARCHIVO
            INT 21h

            JNC GRABADOS        ;SE HAN GRABADO LOS DATOS

            LEA DX,ERROR3
            CALL ESCRIBIR
            RET

;           ############################
;           #    CERRAMOS EL ARCHIVO   #
;           ############################

GRABADOS:   MOV AH,3Eh          ;FUNCION DE CERRAR FICHERO
            MOV BX,[HANDLE]     ;BX=HANDLE DEL ARCHIVO
            INT 21h

            JNC SALIR

            LEA DX,ERROR4
            CALL ESCRIBIR
            RET
SALIR:
            LEA DX,MENSAJE_BIEN
            CALL ESCRIBIR   ;SUBRUTINA DE ESCRIBIR TEXTO
            RET             ;RETORNA
;
EJEMPLO     ENDP            ;FIN DE PROCEDIMIENTO
;
ESCRIBIR    PROC            ;PROCEDIMIENTO 'ESCRIBIR'
            PUSH AX         ;GUARDA EN PILA AX
            MOV AH,9        ;AH=9  FUNCION NUMERO 9 'SALIDA DE CARACTERES'
            INT 21H         ;LLAMADA A INTERRUPCION DEL DOS, CON FUNCION 9
            POP AX          ;RECUPERA EL REGISTRO AX
            RET             ;RETORNAR
ESCRIBIR    ENDP            ;FIN DE PROCEDIMIENTO ESCRIBIR
CODIGO      ENDS            ;FIN DE SEGMENTO DE CODIGO
            END EJEMPLO     ;FIN DE PROGRAMA EJEMPLO
```

Figura 9-1. Programa ejemplo "PRACT9A.ASM".

9.4. Manejo de directorios mediante funciones del DOS

Las funciones de manejo de directorios son (ver página 549 a 554):

- **Función AH=39h**: Crear subdirectorio dentro del directorio activo.
- **Función AH=3Ah**: Borrar subdirectorio.
- **Función AH=3Bh**: Cambiar el subdirectorio.
- **Función AH=47h**: Obtener el directorio activo.

9.5. Búsqueda de archivos o directorios

El modo de búsqueda de archivos es igual al modo en que se realiza en C. Existen dos funciones:

- **Función AH=4Eh (FindFirst):** Buscar el primer archivo que coincida con el nombre a buscar (pueden incluirse asteriscos o interrogantes). En DS:DX ponemos el patrón de búsqueda y en CX el atributo de los archivos a buscar.

```
CX:         bit 0=1     Solo Lectura (ReadOnly)
            bit 1=1     Oculto (Hidden)
            bit 2=1     Sistema (System)
            bit 3=1     Etiqueta de volumen
            bit 4=1     Subdirectorio
            bit 5=1     Bit de Archivo
```

Figura 9-2. Significado de los bits del parámetro de entrada CX.

- **Función AH=4Fh (FindNext):** Buscar el siguiente nombre de archivo

9.6. Programa ejemplo PRACT9B.ASM. Saca en pantalla el nombre y los atributos de todos los archivos del directorio

El programa ejemplo PRACT9B.ASM, busca todos los archivos del directorio indicado y los escribe en pantalla junto con sus atributos. Entender el funcionamiento del programa.

```
;
;PROGRAMA EJEMPLO PRACT9B.ASM:
; ESCRIBIR EN PANTALLA LOS ARCHIVOS DEL DIRECTORIO INDICADO
;-------------------------------------------------
CR     EQU   13        ;Retorno de carro
LF     EQU   10        ;Salto de línea
;
;-------------------------------------------------
;
;Segmento de Datos
;-----------------
;

DATOS SEGMENT         ;Comienzo segmento DATOS

ARCHIV_BUF      DB 21 DUP ('?')     ;BUFFER DEL ARCHIVO A LEER
ATTRIB          DB 0                ;ATRIBUTOS
HORA            DW 0                ;HORA
FECHA           DW 0                ;FECHA
TAM_INF         DW 0                ;WORD INFERIOR DEL TAMA¥O
TAM_SUP         DW 0                ;WORD SUPERIOR DEL TAMA¥O
NOMBRE_F        DB 13 DUP ('?')     ;NOMBRE CON EXTENSION DEL ARCHIVO

BUSC            DB 'c:\*.*',0       ;FICHEROS A BUSCAR

SALTOLIN    DB CR,LF,0              ;SALTO DE LINEA
TEXTO       DB 'BUSCANDO:',0
LONG        DB 0                    ;LONGITUD DEL NOMBRE
```

```
DATOS ENDS
;
;-----------------------------------------------------
;
;Segmento de Pila
;--------
;
PILA SEGMENT STACK  ;Comienzo segmento PILA
     DB 128 DUP('PILA')    ;Inicialización PILA
PILA ENDS
;
;-----------------------------------------------------
;
;Segmento de C¢digo
;------------------
;
CODIGO      SEGMENT     ;Comienzo segmento CODIGO
EJEMPLO     PROC FAR
            ASSUME      CS:CODIGO,DS:DATOS,SS:PILA
;
            PUSH DS     ;Guarda Segmento en pila (DS:AX dirección retorno)
            SUB AX,AX   ;Borrar registro AX=0
            PUSH AX     ;Guarda en Pila AX (IP=0)
;
            MOV AX,DATOS    ;AX=DATOS (SEGMENTO DE DIRECCION DATOS)
            MOV DS,AX       ;DS=AX

;

;           ################################################
;           #    INICIALIZAMOS LA DIRECCION DEL DTA        #
;           #             DS:DX=DIRECCION                  #
;           ################################################
            LEA DX,ARCHIV_BUF   ;DX=DIRECCION DEL BUFFER DE ARCHIVO
            MOV AH,1Ah          ;INTERRUPCION DE CAMBIAR DIRECCION
            INT 21h

            LEA BX,TEXTO
            CALL ESCRIBIR0      ;PONE EN PANTALLA 'BUSCANDO:'
            LEA BX,BUSC
            CALL ESCRIBIR0      ;PONE EN PANTALLA ARCHIVOS A BUSCAR
            LEA BX,SALTOLIN
            CALL ESCRIBIR0      ;SALTO DE LINEA
            LEA BX,SALTOLIN
            CALL ESCRIBIR0      ;SALTO DE LINEA


;           ################################################
;           #      BUSCAMOS EL PRIMER ARCHIVO              #
;           ################################################

            LEA DX,BUSC           ;DX=DIRECCION DE NOMBRE DE ARCHIVO
            MOV AH,4Eh            ;FUNCION BUSCAR EL PRIMER ARCHIVO
            MOV CL,00111111b      ;BUSCAR ARCHIVOS DE TODO TIPO
            MOV CH,0
            INT 21h
```

```
OTRO_A:     CMP AX,18           ;SI AX=18 ARCHIVO NO ENCONTRADO
            JE SALIR
            CMP AX,2            ;SI AX=2 DIRECTORIO NO ENCONTRADO
            JE SALIR

            LEA BX,NOMBRE_F     ;BX=DIRECCION DEL NOMBRE DEL ARCHIVO
            CALL ESCRIBIR0      ;IMPRIMIR TEXTO

            MOV AH,02           ;FUNCION IMPRIMIR CARACTER
            MOV DL,9            ;CARACTER TABULACIàN
            INT 21h
            CMP [LONG],8        ;COMPARA LA LONGITUD DEL NOMBRE CON 8
            JAE NO_TAB          ;SI ES MAYOR O IGUAL SALTA, SINO PON OTRO TAB

            MOV DL,9            ;CARACTER TABULACIàN
            INT 21h


NO_TAB:     MOV AL,[ATTRIB]     ;COGE EL ATRIBUTO
            AND AL,00100000b    ;COMPARA EL BIT 5
            JZ BUC1A
            MOV DL,'A'          ;IMPRIME A DE 'ARCHIVO'
            INT 21h

BUC1A:      MOV AL,[ATTRIB]
            AND AL,00010000b    ;COMPARA EL BIT 4
            JZ BUC1B
            MOV DL,'D'          ;IMPRIME D DE 'DIRECTORIO'
            INT 21h

BUC1B:      MOV AL,[ATTRIB]
            AND AL,00001000b    ;COMPARA EL BIT 3
            JZ BUC1C
            MOV DL,'E'          ;IMPRIME E DE 'ETIQUETA DE VOLUMEN'
            INT 21h

BUC1C:      MOV AL,[ATTRIB]
            AND AL,00000100b    ;COMPARA EL BIT 2
            JZ BUC1D
            MOV DL,'S'          ;IMPRIME S DE 'SISTEMA'
            INT 21h

BUC1D:      MOV AL,[ATTRIB]
            AND AL,00000010b    ;COMPARA EL BIT 1
            JZ BUC1E
            MOV DL,'H'          ;IMPRIME H DE 'HIDDEN (OCULTO)'
            INT 21h

BUC1E:      MOV AL,[ATTRIB]
            AND AL,00000001b    ;COMPARA EL BIT 0
            JZ BUC1F
            MOV DL,'R'          ;IMPRIME R DE 'READONLY (SOLO LECTURA)'
            INT 21h

BUC1F:      LEA BX,SALTOLIN     ;IMPRIME SALTO DE LINEA
            CALL ESCRIBIR0
```

```
            MOV AH,4Fh      ;FUNCION FINDNEXT (COGER EL SIGUIENTE ARCHIVO)
            INT 21h
            JMP OTRO_A

SALIR:      RET             ;RETORNA
;
EJEMPLO     ENDP            ;FIN DE PROCEDIMIENTO
;
ESCRIBIR0   PROC            ;PROCEDIMIENTO 'ESCRIBIR0' CADENA DE CARACTERES
                            ;TERMINADA EN 0
            MOV [LONG],0    ;LONGITUD DEL TEXTO A IMPRIMIR 0
BUC_E1:     MOV DL,[BX]     ;METE EN DL EL CARACTER A IMPRIMIR
            CMP DL,0        ;ES CERO, SI ES CERO SAL
            JE SAL_PRINT

            MOV AH,02       ;IMPRIME EL CARACTER
            INT 21H
            INC BX          ;INCREMENTA EL PUNTERO BX
            INC [LONG]      ;INCREMENTA LA LONGITUD DE TEXTO IMPRIMIDO
            JMP BUC_E1

SAL_PRINT: RET

ESCRIBIR0   ENDP            ;FIN DE PROCEDIMIENTO ESCRIBIR0
CODIGO      ENDS            ;FIN DE SEGMENTO DE CODIGO
            END EJEMPLO     ;FIN DE PROGRAMA EJEMPLO
```

Figura 9-3. Programa ejemplo "PRACT9B.ASM"

9.7. Programas propuestos para el aprendizaje y uso de las funciones de manejo de archivos

9.7.1. *Programa I.1*

Realizar un programa que nos abra un fichero cualquiera '.TXT' y cuente el número de vocales que hay, indicando al final de la cuenta cuántas vocales hay en A, E, I, O y U dentro del archivo.

9.7.2. *Programa I.2*

Realizar un programa que nos pida el NOMBRE (30 bytes), DIRECCION (28 bytes), EDAD (3 bytes) y DNI (9 bytes) y grabe esta información en una archivo FICHAS.DAT.

9.7.3. *Programa I.3*

Realizar un programa que nos pida el DNI y busque en el archivo del punto anterior (FICHAS.DAT) la ficha correspondiente a ese DNI, y la presente en pantalla.

9.7.4. *Programa I.4*

Desarrollar un programa que nos busque una palabra cualquiera dentro de un archivo de texto, devolviéndonos la distancia que hay desde el principio del archivo hasta la palabra encontrada o dándonos un error si no la encuentra.

9.7.5. *Programa I.5*

Realizar un programa, con las funciones anteriormente indicadas, que nos busque en todo el disco duro un archivo que previamente le indicamos.

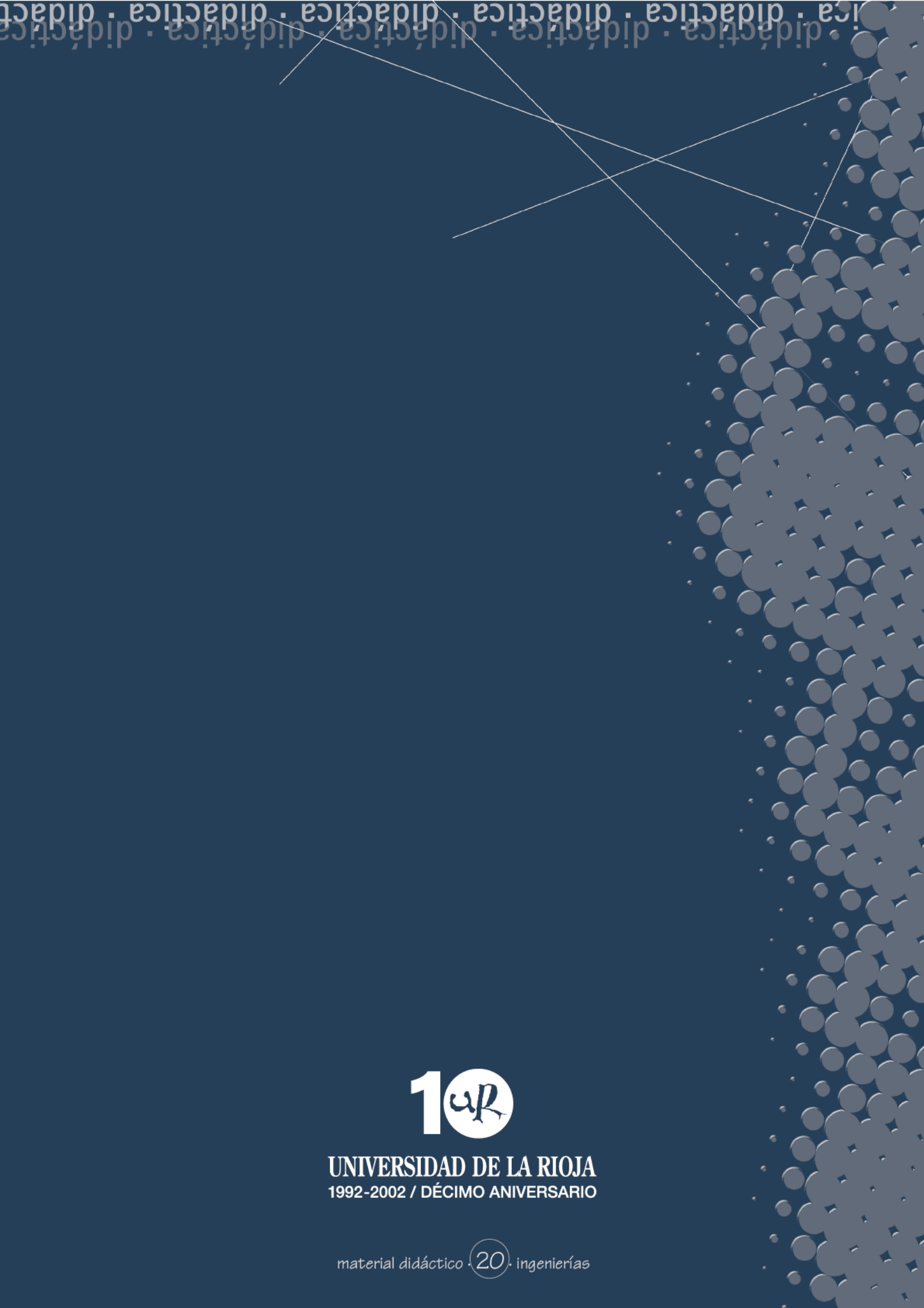
9.7.6. *Programa I.6*

Realizar un programa que cuente en un disco duro cuántos archivos ocultos tiene y el número de bytes total que ocupan.

9.7.7. *Programa I.7*

Realizar un programa que busque en los archivos con extensión '.txt' una cadena cualquiera previamente establecida y nos visualice en pantalla el nombre de todo archivo que la tenga así como el número de línea en la qué está.

ÍNDICE


UNIVERSIDAD DE LA RIOJA
1992-2002 / DÉCIMO ANIVERSARIO
material didáctico · 20 · ingenierías