ESCUELA POLITECNICA NACIONAL

FACULTAD DE INGENIERIA ELECTRICA

ESPECIALIZACION EN ELECTRONICA Y
TELECOMUNICACIONES

DEPURADOR DE PROGRAMAS PARA EL MICROPROCESADOR 80486 (PARTE 3)

JORGE BOLIVAR OSORIO HINOJOSA
ALEX JOHNINE TROYA ALDAZ

TESIS PREVIA A LA OBTENCION DEL TITULO DE INGENIERO EN LA ESPECIALIZACION DE INGENIERIA ELECTRONICA Y TELECOMUNICACIONES DE LA ESCUELA POLITECNICA NACIONAL

MAYO 1997

1140S

APENDICE G

ARCHIVOS FUENTE (.ASM) DE LOS EJEMPLOS DE DEPURACION

APENDICE G

FUNCION.ASM

```
;ejemplo de programa interactivo dirigido por menú para obtener las funciones
;seno, coseno y tangente de un ángulo. (para coprocesadores 80387 en adelante)
;
        dosseg
        .model small
        .386
        .387
        .stack  64h

;_____________________________________________________
;       AREA PARA DATOS
;_____________________________________________________

        .data

menu1   db ' ESCOJA OPCION INGRESANDO UNO DE ESTOS
CARACTERES:  ',10,13
        db '                                    ',10,13
        db '                                    ',10,13
        db '            S- OBTENER SENO DEL ANGULO (0-90°)',10,13
        db '                                    ',10,13
        db '            C- OBTENER COSENO DEL ANGULO (0-90°)',10,13
        db '                                    ',10,13
        db '            T- OBTENER TANGENTE DEL ANGULO (0-90°)',10,13
        db '                                    ',10,13
        db '            E- FIN DEL PROGRAMA     ',10,13
        db '                                    ',10,13
        db '$'

Menu2   db '        INGRESE VALOR ANGULO (.000° - 90.0°):       $'

menu3   db '                RESULTADO =  $'
menu4   db '          PRESIONE UNA TECLA PARA CONTINUAR $'
infinito db '             RESULTADO =  INFINITO$'
Value   db   ?
noventa db   0
seleccion db ?                  ;recibe opción de menú
hex     dd   0                  ;almacena ángulo en hexadecimal (grados)
num1    db   0                  ;recibe el ángulo en decimal (grados)
        db   0
        db   0
        db   0
tecla   db   0                  ;indica si se presionó ESC o ENTER
```

```
posicion   db    0              ;indica posición de la coma en ángulo
ans        dq    0              ;almacena resultado en hexadecimal
milmillon  dd    1000000000     ;constantes
millon     dq    100000000
const1     dd    180000
const2     dd    18000
const3     dd    1800
diez       db    10
deci       dt    0              ;almecena el resultado en decimal

printnum   macro                ;imprime numero ASCII en pantalla
           pusha                ;guarda registros
           and   al,0fh
           or    al,30h         ;lo convierte a ASCII
           mov   dl,al
           mov   ah,2           ;inicializa parametros de interrupcion
           int   21h            ;llama la interrupción
           popa
           endm

arith  macro                    ;macro aritmètica
           push  ax
           push  cx
           finit                ;inicializa coprocesador
           fild  hex            ;carga valor entero hex en ST(0)
           cmp   posicion,0     ;punto decimal al inicio?
           jne   nomil
           fidiv const1         ;divide ST(0) para const1 (180000) pa-
           jmp   hallarad       ;ra convertir a radianes y recorrer la
nomil:                          ;coma decimal
           cmp   posicion,1
           jne   nocien
           fidiv const2
           jmp   hallarad
nocien:
           fidiv const3         ;divide ST(0) para const3 (1800)
hallarad:
           fldpi                ;cargar valor PI en ST(0)
           fmul                 ;multiplicar ST(0) con ST(1)
                                ;se obtiene ángulo en radianes
           cmp   seleccion,'s'  ;se desea obtener seno de ángulo?
           je    seno
           cmp   seleccion,'S'
           je    seno
           cmp   seleccion,'C'  ;se desea obtener coseno de ángulo?
           je    coseno
           cmp   seleccion,'c'
```

```
        je      coseno
        cmp     seleccion,'T'       ;se desea obtener tangente de ángulo?
        je      tangente
        cmp     seleccion,'t'
        je      tangente

seno:
        fsin                        ;obtener seno del ángulo (radianes)
                                    ;contenido en ST(0)
        fild    millon              ;cargar constante millon en ST(0)
        fmul    st,st(1)            ;multiplicar ST(0) con ST(1) para
                                    ;convertir valor decimal a entero
        fistp   ans                 ;sacar valor entero a la variable ANS
        jmp     fin

coseno:
        fcos                        ;obtener coseno del ángulo (radianes)
        fild    millon
        fmul    st,st(1)
        fistp   ans
        jmp     fin

tangente:                           ;obtener tangente del ángulo
        cmp     hex,900             ;valor del ángulo = 90°?
        jne     otro
        mov     noventa,0ffh        ;indica que es tangente de 90°

        mov     bh,3fh              ;atributo color
        clearscreen                 ;macro para limpiar la pantalla

        cursor  0c00h               ;macro para ubicar cursor en
                                    ;fila 12, columna 0
        mov     ah,9h               ;presenta resultado = infinito para
        mov     dx,offset infinito  ;tangente de 90°
        int     21h                 ;llama interrupción
        jmp     fin2
otro:   fsincos                     ;obtener seno y coseno de ST(0)
        fdiv    st(1),st            ;divide ST(0) para ST(1) (Seno/Coseno)
                                    ;resultado se almacena en ST(1)
        fistp   ans                 ;sacar valor entero a ANS
        fild    millon              ;cargar valor entero millon a ST(0)
        fmul    st,st(1)            ;multiplicar ST(0) por ST(1)
        fistp   ans                 ;sacar valor entero (resultado) a ANS

        call    convdec             ;rutina conversión a decimal del
                                    ;resultado de la tangente
        mov     bh,3fh              ;atributo color
```

```
        clearscreen                     ;macro para limpiar pantalla
        cursor    0c00h
        printchar menu3                 ;presentar menu3 en pantalla

        mov       si,offset deci        ;presenta valor entero de tangente
        mov       bx,5
        mov       al,[si+bx]
        add       al,30h
        mov       dl,al
        mov       ah,2
        int       21h
        dec       bx
        mov       al,[si+bx]
        mov       value,al
        call      printit
        dec       bx
        mov       noventa,0eh           ;indica operación fue tangente

        jmp       fin2
fin:    call      convdec               ;rutina conversión a decimal del
fin2:   pop       cx                    ;resultado de seno o coseno
        pop       ax
        endm

readnumero macro  thisnum,valor         ;lee de teclado datos de acuerdo a valor
        local     otra,fin,equal        ;define variables locales
        push      ax                    ;guarda registros afectados
        push      cx
        push      bx
        push      dx
        mov       tecla,0               ;indica no es ENTER ni ESC
        mov       posicion,0ffh
        mov       bx,0
        mov       cx,valor              ;número de dígitos
        mov       si,offset thisnum     ;ubicar datos en thisnum
        mov       al,0
llenar:
        mov       [si+bx],al
        inc       bx
        loop      llenar                ;borra contenido de thisnum
        mov       cx,valor
        mov       bx,0
otra:   cmp       posicion,bl
        jne       leertec
        mov       posicion,0ffh
leertec:
        mov       ah,01h                ;inicializa parámetros
```

```
        int     21h                     ;lee interrupción tecla
        jmp     verificar               ;salta a verificar si tecla válida
equal1: sub     al,30h
equal:  mov     [si+bx],al              ;guardo valor en decimal
        inc     bx
        loop    otra                    ;salta a leer otra tecla
        cursor  170Ah
        printchar menu4
        mov     ah,0
        int     16h
        jmp     fin
verificar:
        cmp     al,00h                  ;verifica si es codigo extendido
        jne     noextendido
        call    caratras                ;mueve cursor un caracter atras
        jmp     otra
noextendido:
        cmp     al,1bh
        jne     next0
        mov     tecla,0ffh              ;escape
        jmp     fin
next0:  cmp     al,0dh
        jne     next1
        mov     tecla,1                 ;enter
        jmp     fin

next1:
        cmp     al,08h                  ;back space?
        jne     next2
        cmp     bx,0
        jne     next3
                                        ;hacer BEEP
        mov     ah,2
        mov     dl,7
        int     21h
        call    cardel                  ;mover cursor una posición adelante
        jmp     otra

next3:  dec     bx
        inc     cx
        jmp     otra
next2:  cmp     al,2eh                  ; es un "." decimal?
        jne     nocoma
decimal:
        cmp     posicion,0ffh
        jne     mal
        cmp     bx,3
```

```
        jae     mal
        mov     posicion,bl
        jmp     equal
nocoma:
        cmp     al,2ch          ;es una "," decimal?
        je      decimal
        cmp     al,30h
        jb      mal
        cmp     al,39h
        ja      mal
        jmp     equal1
mal:                            ;BEEP (caracter no aceptado)
        mov     ah,2
        mov     dl,7
        int      21h
        call    caratras
        call    cardel
        call    caratras
        jmp     otra

caratras:                       ;mover cursor una posición atrás
        mov      dl,08h
        mov     ah,2
        int     21h
        retn
cardel:                         ;mover cursor una posición adelante
        mov     dl,20h
        mov     ah,2
        int     21h
        retn

fin:  pop     dx
        pop     bx
        pop     cx
        pop     ax
        endm

readkey  macro  thiskey         ;lee un dato del teclado
        push    ax              ;guarda registro afectado
        mov     si,offset thiskey
        mov     ah,01h          ;inicializa parámetros
        int     21h             ;lee interrupción tecla
        mov     thiskey,al      ;guarda tecla devuelta en variable
        pop     ax
        endm

clearscreen     macro           ;borra pantalla color
```

```
        pusha
        mov     cx,0                    ;inicializa esquina superior de ventana
        mov     dx,2479h                ;inicializa esquina inferior de ventana
        mov     ax,0600h                ;parámetros de interrupción
        int     10h                     ;llama interrupción
        popa
        endm

cursor  macro   locate                  ;desplaza cursor (localizar)
        pusha                           ;guarda registros
        mov     ah,15                   ;obtiene pantalla actual
        int     10h                 ;llamada interrupción, inicializa valores BX
        mov     dx,locate               ;desplaza posición de pantalla a AX
        mov     ah,2                    ;inicializa parámetro de cursor
        int     10h                     ;llama a la interrupción
        popa
        endm

printchar       macro text      ;imprime caracteres del segmento de datos cadena
        pusha                           ;a ser pasada
        lea     dx,text                 ;en (texto) se imprime hasta ($),
        mov     ah,9                    ;se localiza. imprime desde posición
        int     21h                     ;actual de cursor
        popa
        endm
```

```
;_____________________________________________
;       AREA PARA CODIGO
;_____________________________________________

  .code
        mov     ax,@data                ;dirección segmento datos a AX
        mov     ds,ax                   ;valor AX al registro de segmento


mmenu:  mov     bh,70h
        clearscreen                     ;borra pantalla
        cursor  0c00h                   ;llama macro cursor
        printchar  menu1                ;imprime menú principal

;lee teclado para operación aritmètica correcta
key:    readkey seleccion               ;lee opción de menú
        cmp     seleccion,'E'           ;salir del programa?
```

```
        je      final                   ;si no, continuar
        cmp     seleccion,'e'
        je      final
        cmp     seleccion,'S'           ;verificar si opción váliva
        je      getnu
        cmp     seleccion,'s'
        je      getnu
        cmp     seleccion,'C'
        je      getnu
        cmp     seleccion,'c'
        je      getnu
        cmp     seleccion,'T'
        je      getnu
        cmp     seleccion,'t'
        je      getnu
        jmp     mmenu

getnu:
        mov     bh,70h
        clearscreen                     ;borra el menú
        cursor  0c00h                   ;desplazar cursor
        printchar menu2                 ;preguntar por ángulo

;leer primer numero
        readnumero num1,4               ;lee entrada de teclado
        xor     eax,eax                 ;convertir ángulo a hexadecimal
        cmp     tecla,0ffh              ;eliminando el punto decimal
        je      mmenu
        mov     bx,0
        cmp     posicion,0
        jne     moveprim
        inc     bx
moveprim:
        mov     ah,[si+bx]
        inc      bx
        cmp     posicion,1
        jne     movesec
        inc     bx
movesec:
        mov     al,[si+bx]
        inc      bx
        cmp     posicion,2
        jne     moveter
        inc     bx
moveter:
        mov     cl,[si+bx]
        aad
```

```
        mov     bx,10
        mul     bx
        xor     ch,ch
        add     ax,cx
        cmp     ax,900                  ;compara si ángulo > 90°
        jbe     bien
        mov     ah,2
        mov     dl,7
        int      21h
        jmp     getnu
bien:
        mov     hex,eax                 ;proceder con arithmetic
        mov     noventa,0
        arith                           ;hace operación seno, coseno o tangente
        cmp     noventa,0ffh
        je      fin4
        cmp     noventa,0eh
        je      fin5
        mov     bh,3fh
        clearscreen                     ;borra el menú
        cursor  0c00h                   ;desplazar cursor
        printchar menu3                 ;preguntar por angulo
        mov     si,offset deci          ;presenta parte entera de seno o coseno
        mov      bx,4
        mov     al,[si+bx]
        add     al,30h
        mov     dl,al
        mov     ah,2
        int     21h
        dec     bx
fin5:
        mov     cx,4                    ;presenta parte decimal para todos
        mov     ah,2                    ;los casos
        mov     dl,2ch
        int     21h
sig:
        mov     al,[si+bx]
        mov     value,al
        call    printit
        dec     bx
        loop    sig
fin4:  cursor   170Ah
        printchar menu4                 ;esperar por tecla
        mov     ah,0
        int     16h
        jmp     mmenu                   ;bifurca a menu principal y repite
convdec:                                ;rutina conversión a decimal
```

```
mov     si,offset ans
mov     eax,[si] ;ans
mov     edx,[si+4]
div     milmillon ;ebx
mov     di,offset deci ;mantisf
push    eax
xor     eax,eax
mov     [di],eax                ;borro el contenido de mantisa
pop     eax
div     diez
mov     byte ptr [di+5],al
shl     ah,4
mov     byte ptr [di+4],ah

mov     eax,edx                 ;reciduo a eax
mov     edx,0
mov     ebx,100000000
div     ebx
add     [di+4],al


mov     eax,edx                 ;reciduo a eax
mov     edx,0
mov     ebx,10000000
div     ebx
shl      eax,28
add     [di],eax

mov     eax,edx
mov     edx,0
mov     ebx,1000000
div     ebx
shl     eax,24
add     [di],eax

mov     eax,edx
mov     edx,0
mov     ebx,100000
div     ebx
shl     eax,20
add     [di],eax

mov     eax,edx
mov     edx,0
mov     ebx,10000
div     ebx
shl     eax,16
```

```
        add     [di],eax

        mov     eax,edx
        mov     edx,0
        mov     ebx,1000
        div     ebx
        shl     eax,12
        add     [di],eax

        mov     eax,edx
        mov     edx,0
        mov     ebx,100
        div     ebx
        shl     eax,8
        add      [di],eax

       mov      eax,edx
        mov     edx,0
        mov     ebx,10
        div     ebx
        shl     eax,4
        add     [di],eax
        add     [di],edx
        retn


printit proc    near            ;procedimiento para imprimir numeros
        push    ax
        push    cx
        push    dx
        mov     al,value                ;carga datos
        and     al,0f0h                 ;mantiene solo 4 bits superiores
        mov     cl,4            ;desplazamiento circular de cuatro bits inferiores
        ror     al,cl                   ;desplazamiento circular
        printnum                        ;macro printnum
        mov     al,value                ;cargar datos
        and     al,0fh                  ;mantiene solo 4 bits inferiores
        printnum
        pop     dx
        pop     cx
        pop     ax
        retn                            ;fin procedimiento
        printit     endp
final:  mov     bh,07
        clearscreen
        .exit                           ;volver al DOS
         end
```

PRURAT.ASM

```
;programa ejemplo de uso del ratón en lenguaje enzamblador
;presenta información del modo de video y de la posición del puntero

    dosseg          ;dice al enlazador que ordene los segmento de acuerdo a
                    ;la convención estándar del DOS
    .model small    ;define un modelo de memoria tipo Small: un segmento
                        ;para datos y otro para código
    .486                ;especifica que se acepten las instrucciones para el
                        ;microprocesador 80486

;_____________________________________________
;AREA PARA DATOS
;_____________________________________________
    .data                   ;inicio del segmento de datos
;**** Variables para el uso del ratón
vari   STRUCT               ;define una variable tipo estructura
       VAR11  dw    0       ; define una variable tipo palabra (16 bits)
       VAR12  dw    0
vari ENDS
var1 vari <0,0>             ;crea una variable tipo vari (estructura) , para uso
                            ;con macros Mreseteo
                            ;y Mmovera

RESULT STRUCT
       VAR31  dw    0
       VAR32  dw    0
       VAR33  dw    0
       VAR34  dw    0
RESULT  ENDS
VAR3 RESULT <0,0,0,0>  ;para uso con macro Mpresionado
VAR3a RESULT <0,0,0,0> ;para uso con macro Mliberado

RESULT1 STRUCT
       VAR21  dw    0
       VAR22  dw    0
       VAR23  dw    0
RESULT1  ENDS
VAR2 RESULT1 <0,0,0>    ;para uso con macro Mposicion

var     dw   0          ;indica si el punetro del ratón est visible
ratonpresente db  0     ;indica si el controlador del ratón est instalado
```

```
;****

;mensaje
inicial db    '          PROGRAMA PARA PROBAR EL RATON EN MODO
TEXTO'
        db    13,10,10              ;Enter (13) : pone el cursor al inicio
                                    ;Alimentación de línea (10)
        db    'Presione una tecla para continuar'
        db    13,10
        db    '$'                   ;fin de string de caracteres

cont          db    '+ Haga click en el cuadro o presione una tecla '
        db    'para terminar'
        db    13,10          ;Enter pone el cursor al inicio y alimentación de
                             ;línea
        db    ' modo video='
video   db    0                     ;almacenan el modo de vídeo en hexadecimal
        db    0
        db    'h'
        db    '  pagina activa='
activa  db    0              ;almacenan la página activa en decimal
        db    '  número de columnas='
columnas db   0              ;almacenan el número de columnas en decimal
        db    0
        db    0
cr      db    0dh
finstr  db    '$'            ;fin de mensaje

cont1   db    "x ="          ;mensaje
        db    30h            ;almacenan la posición horizontal en pixels del
puntero
        db    30h
        db    30h
        db    ' y ='
        db    30h            ;almacenan la posición vertical en pixels del puntero
        db    30h
        db    30h
        db    0dh,'$'        ;fin de mensaje


;_____________________________________
;AREA PARA CODIGO
;_____________________________________
    .code                           ;inicio del segmento de codigo
```

```
        include mouse.asm       ;se incluye el archivo de macros del ratón

        extrn pasborra:near     ;incluir el procedimiento externo Pasborra para
                                ;borrar la pantalla.
                                ;se debe enlazar: Link Prurat1+Pasborra
                                ;previamente debe ensamblarse  el archivo
                                ;Pasborra.asm y Prurat1.asm  separadamente

prurat:
        mov     ax,@data        ;se pone el segmento de datos en ds y es
        mov     ds,ax
        mov     es,ax
        call    pasborra        ;borrar la pantalla

        ;Poner mensaje inicial
        mov     ah,02           ;mover el cursor a la posición
        mov     dx,0h           ;00,00 (en caracteres)
        int     10h

        mov     cx,80           ;número de caracteres a poner en pantalla
        mov     al,' '          ;se escribe al carácter espacio blanco
        mov     bl,17h          ;fondo azul(1), carácter rojo(4)


        mov     bh,0            ;página de vídeo activa es 0
        mov     ah,9            ;función BIOS escribir caracteres y atributo desde
                                ;posición del cursor sin alterarla
        int     10h             ;Poner carácter y atributo en pantalla
                                ;contenidos en al y bl respectivamente
                                ;bh indica la página de vídeo en la que se
                                ;pondrán los caracteres

        mov     dx,offset inicial       ;mover a dx la dirección de inicial
        mov     ah,9
        int     21h             ;escribir el mensaje inicial

        mov     ah,0
        int     16h             ;esperar hasta que se presione una tecla

        mov     bh,0
        mov     al,' '
        mov     bl,24h          ;fondo verde
        mov     cx,1680         ;llenar franjas con fondo verde
        mov     ah,9
```

```
        int     10h

        mov     ah,03h          ;obtener información del cursor y poner en la pila
        int     10h
        push    cx

        mreseteo  var1,var,ratonpresente  ;macro: inicializar el ratón

        mov     var1.var11,105          ;mover puntero de ratón a la posi-
        mov     var1.var12,105          ;ción 105,105 en pixels
        mmovera var1,ratonpresente      ;macro: mover puntero

        mov     ah,0fh                  ;obtener información de vídeo
        int     10h                     ;al = número de car cteres por columna
        push    ax              ;ah = modo de vídeo
        push    bx              ;bh = página activa
        push    ax              ;guardar en la pila


        and     al,0fh          ;transformar el primer dígito de número
                                ;de columnas a
        cmp     al,0ah          ;formato a ASCII
        jbe     menoraa         ;saltar si es dígito
        add     al,7h           ;sumar 7 si es letra ( A a F)
menoraa:
        add     al,30h          ;sumar 30h para pasar a formato ASCII
        mov     video+1,al      ;almacenar el valor
        pop     ax              ;transformar el segundo dígito
        mov     cl,4
        shr     al,cl
        and     al,0fh
        cmp     al,0ah
        jbe     menorab
        add     al,7h
menorab:
        add     al,30h
        mov     video,al

        add     bh,30h          ;convertir el número de página a formato
        mov     activa,bh       ;ASCII

        mov     al,ah                   ;transformar el número de columnas
        xor     ah,ah                   ;a base decimal y luego a formato ASCII
```

```
        cmp   al,64h            ;primero hallamos las centenas
        jb    menora100b        ;el número es mayor a 100d?
        mov   bl,100            ;si, dividir para 100d
        div   bl
        add   al,30h            ;sumar 30h para pasar a ASCII
        mov   columnas,al       ;almacenarlo en columnas

        mov   al,ah             ;movemos el residuo a al
menora100b:                     ;hallamos las decenas
        cmp   al,0ah            ;es mayor a 10d?
        jb    menora10b
        mov   bl,10             ;si, dividir para 10h
        div   bl
        add   al,30h            ;sumar 30h
        mov   columnas+1,al     ;almacenar en columnas +1

        mov   al,ah             ;movemos el residuo a al
menora10b:                      ;que representan las unidades
        add   al,30H            ;sumar 30h para pasar a ASCII
        mov   Columnas+2,al     ;almacenar en columnas + 2

        pop   bx
        pop   ax

        mov   ah,02             ;mover el cursor a la posición
        mov   dx,0a0ah          ;10d,10d (en car cteres)
        int   10h

        mov   ah,9              ;función 9 del DOS: mostrar
        mov   dx,offset cont    ;un string terminado en '$'
        int   21h

        mov   ch,20h            ;ocultar el cursor
        mov   ah,01
        int   10h

lazorat:

        mov   al,7              ;carácter en el cuadro (punto)
        mov   bl,78h            ;atributo del cuadro
                                ;(fondo blanco y carácter gris obscuro)
        call  cuadro            ;dibujar cuadro
```

```
        mov     ah,02
        mov     dx,0d00h
        int     10h ;

        mposicion var2,ratonpresente    ;macro: obtener la posición del puntero
                                        ;del ratón en la variable var2 tipo
                                        ;estructura

        xor     dx,dx
        mov     ax,var2.var22           ;transformar la posición horizontal de
        cmp     ax,64h                  ;hexadecimal a decimal y luego a formato
        jb      menora100               ;ASCII
        mov     bx,100
        div     bx
        add     al,30h
        mov     cont1+3,al
        mov     ax,dx
        xor     dx,dx
menora100:
        push    ax
        mov     al,30h
        mov     cont1+4,al
        pop     ax
        cmp     ax,0ah
        jb      menora10
        mov     bl,10
        div     bl
        add     al,30h
        mov     cont1+4,al
        mov     al,ah
menora10:
        add     al,30H
        mov     cont1+5,al
        xor     dx,dx
        mov     ax,var2.var23           ;transformar la posición vertical de hexade-
        cmp     ax,64h                  ;cimal a decimal y luego a formato ASCII
        jb      menora100a
        mov     bx,100
        div     bx
        add     al,30h
        mov     cont1+10,al
        mov     ax,dx
        xor     dx,dx
menora100a:
```

```
        push    ax
        mov     al,30h
        mov     cont1+11,al
        pop     ax
        cmp     ax,0ah
        jb      menora10a
        mov     bl,10
        div     bl
        add     al,30h
        mov     cont1+11,al
        mov     al,ah
menora10a:
        add     al,30H
        mov     cont1+12,al

        mov     dx,offset cont1         ;mostrar mensaje y valores de posición
        mov     ah,9
        int     21h

        mov     al,' '                  ;poner los valores de posición del cursor
        mov     di,offset cont1+3       ;en blanco
        mov     cx,3
        rep     stosb                   ;repetir stosb el número de veces que indica
        mov     di,offset cont1+10      ;cx
        mov     cx,3
        rep     stosb

        mov     ah,1                    ;ver si se ha presionado una tecla
        int     16h
        jnz     finpru                  ;si: salir

        Mpresionado  0,var3,ratonpresente   ;macro: obtener en var3 el estado
                                            ;del botón izquierdo (0), y la
                                            ;posición del puntero desde la
                                            ;última vez que se presionó el bo-
                                            ;tón izquierdo

        cmp     var3.var31,1                ;se presionó el botón
        jne     lazorat                     ;no: entonces saltar a lazorat

        mov     al,20h ;178
        mov     bl,40h ;00
        call    cuadro
```

```
verificar:
      Mliberado   0,var3a,ratonpresente    ;macro: obtener en var3a el estado
                                           ;del botón izquierdo (0), y la
                                           ;posición del puntero desde la
                                           ;última vez que se liberó el bo-
                                           ;tón izquierdo
      cmp    var3a.var31,0                 ;botón izquierdo liberado?

      jne    verificar                     ;= 1 no liberado, saltamos a verificar
                                           ;hasta que se libere

      cmp    var3a.var33,70                ;si: verificamos si la posición
                                           ;del puntero al liberar el botón
      jb     lazorat                       ;estaba encima del cuadro (ubicado
      cmp    var3a.var33,80                ;en 70 a 80 horizontal en pixels
      ja     lazorat                       ;y 70 a 80 vertical en pixels
      cmp    var3a.var34,70                ;si no está en los límites
      jb     lazorat                       ;saltamos a lazorat
      cmp    var3a.var34,80
      ja     lazorat

FinPru:
      pop    cx                            ;restauramos el cursor
      mov    ah,01                         ;con los valores antes de entrar
      int    10h                           ;al programa

      mmuestra 0,var,ratonpresente         ;macro:hacer invisible el puntero
                                           ;del ratón

      call   pasborra

      mov    ah,4ch                        ;terminamos el proceso (programa)
      int    21h                           ;y retornar al DOS

cuadro:                                    ;dibuja un cuadro de 2x2 caracteres
      mov    ah,02
      mov    dx,0909h
      int    10h

      mov    cx,2
      mov    bh,0
otrafila:
      push   cx
      mov    ah,9
```

```
mov     cx,2            ;número de caracteres a poner en pantalla
int     10h             ;poner carácter y atributo en pantalla
                        ;contenidos en al y bl respectivamente
                        ;bh indica la página de vídeo en la que se
                        ;pondrán los caracteres

mov     ah,02           ;mover el cursor a la posición
mov     dx,0a09h        ;10d,9d (en caracteres)
int     10h

pop     cx
loop    otrafila        ;lazo hasta que cx=0
retn                    ;retorno

end     Prurat          ;se indica que el programa se ejecuta
                        ;inicialmente desde Prurat
end                     ;fin de todo el programa
```

PASBORRA.ASM

```
;módulo tipo publico que es invocado desde PRURAT.ASM y sirve para borrar la
;pantalla (para monitores a color). Se utiliza el método de escritura directa en
memoria de video

pantalla equ 0b800h          ;define a pantalla como el valor 0B800H

        .model small
        .code
        public pasborra      ;define al módulo como público
pasborra proc near           ;define el procedimiento pasborra

    push    es               ;guardar en la pila los registros importantes
    push    ax
    push    cx
    push    dx
    push    di
    mov     ax,pantalla      ;mover a AX la dirección inicial de la memoria de
                             ;vídeo
    mov     es,ax            ;pasar la dirección de vídeo a ES
    mov     di,0
    mov     cx,2000          ;número de localidades de memoria a escribir
                             ;una palabra por cada carácter (un byte de atributo y
                             ;un byte de código ASCII)
    cld                      ;incremento del puntero para trabajo con cadenas de
                             ;caracteres
    mov     al,' '           ;grabar en cada localidad el código ASCII del espacio
                             ;en blanco
    mov     ah,07h           ;grbar en cada localidad de atributo el atributo 07h
                             ;(fondo negro y letras blancas)
    rep     stosw            ;escribir las localidades de memoria
    pop     di               ;restablecer los registros desde la pila
    pop     dx
    pop     cx
    pop     ax
    pop     es

    ret                      ;retorno
    pasborra endp            ;fin del procedimiento pasborra
    end                      ;fin del archivo
```

MODULO INICIAL.ASM

```
        dosseg
        .model  small
        .486

;_____________________________
;AREA PARA PILA
;_____________________________
        .stack  200h

;_____________________________
;AREA PARA DATOS
;_____________________________
        .data

        extrn   ambiente:word,ejecutamenu:byte
        extrn   pixelx:byte,pixely:byte,ratonpresente:byte
        extrn   tabinf:near,tabbuf:near
        extrn   VentNueva:byte,VentActual:byte,vent:byte
        public  visible
        public  dirret,mraux2,mraux3,videoseg,valptr
        public  mopaux,mopaux1,tiprut,otroaux,colven
        public  areamac,bmac,bgmac,bver

        bopc    db      0               ;bandera para mostrar subopciones
        bmac    db      0               ;bandera de ejec. de macros
        bver    db      0               ;bandera de comprob. de macros
        bgmac   db      0               ;bandera de grab. de macros
        nivel   db      0               ;nivel del menu actual
        nivel1  db      0               ;nivel de la opcion actual
        nivel2  db      0               ;nivel de la subopcion actual
        respnivel db    0
        buffer  db      8 dup(0)        ;indica camino hacia menu actual
        buffer1 db      8 dup(0)        ;camino hacia opcion actual
        buffer2 db      8 dup(0)        ;camino hacia subopcion actual
        respbuf db      8 dup(0)
        nclave  db      0               ;ubicacion de BUFFER's en TABBUF
        nclave1 db      0
        nclave2 db      0
        x       db      0               ;# de opcion actual
        y       db      0               ;# de subopcion actual
        xprev   db      0               ;X que origino menu actual
        acabar? db      'Quieres terminar (S/N)?',0,2,'SN'
        miodir  dw      0               ;Variables de macros del MASM
```

```
mopaux  db   0            ;sitios para paso de dato de
mopaux1 db   0            ;teclado si hay rutina de escape
mopact  db   0
mbcbyte db   0
mbcmenu db   0
mbcpos  dw   0
mbcbuf  dw   0
videoseg dw  0                ;segmento de video
panbuf  db   3680 dup(0)      ;sitio para salvar pantalla
mmvx1   db   0
mmvlin  db   0
mmvlin1 db   0
mmvcar  db   0
mmvdir  dw   0
mv?aux  db   0
mv?aux1 db   0
mpmprx  dw   0
maylin  db   0
maycol  db   0
dirsal  dw   2 dup(0)         ;sitio para dir. de rutinas de usuario
tiprut  db   0                ;tipo de rutina: 1, 2 o 3
tiprut1 db   0                ;tipo de rutina de opcion actual
mraux1  db   0
mraux2  dw   0            ;Aqui se guardan datos mientras se
mraux3  dw   0            ;corre la rutina del usuario.
dirret  dw   2 dup(0)     ;Direccion de retorno
diract  dw   2 dup(0)
inword  dw   0
getword dw   0            ;salida de GETKEY
mnmos   db   0
mdirop  dw   0
moslin  db   0
moscol  db   0
mpddir  dw   0
mcudir  dw   0
mibdir  dw   0
mibx    db   0
faseptr dw   0            ;puntero de fase de programa (73,23)
valptr  dw   0            ;puntero de valor que va en (73,24)
msfdir  dw   0
fnorm   db   "Normal"     ;fases del programa.
fgrab   db   "Grab-M"
fprob   db   "Prob-M"
fcorr   db   "Corr-M"
fmac    db   "Macros"
otroaux db   0            ;bandera que indica paso por "ootr".
blanco  db   78 dup (20h),0
```

```
insprog db   "                OPCIONES VALIDAS EN LOS MENUES.",0
        db   " Estas son las opciones válidas para cualquier menú :",0
        db   " Se puede elegir una opción por su `letra clave' o "
        db   "con las flechas y ENTER.",0
        db   " ^O  presenta/quita las subopciones de una "
        db   "opción en una ventana.",0
        db   " ESC permite salir al nivel superior de menúes.",0
        db   " ^P  presenta esta pantalla de ayuda.",0
        db   " ^K  lleva a las opciones de macros del programa.",0
        db   "                ...Presiona cualquier tecla...",0
cscrnm  db   " Nombre del macro (7 car. máx.) o ENTER para salir...",0
nombuf  db    7,9 dup (0)    ;buffer para recibir el nombre del macro
areamac db    64 dup (0)     ;indice de macros
macl    dw    8*64 dup (0)   ;area de macros
nummac  db    0              ;numero del macro actual en el indice
contmcr db    0              ;interface de macros de INICIAL y usuario
longmac db    0              ;longitud del macro actual
contmac db    0              ;contador de posicion en el macro actual
buffmac dw    64 dup (0)     ;macro actual
nombmac db    7 dup (0)      ;nombre del macro actual
nomac   db   "Macros solo desde menú principal. Presiona una tecla.",0
noroom  db   "No hay sitio para nuevo macro. Presiona una tecla.",0
grmac   db   "Inicio grabación de macro. Terminar: ^K. "
        db   "Presiona una tecla.",0
borrmac db    "Borro este macro (S/N)?",0,2,"SN"
promac  db    "Inicio prueba de macro. Seguir: ENTER. Terminar: ^K",0
ejemac  db    "Inicio corrida de macro. Presiona una tecla.",0
escmcr  db    "Fin del buffer y fin del macro. Presiona ESC.",0
insmac  db    33 dup (20h),"MACROS.",0
        db   "      Se pueden usar hasta 8 macros de 64 "
        db   "opciones máximo.",0
        db   "      Ingresa desde menú principal con ^K.",0
        db   "      Escribe el nombre del macro. Termina con ENTER.",0
        db   "      Si es nuevo, defínelo.",0
        db   "      Si es viejo lo puedes probar, correr o borrar.",0
        db   "      Este es el índice actual de macros:",0
        db   6 dup (20h)
lismac  db   64 dup (20h),0
pcbmac  db    "Probar...P  Correr...C  Borrar...B",0,3,"PCB"
colven  db    bnnorm         ;atributos: color de ventana
collin  db    bnnorm         ;color de linea
colexp  db    bnnorm         ;color de explicacion
colact  db    bnsubb         ;color de subopcion actual
collact db    bninv          ;color de opcion actual

        ; Variables para uso del ratón
                                ;      *Para      los      macros:
```

```
mreseteo,Mmovera,MXlimit,MYlimit,Mmovimiento,Moveratio
        var1    STRUCT
           var11 dw 0
           var12 dw 0
        var1    ENDS
        varraton1 var1 <0,0>

        ;* Para los Macros: Mposicion, MformaCursor
        var2 STRUCT
                var21 dw 0
                var22 dw 0
                var23 dw 0
                var2    ENDS
        varraton2 var2 <0,0,0>

        ;* Para macros: MPresionado,Mliberado,Mocultar
        var3 STRUCT
                var31 dw 0
                var32 dw 0
                var33 dw 0
                var34 dw 0
        var3    ENDS
        varraton3 var3 <0,0,0,0>

        visible dw 0      ; indica si el cursor esta activado/desactivado (1/0)

        Linea1          db 4,12,20,28,36,44,52,60,68,76,0
        Linea2          db 1,36,37,63,80,0 ;sirve para líneas 2,3,4,5
        Linea21         db 72,79,0
        Linea6          db 36,37,80,0  ;sirve para líneas 6,7
        Linea10         db 1,36,46,70,79,80,0 ; sirve para líneas 10,11,12,13,14,15
                                                ; 16,17,18,19,20,21,22
        Linea23         db 7,15,24,33,39,45,53,62,72,80,0
        numeroop        db 0    ; indica cuantasa opciones hay en
                                ; la línea de menu

;_____________________________
;AREA PARA CODIGO
;_____________________________
        .code

        extrn   rinic:near,rfinal:near,funciones:near
      extrn   nmaxbuf:abs,restorvec:near

ninfbuf equ     7               ;informaciones asociadas a buffer
longlin equ     160             ;80*2 (incluyendo atributos)
fleder  equ     77              ;teclas pulsadas:
```

```
fleizq  equ    75
flearr  equ    72
fleaba  equ    80
cr      equ    13
ctrlo   equ    15
ctrlp   equ    16
ctrli   equ    9
ctrlk   equ    11
esqsi   equ    0dah          ;codigos para dibujo de cuadro
linsup  equ    0c4h
esqsd   equ    0bfh
linlat  equ    0b3h
esqii   equ    0c0h
esqid   equ    0d9h
bnnorm  equ    7             ;video normal (B/N)
bnsubb  equ    09h           ;subrayado intenso
bninv   equ    78h           ;inverso
maxmac  equ    64            ;macros de 64 words maximo

        include inimac.asm   ;incluir archivo de macros INIMAC.ASM
        include mouse.asm
inicial proc   near

        subrut               ;subrutinas de macros del MASM.
inicio:
        mov    ax,@data      ;inicializo DS y ES
        mov    ds,ax
        mov    bx,4ch
        mov    ax,word ptr es:[bx]
        mov    ambiente,ax
        mov    ax,40h
        mov    es,ax
        mov    ax,es:[63h]
        mov    dx,0b000h
        cmp    ax,3b4h
        je     vidaddr
        add    dx,800h
vidaddr:
        mov    videoseg,dx
        mov    es,dx
        mov    ax,0720h      ;20h = " ", 07h = video normal.
        mov    cx,2000
        cld
        xor    di,di
        rep    stosw         ;borro pantalla
        mov    cl,25
i_1:    pchar  0ah           ;escribo 25 LF para poner
```

```
        dec     cl              ;cursor al fin de pantalla
        jnz     i_1
        mov     faseptr,offset fnorm
        mov     valptr,offset blanco
        mreseteo varraton1,visible,ratonpresente
        mov     varraton1.var11,77FFh
        mov     varraton1.var12,7700h
        formacursor 0,varraton1,ratonpresente
        mmuestra  0,visible,ratonpresente
        mov     varraton1.var11,0       ;7
        mov     varraton1.var12,0       ;192
        mmovera varraton1,ratonpresente     ;mueve el cursor del raton a la posición
                                            ;que estaba antes de salir de pantalla
        call    rinic           ;rutina inicial de usuario.
        guarpantalla            ;muestro instrucciones
        lmppnt
        ayuda   insprog,8,collact
        getkey
        restpantalla
        mov     nivel,0         ;inicializo variables que
        mov     buffer,1        ;manejan flujo del programa.
        mov     buffer1,12h     ;11h
        mov     bopc,0          ;no subopciones.
        mov     bmac,0          ;no macros.
        mov     bver,0          ;no verificar macros.
        mov     bgmac,0         ;no grabar macros.
        mov     x,2     ;1              ;opcion actual
a_nivel:
        mov     y,0             ;subopcion actual
b_subnivel:
        bufcla  nclave,buffer   ;Variables del menu actual
        movbuf  respbuf,buffer
        mov     al,nivel
        mov     respnivel,al
        movbuf  buffer1,buffer
        mov     al,nivel
        mov     nivel1,al       ;Variables de opcion actual
        inc     nivel1
        nunivel buffer1,nivel1,x
        bufcla  nclave1,buffer1
        movbuf  buffer2,buffer1
        mov     al,nivel1
        mov     nivel2,al       ;Variables de subopcion actual
        inc     nivel2
        nunivel buffer2,nivel2,y
        bufcla  nclave2,buffer2
        explica nclave2         ;Explico que hace la subopcion
```

```
e_ventana:
        cmp     bopc,1          ;muestro ventana de subopciones?
        jne     b_s5
        guarpantalla            ;si, pero salvo pantalla.
        menuvent nclave1
b_s5:
      menulinea nclave          ;muestro menu,
        statusf                 ;fase del programa y
        statusv                 ;valor dado por usuario.
        mov     otroaux,0
c_opcion:
        opcion?                 ;recibe datos de teclado.
        mov     al,mopact       ;# de la opcion elegida.
c_op1:  actuar                  ;salta a la rutina correcta.
        dw      offset oizq,offset oder,offset oarr
        dw      offset oaba,offset oret,offset ootr
        dw      offset octo,offset octp,offset octk
        dw      offset octl
oizq:
        menizq  nclave          ;pasar a opcion a la izquierda
        jmp     b_subnivel
oder:
        mender  nclave          ;opcion a la derecha
        jmp     b_subnivel
oarr:
        cmp     bopc,1          ;subopcion superior si
        je      c_oa1           ;la hay y si BOPC=1
        jmp     c_opcion
c_oa1:
        cmp     tiprut1,3       ;opcion tiene subopciones?
        jz      c_oa2
        jmp     c_opcion        ;no.
c_oa2:  menarr  nclave1         ;si.
        jmp     b_subnivel
oaba:
        cmp     bopc,1          ;subopcion inferior si
        je      c_ob1           ;la hay y si BOPC=1
        jmp     c_opcion
c_ob1:
        cmp     tiprut1,3
        jz      c_ob2
        jmp     c_opcion
c_ob2:  menaba  nclave1
        jmp     b_subnivel
oret:
        inc     nivel           ;se presiono ENTER, correr
        nunivel buffer,nivel,x  ;rutina de opcion actual.
```

```
c_or2:  cmp    y,0
        jne    c_or1
        cmp    tiprut1,3
        je     c_or3
        jmp    c_oo2
c_or3:  mov    x,1
        jmp    c_oo2
c_or1:  mov    al,y
        mov    x,al
        jmp    c_oo2
ootr:
        valida? nclave          ;se digito una opcion desconocida
        je     c_oo1            ;esta entre las validas del menu?
        jmp    e_ventana        ;no.
c_oo1:
        mov    y,0              ;si, correr rutina de esa opcion.
        mov    x,al
        inc    nivel
        nunivel buffer,nivel,x
        mov    otroaux,1
c_oo2:
        bufcla  nclave,buffer   ;aqui se transfiere el control al
        rutina  nclave          ;usuario y este lo devuelve.
c_oo3:  actuar                  ;actuar de acuerdo a tiprut.
        dw     offset rnorm,offset rinv,offset rrama
rnorm:
        movbuf  buffer,respbuf  ;normal, quedarse en nivel actual.
        mov    al,respnivel
        mov    nivel,al
        jmp    b_subnivel
rinv:
        movbuf  buffer,respbuf  ;inversa, salir a nivel superior.
        mov    al,respnivel
        mov    nivel,al
        jmp    d_supernivel
rrama:
        cmp    otroaux,1        ;rama, profundizar.
        jne    rrama1
        mov    x,1
rrama1: jmp    a_nivel
octo:
        cmp    bopc,1           ;si BOPC=1 hacer 0 y viceversa.
        je     c_oco1
        mov    bopc,1
        jmp    c_oco2
c_oco1: mov    bopc,0
        mov    y,0
```

```
c_oco2: jmp    b_subnivel
octp:
        guarpantalla                ;mostrar instrucciones.
        lmppnt
        ayuda  insprog,8,collact
        getkey
        restpantalla
      jmp    b_subnivel

octk:                               ;Rutinas de macros.
        mov    contmcr,0            ;reset a byte de interface.
        guarpantalla
        mov    ax,offset blanco
        pmenu  24,colexp
        cmp    nivel,0              ;nivel actual era 0?
        je     c_ok0
        mov    ax,offset nomac      ;no, salir.
        pmenu  23,collact
        getkey
        jmp    c_ok5
c_ok0:
        mov    x,1                  ;los macros se deben iniciar
        mov    y,0                  ;desde el mismo punto de partida.
        mov    bopc,0
        mov    ax,0                 ;borrar area de nombre de macro.
        push   es
        push   ds
        pop    es
        mov    di,offset nombuf
        inc    di
        mov    cx,9
        rep    stosb
        mov    cx,maxmac            ;borro buffer de macros.
        mov    di,offset buffmac
        rep    stosw
        mov    contmac,0
        mov    cx,7                 ;borro nombre actual.
        mov    di,offset nombmac
        rep    stosb
        pop    es
        indmac                      ;prepara un indice de macros.
        lmppnt
        ayuda  insmac,8,collact     ;instrucciones de macros.
        mov    ax,offset escmm
        pmenu  23,collact
        mov    faseptr,offset fmac
        statusf                     ;muestra fase del programa.
```

```
        inbuff nombuf              ;recibe un nombre.
        mov    si,offset nombuf
        inc    si
        mov    al,[si]
        cmp    al,0                ;se dio nombre?
        jne    c_ok01              ;si.
        mov    faseptr,offset fnorm    ;no, salir.
        statusf
        jmp    c_ok5
c_ok01: getnmac                    ;obtengo numero de macro.
        cmp    nummac,0ffh         ;macro nuevo sin sitio?
        jne    c_ok1               ;no
        mov    ax,offset noroom
        pmenu  23,collact
        getkey
        mov    faseptr,offset fnorm
        jmp    c_ok5
c_ok1:  cmp    nummac,80h          ;macro antiguo?
        jb     c_ok2
        jmp    c_ok3               ;si.
c_ok2:  mov    ax,offset grmac     ;no, grabar macro nuevo.
        pmenu  23,collact
        mov    faseptr,offset fgrab
        statusf
        getkey
        push   es
        push   ds
        pop    es
        mov    si,offset nombuf
        add    si,2
        mov    di,offset nombmac
        mov    cx,7
        rep    movsb
        pop    es
        mov    bgmac,1             ;bandera de grabacion.
        jmp    c_ok5
c_ok3:  sub    nummac,80h          ;macro viejo.
        mostrar pcbmac,23,collact
        actuar
        dw     offset c_ok31,offset c_ok32,offset c_ok33
c_ok31: mov    ax,offset promac    ;probar macro.
        pmenu  23,collact
        mov    faseptr,offset fprob    ;mostrar fase.
        statusf
        getkey
        call   inimac
        mov    bver,1              ;banderas para probar macro.
```

```
        mov     bmac,1
        jmp     c_ok5
inimac:
        push    es                      ;paso macro al buffer Buffmac
        push    ds
        pop     es
        mov     si,offset mac1
        mov     al,nummac
        mov     cl,2*maxmac
        mul     cl
        add     si,ax
        mov     di,offset buffmac
        mov     cx,maxmac
        rep     movsw
        mov     si,offset areamac
        mov     al,nummac
        mov     cl,8
        mul     cl
        add     si,ax
        mov     al,byte ptr[si]         ;obtengo datos de macro actual.
        mov     longmac,al
        inc     si
        mov     di,offset nombmac
        mov     cx,7
        rep     movsb
        mov     contmac,0
        pop     es
        retn

;RUTINA DE MANEJO DEL RATON

public pantalla
pantalla proc near  ;rutina de tratamiento del ratón
    mmuestra 1,visible,ratonpresente ;macro: hace visible el puntero
                                     ;del ratón
    mposicion varraton2,ratonpresente ;macro: obtiene la posición del
                                      ;puntero del ratón
    mpresionado 0,varraton3,ratonpresente  ;obtiene el estado del botón
                                           ;izquierdo del ratón
    mov     ax,varraton2.var23      ;ax = posición vertical
    mov     bl,pixely               ;bl = pixel por carácter en y
    div     bl                      ;ax/bl
    cmp     ejecutamenu,1           ;se está ejecutando alguna
                                    ;opción de menú?
    je      noopcion                ;si, ratón no debe realizar nin-
                                    ;guna opción
    cmp     ax,1                    ;el ratón está en línea 1 (menú)?
```

```
        ja      no1                     ;no
        mov     si,offset linea1        ;si
        cmp     varraton3.var32,1       ;se presiono botón izquierdo del raton?
        jne     noenter1                ;no
        cmp     varraton2.var22,32      ;es la primera opción(..)?
        jbe     esesc                   ;si, enviar código de ESC
        mov     ejecutamenu,1           ;indicar que se está ejecutando opción
                                        ;de menú
        mov     ax,1c0dh                ;enviar el código de Enter
        jmp     finpantalla
noenter1:
        mmoviniento varraton1,ratonpresente
        cmp     varraton1.var11,0
        jne     simovio
        cmp     varraton1.var12,0
        je      noopcion
simovio:

        cmp     vent,1                  ;se está en edición de ventanas
        je      noopcion                ;si, entonces ratón no hace nada
        call    scanlinea               ;entrega en bl la opción sobre la
                                        ;que está el puntero del ratón
        cmp     x,bl                    ;es igual a la opción actual?
        je      noopcion                ;si
        cmp     bx,0ffh                 ;se está fuera de cualquier opción?
        je      finpantalla1            ;si, no se hace nada
        cmp     bl,numeroop             ;es el valor de bl mayor anúmero
                                        ;máximo de opciones del menú?
        ja      noopcion                ;si, no hacer nada
        inc     bx                      ;incrementamos bl
        mov     x,bl                    ;incrementar el apuntador de opciones
        mov     ax,4b00h                ;código de flecha a la izquierda
        jmp     finpantalla
esesc:
        mov     ax,011bh                ;código de ESC
        jmp     finpantalla

 no1:
        cmp     ax,23           ;es la línea 23 (funciones)?
        jne     no23            ;no
        cmp     varraton3.var32,1       ;se presiono botón izquierdo del raton?
        jne     noopcion                ;no, no hacer nada
        mov     si,offset linea23       ;
        call    scanlinea       ;ver que sobre que función se está
        mov     ax,3a00h        ;código de F1
        add     ah,bl           ;sumar el número de opción para obtener
                                ;el código de la función respectiva
```

```
        mov     ejecutamenu,1       ;indicar que se está ejecutando
                                    ;una opción
        cmp     ax,4000h
        je      noleer
        cmp     ax,4300h
        je      noleer
        mov     ejecutamenu,0
noleer:
        jmp     finpantalla

no23:
        cmp     ejecutamenu,1
        je      noopcion
        cmp     varraton2.var21,1   ;se presionó botón izquierdo del ratón?
        jne     noopcion            ;no
        cmp     ax,2                ;es mayor que la línea 2?
        jb      noopcion            ;no, no hacer nada
        cmp     ax,5                ;está en las líneas 2 a 5?
        ja      no2_5               ;no
        cmp     varraton3.var32,1   ;se presionó botón del ratón
        jne     noopcion
        mov     si,offset linea2
        push    ax
        call    scanlinea           ;bl = ventana sobre la que está el puntero
        pop     ax
        cmp     ax,4                ;estamos en línea 2, 3 o 4?
        ja      no3                 ;no
        cmp     bx,1                ;es la barra de movimiento?
        jne     no3                 ;no
        cmp     vent,1              ;se estaba editando ventanas?
        jne     no3                 ;no
        mov     ventnueva,2         ;activamos la ventana 2(código)
        mov     al,Ventnueva
        cmp     al,Ventactual       ;es ventana actual igual a ventana nueva?
        je      espgup              ;si
        mov     ax,0f00h            ;enviar código de TAB
        mov     ventnueva,2
        jmp     finpantalla
espgup:
        mov     ax,4900h            ;enviar código de Página Arriba
        jmp     finpantalla

no3:
        cmp     bx,2                ;es área de ventana 2 (código)?
        ja      novent2             ;no
        cmp     ax,2
        je      noopcion
```

```
        mov     VentNueva,2         ;activar ventana 2
        jmp     fin3_22
novent2:
        cmp     bx,3            ;es la barra de la ventana 0?
        jne     novent0a        ;no
        cmp     vent,1          ;se estaba editando en ventanas?
        jne     activarvent0    ;no
        mov     al,ventnueva
        cmp     al,ventactual
activarvent0:
        mov     VentNueva,0     ;activar ventana 0
        je      editvent0
        jmp     fin3_22
editvent0:
        cmp     varraton3.var32,1
        jne     noopcion
        mov     ax,4800h        ;enviar código de flecha arriba
        jmp     finpantalla

novent0a:
        cmp     bx,4            ;está sobre la ventana 0
        jne     novent0         ;no
        mov     VentNueva,0
        jmp     fin3_22

novent0:
        cmp     bx,5            ;es la ventana 1?
        jne     noopcion        ;no
        mov     VentNueva,1     ;activar ventana 1
        mov     al,ventnueva    ;
        cmp     al,ventactual   ;ventana nueva = a ventana actual
        je      editvent1       ;si
        jmp     fin3_22
editvent1:
        mov     si,offset linea21
        call    scanlinea       ;es la barra de ventana 1
        mov     ax,4b00h        ;enviar código de flecha a la izquierda
        cmp     bx,2
        jne     flechaizq21
        mov     ax,4d00h        ;código de flecha a la derecha
flechaizq21:
        jmp     finpantalla

no2_5:
        cmp     ax,7            ; está el puntero en líneas 6 o 7?
        ja      no6_7           ;no
        push    ax
```

```
        mov     si,offset linea6
        call    scanlinea       ;ver sobre que ventana está el puntero
        pop     ax
        cmp     bx,1            ;estamos en la barra de ventana 2
        jne     novent2a        ;no
        mov     VentNueva,2     ;actualizar ventana 2
        jmp     fin3_22
novent2a:
        cmp     bx,2            ;estamos en ventana 2
        jne     novent0b        ;no
        cmp     vent,1          ;estabamos editando en ventanas
        jne     activarvent0a   ;no
        mov     al,ventnueva
        cmp     al,ventactual
activarvent0a:
        mov     VentNueva,0
        je      editvent0a
        jmp     fin3_22
editvent0a:
        cmp     varraton3.var32,1 ;se presionó botón izquierdo del ratón
        jne     noopcion
        mov     ax,5000h        ;enviar código de flecha abajo
        jmp     finpantalla

NoVent0b:
        cmp     bx,3            ;esta sobre la ventana 0
        jne     noopcion
        mov     VentNueva,0     ;actualizar ventana 0
        jmp     fin3_22

no6_7:
        cmp     ax,22           ;estamos en las líneas 8 a 22
        ja      noopcion        ;no
        mov     si,offset linea10 ;
        push    ax
        call    scanlinea       ;vemos en que posición está el puntero
        pop     ax              ;
        cmp     ax,16           ;es línea 8 a 16
        ja      no15a           ;
        cmp     ax,14           ;está sobre líneas de 14 a 16
        jb      no15a
        cmp     bx,1            ;está en la barra de ventana 2
        jne     no15            ;no
        cmp     vent,1
        jne     no15
        mov     ventnueva,2
        mov     al,Ventnueva
```

```
        cmp     al,Ventactual
        je      espgdown
        mov     ax,0f00h        ;enviar códig de TAB
        jmp     finpantalla
espgdown:
        cmp     varraton3.var32,1
        jne     noopcion
        mov     ax,5100h        ;enviar código de página abajo
        jmp     finpantalla

no15a:
        cmp     ax,11
        ja      no10
        cmp     bx,6
        jne     no15
        cmp     vent,1
        jne     no15
        cmp     ventactual,4
        jb      no15            ;cambiado 97/03/09
        mov     al,ventnueva
        cmp     al,ventactual
        jne     igual10_20
        mov     cx,0fffeh
retardo1:
        loop    retardo1
        jmp     espgup
igual10_20:
        cmp     vent,1
        jne     no15
        mov     ventnueva,5     ;actualizar ventana 5
        mov     ax,0f00h
        jmp     finpantalla
no10:
        cmp     ax,20
        jb      no15
        cmp     bx,6
        jne     no15
        cmp     vent,1
        jne     no15
        cmp     ventactual,4
        jb      no15        ;cambiado 97/03/09
        mov     al,ventnueva
        cmp     al,ventactual
        je      espgdown1
        jmp     igual10_20
espgdown1:
        mov     cx,0ffffh
```

```
retardo:
        loop    retardo
        mov     ax,5100h
        jmp     finpantalla
no15:
        cmp     bx,2
        ja      novent2b
        mov     VentNueva,2  ;actualizar ventana 2
        jmp     fin3_22
Novent2b:
        cmp     bx,3
        jne     Novent3
        mov     VentNueva,3   ;actualizar ventana 3
        jmp     fin3_22
Novent3:
        cmp     bx,4
        jne     Novent4
        mov     VentNueva,4   ;actualizar ventana 4
        jmp     fin3_22
NoVent4:
        cmp     bx,6
        ja      noopcion
        mov     Ventnueva,5   ;actualizar ventana 5
        jmp     fin3_22
noopcion:
        mov     bx,0ffh
        jmp     finpantalla1
fin3_22:
        mov     ax,3b00h
finpantalla:
        mov     bx,0
        mmuestra 0,visible,ratonpresente  ;macro: ocultar puntero de ratón
finpantalla1:
        retn
pantalla endp


scanlinea:
        mov     ax,varraton2.var22
        mov     bl,pixelx
        div     bl
        mov     bx,0
mayor:
        cmp     byte ptr [si+bx],0
        je      nohayopciones
        cmp     al,[si+bx]
        inc     bx
```

```
        jbe     menorigual
        jmp     mayor
nohayopciones:
        mov     bx,0ffh
menorigual:
        retn

c_ok32: mov     ax,offset ejemac        ;correr macro.
        pmenu   23,collact
        mov     faseptr,offset fcorr
        statusf
        getkey
        call    inimac
        mov     bver,0                  ;banderas para correr macro
        mov     bmac,1
        jmp     c_ok5
c_ok33: mov     faseptr,offset fnorm
        statusf
        mostrar borrmac,23,collact      ;borrar macro?
        cmp     al,1
        jne     c_ok5                   ;no
        mov     ax,8                    ;si
        push    es
        push    ds
        pop     es
        mov     cl,nummac
        mul     cl
        add     ax,offset areamac
        mov     di,ax
        mov     cx,8
        xor     ax,ax
        rep     stosb
        mov     ax,2*maxmac
        mov     cl,nummac
        mul     cl
        add     ax,offset mac1
        mov     di,ax
        mov     cx,maxmac
        xor     ax,ax
        rep     stosw
        pop     cs
c_ok5:  restpantalla
        jmp     b_subnivel

octl:
d_supernivel:
        getxpr                  ;xprev=ultimo nibble del buffer <>0
```

```
        nunivel buffer,nivel,0  ;y que ahora pasa a ser 0.
        dec     nivel
        cmp     nivel,0ffh      ;estaba en menu principal?
        je      d_s1
        mov     al,xprev        ;no, voy a nivel superior.
        mov     x,al
        jmp     a_nivel
d_s1:
        mostrar acabar?,24,colexp
        actuar
        dw      offset d_s3,offset d_s2
d_s2:
        mov     x,1             ;no acabar.
        mov     nivel,0
        nunivel buffer,nivel,1
        jmp     a_nivel
d_s3:
        call    rfinal          ;rutina de salida del usuario.
final:
        call    restorvec
        mov     visible,0
        mmuestra    1,visible,ratonpresente
        push    ax
        xor     ax,ax
        mov     fs,ax
        mov     gs,ax
        pop     ax
        mov     ah,4ch
        int     21h

inicial endp
        end     inicio
        end
```

MODULO PR0486.ASM

```
        .model  small
        .486
        .data
         public  tabbuf,tabinf,nmaxbuf,ventnueva,ventactual,vent
         public  ejecutamenu,ambiente,pixelx,pixely,ratonpresente
         extrn   visible:word
        extrn   dirret:near,mraux2:word,mraux3:word,otroaux:byte
        extrn   videoseg:word,valptr:word,colven:near
        extrn   mopaux:byte,mopaux1:byte,tiprut:byte
        extrn   areamac:near,bgmac:byte,bmac:byte,bver:byte
        extrn   RegAx:Dword,RegBx:Dword,RegCx:Dword,RegDx:Dword
        extrn   RegDs:word,RegEs:word,RegCs:word,RegSS:word
        extrn   RegSi:Dword,RegDi:Dword,RegBp:dword,RegSp:dword
        extrn   Flags:word,Eflags:Dword,RegIp:Dword,regfs:word,reggs:word
        extrn   PointerLst:dword,XferAddr:dword
        extrn   vector2f:dword,FLoadExe:byte
        extrn   PSPparent:WORD,FileSize:word,FileName:byte
        extrn   FinNormal:byte,TablaBrk:byte

nmaxbuf equ     30                              ;# de buffers

tabbuf  db      01h,00h,00h,00h,00h,00h,00h,00h     ;principal
        db      11h,00h,00h,00h,00h,00h,00h,00h     ;salir
        db      21h,00h,00h,00h,00h,00h,00h,00h     ;an
        db      31h,00h,00h,00h,00h,00h,00h,00h     ;informacion
        db      41h,00h,00h,00h,00h,00h,00h,00h     ;depurador
        db      41h,01h,00h,00h,00h,00h,00h,00h     ;salir a depurar
        db      41h,02h,00h,00h,00h,00h,00h,00h     ;dp
        db      41h,03h,00h,00h,00h,00h,00h,00h     ;dc
        db      41h,04h,00h,00h,00h,00h,00h,00h     ;dj
        db      41h,05h,00h,00h,00h,00h,00h,00h     ;dl
        db      41h,06h,00h,00h,00h,00h,00h,00h     ;dr   *
        db      41h,16h,00h,00h,00h,00h,00h,00h     ;salir a dr
        db      41h,26h,00h,00h,00h,00h,00h,00h     ;drc
        db      41h,36h,00h,00h,00h,00b,00h,00h     ;dra
        db      41h,46h,00h,00h,00h,00h,00h,00h     ;drb
        db      41h,56h,00h,00h,00h,00h,00h,00h     ;drm
        db      41h,66h,00h,00h,00h,00h,00h,00h     ;drf
        db      41h,07h,00h,00h,00h,00b,00h,00h     ;di
        db      41h,08h,00h,00h,00h,00h,00h,00h     ;deb
        db      41h,09h,00h,00h,00h,00h,00h,00h     ;dm    *
        db      41h,19h,00h,00h,00h,00h,00h,00h     ;salir a dm
        db      41h,29h,00h,00h,00h,00h,00h,00h     ;dml
```

```
        db    41h,39h,00h,00h,00h,00h,00h,00h      ;dmb
        db    41h,49h,00h,00h,00h,00h,00h,00h      ;dmc
        db    41h,59h,00h,00h,00h,00h,00h,00h      ;dmf
        db    41h,0Ah,00h,00h,00h,00h,00h,00h      ;df
        db    51h,00h,00h,00h,00h,00h,00h,00h      ;salir
        db    61h,00h,00h,00h,00h,00h,00h,00h      ;shell
        db    71h,00h,00h,00h,00h,00h,00h,00h      ;ensambla
        db    81h,00h,00h,00h,00h,00h,00h,00h      ;funciones

tabinf  dw    offset exp0,offset menu0,offset rut0  ;principal
        db    3
        dw    offset sup,offset menu0,offset salida
        db    2
        dw    offset ean,offset menu0,offset ran    ;an
        db    1
        dw    offset ei,offset menu0,offset rinfo   ;informacion (i)
        db    1
        dw    offset ed,offset md,offset rut0       ;depurador(d)
        db    3
        dw    offset sup,offset md,offset salida
        db    2
        dw    offset edp,offset menu0,offset rdp    ;dp
        db    1
        dw    offset edc,offset menu0,offset rdc    ;dc
        db    1
        dw    offset edj,offset menu0,offset rdj    ;dj
        db    1
        dw    offset eal,offset menu0,offset ral    ;dl
        db    1
        dw    offset edr,offset mdr,offset rut0     ;dr
        db    3
        dw    offset sup,offset mdr,offset salida
        db    2
        dw    offset edrc,offset menu0,offset rdrc  ;drc
        db    1
        dw    offset edra,offset menu0,offset rdra  ;dra
        db    1
        dw    offset edrb,offset menu0,offset rdrb  ;drb
        db    1
        dw    offset edrm,offset menu0,offset rdrm  ;drm
        db    1
        dw    offset edf,offset menu0,offset rdrf   ;drf
        db    1
        dw    offset EDI1,offset menu0,offset rdi   ;di
        db    1
        dw    offset edeb,offset menu0,offset rdeb  ;deb
        db    1
```

```
        dw    offset edm,offset mdm,offset rut0      ;dm
        db    3
        dw    offset sup,offset mdm,offset salida
        db    2
        dw    offset edml,offset menu0,offset rdml   ;dml
        db    1
        dw    offset edmb,offset menu0,offset rdmb   ;dmb
        db    1
        dw    offset edmc,offset menu0,offset rdmc   ;dmc
        db    1
        dw    offset edf,offset menu0,offset rdmf    ;dmf
        db    1
        dw    offset edf,offset menu0,offset rdf     ;df
        db    1
        dw    offset ef,offset menu0,offset rut0     ;termina(t)
        db    2
        dw    offset eshell,offset menu0,offset shell     ;shell
        db    1
        dw    offset ensa,offset menu0,offset ensambla ;ensambla
        db    1
        dw    offset edf,offset menu0,offset rf      ;funciones(f)
        db    1


exp0    db    '',0
ean     db    'Permite cargar nuevo programa  a depurar (.EXE o .COM)',0
eal     db    'Resetear archivo actual',0
eag     db    'Graba archivo nombrado en su estado actual',0
ei      db    'Información sobre PRO.EXE',0
ed      db    'Depurador interactivo',0
edp     db    'Ejecuta la instrucción remarcada',0
edc     db    'Corre hasta cumplir un evento: dirección o condición',0
edj     db    'Ejecuta hasta terminar o encontrar un punto de ruptura',0
edr     db    'Puntos de ruptura',0
edrc    db    'Marca un punto de ruptura',0
edra    db    'Activa o desactiva un punto de ruptura',0
edrb    db    'Elimina un punto de ruptura',0
edrm    db    'Muestra la tabla de puntos de ruptura',0
EDI1    db    'Ejecuta una instrucción',0
edeb    db    'Busca una instrucción en el segmento de código actual',0
edm     db    'Operaciones en la ventana principal de memoria',0
edml    db    'Llena desde la posición actual en memoria con un valor',0
edmb    db    'Busca un grupo de datos desde posición actual en memoria',0
edmc    db    'Copia desde posición actual en memoria a otra area',0
edf     db    'Funciones especiales',0
ef      db    'Fin del programa',0
sup     db    'Retornar al nivel superior',0
```

```
eshell db    'Salir temporalmente a DOS , se retorna con EXIT',0
ensa   db    'Ensambla y Enlaza un Archivo .ASM previamente editado con '
       db    '<F6>',0
mensens db     10,13,'**** Ensamblado terminado. Presiona ENTER para Enlazar
o ESC para terminar ****$'
mensens1 db    10,13,'****** Proceso terminado. Presiona una tecla para '
       db    'continuar ******$'
menu0 db     '.. Archivo Infor Depurar!'
       db    ' Salir sHell eXe  Función',0,8,' AIDSHXF',0ffh
md     db    '.. Paso  Corre eJecuta Reinicio pArada! Instr Busca'
       db    ' Memoria! Función',0,10,' PCJRAIBMF',0ffh
mdm    db    '.. Llena  Busca  Copia  Función',0,5,' LBCF',0ffh
mdr    db    '.. Crea  Act/des Borra  Muestra'
       db    ' Función',0,6,' CABMF',0ffh

;************ datos del programa **************
revers   db    70h          ;video reverso
warning db     0f4h
normal  db     7
subbold db     9
bold     db    0fh
bmenu  db     30h          ;atributo para menues
float    db    0
cual2    db    0
cualsn   db    1           ;variables para funciones
overflow db    'instruccion incorrecta, intente otra vez',0
autoseg  dd    ?           ;memoria relativa (PgUp y PgDn)
autooff  dd    ?           ;memoria relativa (PgUp y PgDn)
mem_loc db     94 DUP(0)      ;variables para coprocesador
ST0    dt    0           ;PILA COPROCESADOR
ST1    dt    0
ST2    dt    0
ST3    dt    0
ST4    dt    0
ST5    dt    0
ST6    dt    0
ST7    dt    0
NANPOS  db     '+NAN           ',0
NANNEG  db     '-NAN           ',0
ZERO    db     '+0             ',0
memocop1  db    332 dup(0)    ;memoria para ventana1 del coprocesador
memocop2  db    29  dup(0)    ;memoria para ventana2 del coprocesador
prevdic   dw    0           ;puntero mamoria1 coproc.
copcop   db    0           ;identifica tipo de numero de la pila del cop.
signm    db    0           ;signo de mantisa
signexp  db    0           ;signo exponente
dms      dd    0           ;dato en decimal del exponente
```

```
var2    dd    0          ;resultado de 1/2^n
mantis  dd    0          ;contiene mantisa en hexadecimal
mantisf dt    0          ;mantisa en decimal
statcon dw    0          ;palabra de status del coprocesador
contcon dw    0          ;palabra de control del coprocesador
f3?     db    0          ;Presentar coprocesador o No
f5?     dw    0          ;presentar instruccion completa o No
codhex  dw    0          ;contiene igual valor que regIp
codhex2 dw    0
count   db    0
prevcod dw    0          ;para codigo extendido
videomod      db    0    ;modo de video
cflujo  db    70h,3,7,79h,70h ;atributos para menues en color
bnflujo db    70h,7,7,7,70h   ;y monocromaticos
vntmono db    'winbn.vnt',0   ;archivos de ventanas:monocromatico
vntcolor db   'wincolor.vnt',0 ;a color
archmcr db    'pro486.mcr',0  ;archivo de macros
vlin0   db    'LINEA0',0      ;nombres de las ventanas
vcod    db    'CODIGO',0
vcod2   db    'CODIGO2',0     ;para codigo extendido
vpro1   db    'PRO1',0        ;presentación
vpro2   db    'PRO2',0        ;presentación
vreg    db    'REGIST',0
vpsw    db    'FLAGS',0
vexpc   db    'EXPCOD',0
vstk    db    'STACK',0
vopm1   db    'OPCMEM1',0
vcop0   db    'MEMCOP0',0
vcop1   db    'MEMCOP1',0
vcop2   db    'MEMCOP2',0
vhm     db    'HEXMEM',0
vam     db    'ASCMEM',0
vcc     db    'CCOD',0
vc1     db    'COP1',0
vc2     db    'COP2',0
vc2s    db    'COP2SEG',0
vc2o    db    'COP2OFF',0
vc2h    db    'COP2HEX',0
vc2d    db    'COP2DESP',0
vmemrel db    'MEMREL',0
vdh     db    'DHEX',0
vda     db    'DASC',0
vr      db    'REG',0
vb      db    'DBIN',0
vhas    db    'HASTA',0
vens    db    'ENS',0
vcond   db    'COND',0
```

```
vcban   db     'CBAND',0
vrnom   db     'RUPNOM',0
vrdir   db     'RUPDIR',0
vfnorm  db     'FINNORM',0
vrv     db     'REGVAL',0
vf9     db     'DECHEX',0
vasc1   db     'ASCII1',0
vasc2   db     'ASCII2',0
vnomarch db    'NOMARCH',0
vrup    db     'RUPTURAS',0
vll     db     'LLENAR',0
vcopia  db     'COPIAR',0
vfunc   db     'FUNCION',0
vayp    db     'AYP',0
vaya    db     'AYA',0
vayd    db     'AYD',0
vaye    db     'AYE',0
vaym    db     'AYM',0
vayr    db     'AYR',0
vinfo   db     'INFO',0
vbarra1 db     'BARRA1',0
vbarra2 db     'BARRA2',0
vnarch  db     'NARCH',0
video   db     'VIDIO',0
vcop    db     'COP',0         ;ventana de error de coprocesador no presente
vinst   db     'INST',0        ;ventana de error de instrucción inválida
vnoarchivo db  'NOFILE',0
barra1  db     11 dup('▒')
barra2  db     11 dup('▒')
regist  dw     14 dup(0)       ;buffers para ventanas
codigo  db     14 dup(58 dup(0))
psw1    db     23 dup(0)
expcod  db     36 dup(?)
mem1    db     104 dup(0)
stck    db     18 dup(0)
memo1   db     104 dup(0)      ;direcciones
bufrup  db     168 dup(0)      ;buffer para ventana de rupturas(8*21)
opcseg  db     1,1,1,1,1       ;opciones predefinidas: DS:0,
opcoff  db     5 dup(17)       ;DS:8, ES:10h, DS:18h, DS:20h
xxxxseg dd     ? ;0            ;registro segmento para memoria
xxxxoff dd     ? ;0            ;registro offset para offset
signoff db     '++++++'        ;signo del desplazamiento
dir1    dw     0,0             ;dir. de areas de memoria que se veran
dir21   dw     0,0             ;offset,segmento
dir22   dw     0,0
dir23   dw     0,0
dir24   dw     0,0
```

```
dir25  dw     0,0
dir21r dw     5 dup(0,0)     ;respaldo de dir21...dir25
segcod dw     ?              ;dir. inicial de desensamblado
offcod dw     ?              ;offset inicial (pagina 0)
tabcod dw     11 dup(?)      ;tabla de inicios de paginas de codigo
proxins dw    ?              ;linea que coincide con CS:IP
tabrup db     13 dup(?)      ;1=ruptura en esa linea
banens db     0
toggsub dw    1              ;toggle ingreso a subr.,      , reps
toggcorre db  0              ;toggle corrida hasta direccion(0)/condicion
opcpaso db    'SiNo'         ;opciones: correr subrutinas?
opccorr db    'Direc Cond '  ;opciones: hasta direccion o condicion?
rd6?   db     0              ;bandera para rd600 y rd700
aux    dw     ?,?
ceros  db     10 dup('0')
uno    db     1
uline2 db     27 dup('_')
uline  db     10 dup('_')
uline1 db     '____',0
bufcorr dw    4 dup('__')
dsegm  dw     0
nlinea db     0
dhex3  db     4,7 dup(0)          ;para mem. relativa
dhexmr db     27,28 dup(0)        ;para mem. relativa
dhex1E db     8,13 dup(0)
dhex1  db     4,7 dup(0)
dhex2  db     4,7 dup(0)
ddec   db     10,15 dup(0)
dbin   db     1,0,0
dbyte  db     2,0,0,0,0
insbuff db    79,81 dup(0)        ;para recibir nemotecnico
nombrk db     10,11 dup(0)        ;nombre de breakpoint
archivo db    79,81 dup(0)        ;nombre del Objetivo
InsEns db     20 dup(0)
atrprev db    0
dirprev dw    0
atrprev1 db   0
cual   db     0                   ;variable para los registros
dirprev1 dw   0
nomreg db     3,0,0,0,0
rupact db     0                   ;ruptura actual - 1
brup   db     0                   ;bandera para rut. de rupturas
StackLow dw   ?
StackHi  dw   ?
siarch   db   0
bufm1    dw   7 dup(0)
chk1     db   6ch
```

```
chk21   db      13
chk22   db      33
regs    db      'DSESSSCSAXBXCXDXSIDIBP'
linea0  db      44 dup(' '),20 dup(0)
linea1  db      80 dup(' ')
l0ini   db      '<^O=Menúes> <F2=Ayuda>',0
F7Direcc  db      'Direc',0
F7Condic  db      'Cond',0
F8SiSubr  db      'SiSubr',0
F8NoSubr db       'NoSubr',0
l1ini   db      'F1Edit F2Ayuda F3Coproc F4PntUsr '
        db      'F5Cod F6Asm F7Direc F8NoSubr F9Dec/Hex F10ASCII',0
nomodo  db      'Modo de video no soportado.',0
l23d_   db      'TAB: próxima ventana  SHIFT TAB: ventana'
        db      ' anterior  ESC: menú principal',0
l24d_   db      'Acceso a las ventanas del depurador',0
l0reg   db      'Alterar valor registros',0
l1reg   db      'Use las flechas para pasar de uno a otro registro',0
l0ban   db      'Alterar valor banderas',0
l1ban   db      'Use las flechas ',26,' y ',27,' para pasar a otra bandera',0
l0cod   db      'Desensamblador: Página ',0
l1cod   db      26,' y ',27,' : pasar al otro campo.  '
        db      'PGUP y PGDN : cambiar de página',0
l1cod1  db      '** ATENCION: PGDN: próxima página   PGUP: página 9 **',0
l0op1   db      'Acceso a memoria',0
l0op21  db      'Acceso a memoria relativa',0
l1op2   db      '[ y ] : Cambio de página (Segmento).'
        db      '  PGDN y PGUP : Cambio de página (Offset)',0
l0op22  db      'Offset para ventana de memoria relativa',0
l1op2d  db      'Escriba el desplazamiento con su signo',0
l1op2o  db      'Escriba el offset',0
l1op2s  db      'Escriba el segmento',0
l0hex   db      'Alterar  memoria',0
l1hex   db      'Flechas: movimiento en las ventanas.'
        db      '  PGUP y PGDN: cambio de páginas',0
l0asc   db      'CR=ENTER',0    ;TAB=[F10]TAB,ESC=[F10]ESC',0
l23ens  db      'Dirección=     :    . Escriba'
        db      ' nemotécnico o ESC para salir',0
errens  db      'Este nemotécnico no es correcto',0
EnsBusc  db       'Escriba el nemotécnico de la instrucción que desea encontrar',0
NoEnsB  db      'No se encuantran  mas coincidencias  en el segmento de código',0
EnsW1   db      'Es posible encontrar el código de la  instrucción, '
        db      'pero el flujo',0
EnsW2   db      '  del programa no  sera necesariamente  '
        db      'de la forma mostrada.',0
instruc   db      'Escriba el nemotécnico de la instrucción a ejecutarse',0
l23bkp    db       'PGUP y PGDN: Pasar de página en ventana de rupturas.   ESC:
```

```
Salir',0
l23cd   db      'TAB: Pasar al otro campo.  ESC: Salir.',0
l23cc   db      24,' y ',25,' : Pasar a otra opción.'
        db      '  ESC: Salir.  ENTER: Opción',0
l23rcmp db      'Escriba el registro , o ESC para salir',0
l23cmpr db      '...y ahora el valor, o ESC para terminar',0
igual   db      '           ',0
noesreg db      'Este no es un registro. Presiona una tecla'
        db      ' o ESC para salir',0
nombre? db      'Escriba el nombre del archivo a depurar (.EXE o .COM)  ESC =
salir',0
leo     db      'Leyendo...',0
nonomb  db      'Aun no se ha ingresado el nombre del archivo.  Presione una tecla ',0
nohay   db      'Este archivo no existe. Presione una tecla',0
long0   db      'No puedo aceptar tamaño 0',0
errgrab db      'Error al grabar el archivo',0
valor?  db      'Escriba el valor o ESC para salir',0
bytes?  db      'Escriba el número de bytes o ESC para salir',0
buscasc? db     'Secuencia en ASCII o hexadecimal (A/H)?',0
busca?  db      'Escriba la secuencia o ESC para salir',0
NoHayStr db     'No encuentro la secuencia en todo el segmento',0
sigo?   db      'ENTER: Busca próxima.   ESC: Termina.',0
noesbrk db      'No existe ese punto en la lista.   Presione una tecla',0

;MENSAJES AL MOMENTO DE LA CARGA
NoVnt   db      'El archivo de ventanas ha sido alterado',0
ErrAbrir db     'No puedo abrir archivo de ventanas',0
ErrMcr  db      'Error al abrir/crear archivo de macros',0
NoChild db      'No encuentro todos mis programas asociados',0
ArchOK  db      '<Objetivo> listo',0
NoParam db      '*** Atención: No tengo <Objetivo>... ***',0
MalParam db     'Archivo a depurar no encontrado. Presione una tecla',0
MalExec db      'Problemas en la carga del <Objetivo>...',0
Listo   db      'Operación terminada. Presione una tecla.',0
TecSal  db      'para salir',0
TecCont db      'Presione una tecla para continuar',0

;MENSAJES DE ERROR  AL CORRER PROGRAMAS
NoFuncion db    'Número de función no válida <presiona una tecla> ','$'
Noarch  db      'Archivo editor no encontrado <presiona una tecla> ','$'
Nopath  db      'Path no encontrado <presiona una tecla> ','$'
Noacceso db     'acceso denegado <presiona una tecla> ','$'
Memoriades db   ' Bloques de control de memoria destruídos '
        db      '<presiona una tecla>','$'
Nomemoria db    'Memoria insuficiente <presiona una tecla> ','$'
Noblock db      'Dirección de bloque de memoria inválido '
        db      '<presiona una tecla>','$'
```

```
Noambiente  db  'Ambiente inválido <presiona una tecla> ','$'
NoFormato   db   Formato inválido <presiona una tecla> ','$'

mensajes     dw 0     ;desplazamiento de los mensajes de error
         dw offset Nofuncion
         dw  offset noarch
         dw  offset nopath
         dw  0
         dw  offset noacceso
         dw  0
         dw  offset memoriades
         dw  offset nomemoria
         dw  offset noblock
         dw  offset noambiente
         dw  offset noformato
varea   db     50*256 dup(0)   ;area de ventanas
;valores originales de los vectores de interrupción 06h y 07h
old6h  dw 0
         dw 0
old7h  dw 0
         dw 0
error    db 0
errorcj db 0
;Variables para correr Edit.com,Command.com, Masm.exe, Link.exe
narch2  db   40,40 dup(' ')

ml1      db   41,' ' ;              ;parámetros para LINK.exe
narch3   db   40 dup(' '),0dh

Parametros   db   11,' pro486.hlp',0dh    ;parámetros par F2
Parametros1 db   1,' ',0dh             ;parámetros para COMMAND.COM

nombre3   db    'link.exe',0  ;path de Link.exe
Nombre2  db    'masm.exe',0  ;path de masm.exe
Nombre1  db    'command.com',0   ;path de command.com
ambiente   dw   0 ;guarda el valor del bloque de ambiente desde psp (4Ch)
Nombre    db    'edit.com',0          ;path de edit.com

narch1     db   41,' '               ;parámetros para F6 y para MASM.exe
Narch      db   40 dup (' '),0dh

viejosp    dw  0                    ;guarda sp antes de hacer EXEC
viejoss    dw  0                    ;guarda ss antes de hacer EXEC

FCbloque1  db 0                      ;bloques de control de archivo
             db 11 dup(0)
             db 25 dup(0)
```

```
FCbloque2  db 0
           db 11 dup(0)
           db 25 dup(0)
FCbloque3  db 0
           db 11 dup(0)
           db 25 dup(0)
FCbloque4  db 0
           db 11 dup(0)
           db 25 dup(0)
PARMBLOQUE STRUCT                ;Bloque de parámetros para la función
           ambiente1       dw 0  ;4Bh de DOS
           paramaddress    dw 0
           paramsegmento   dw 0
           fcb1offset      dw 0
           fcb1segmento    dw 0
           fcb2offset      dw 0
           fcb2segmento    dw 0
PARMBLOQUE ENDS

pb PARMBLOQUE <0,0,0,0,0,0,0>
pb1 PARMBLOQUE <0,0,0,0,0,0,0>

; variables para uso del raton
ventActual  db 0  ; indica la ventana activa actual
VentNueva   db 0  ; indica la nueva ventana seleccionada
vent        db     0
ejecutamenu db 0
pixely    db 8 ;indican el número de pixel horixontal y vertical por caracter
pixelx    db 8 ;inicialmente se asume de 8x8 para pantalla a color de 80x25
ratonpresente db 0
var1    STRUCT
          var11 dw 0
          var12 dw 0
var1    ENDS
varraton1 var1 <0,0>

        .code
        public rinic,rfinal,funciones
        extrn  pantalla:proc,savepspchild:proc,restpspchild:proc
        extrn  desensdet:proc,AcortaMiMem:proc
        extrn  InstHandler63:proc,SetTerm:proc
        extrn  LoadExe:proc,ReduceLevel:proc
        extrn  ExUniq:proc,ExHasta:proc,ExPaso:proc,ExCond:proc
        extrn  Assem:proc,ExTotal:proc,testtrap:proc
        extrn  AddBrkPt:proc,DelBrkPt:proc,TogBrkPt:proc
        extrn  EncReg:proc,EncSreg:proc,ExIns:proc
        extrn  llenar:proc,buscar:proc,copiar:proc
```

```
        include provnt.asm
        include mpro.asm
        include mouse.asm
          sbrvnt
          sbrmcr
          sbrpro

retorno macro                   ;construye direccion de
        cld                     ;regreso y salta.
        mov   ax,@data
        mov   ds,ax
        mov   ax,videoseg
        mov   es,ax
        mov   ax,mraux3
        mov   bp,offset dirret
        mov   ds:[bp],ax
        mov   ax,mraux2
        mov   ds:[bp+2],ax
        jmp   dword ptr dirret
        endm


;*************** rutinas del programa ************

;Regreso general de todas las opciones a FLUJO
Salida:
        mov   ejecutamenu,0
        retorno
rinic:  call  testtrap      ;para elegir rutina de paso a paso
        call  redefvect     ;Redefine vectores 06h y 07h, instrucción
                            ;no válida y coporcesador no presente
rini1:  mov   ah,0fh        ;determino modo de video
        int   10h           ;en AL
        push  es
        mov   bx,40h        ;ubico segmento de info
        mov   es,bx
        mov   ah,es:[10]    ;veo tipo de monitor
        pop   es
        cmp   al,2          ;acepto modos 80*25 de:16 grises
        mov   videomod,al
        je    esbn
        cmp   al,3          ;color
        je    modook
        cmp   al,7          ;o blanco y negro
        je    modook
        jmp   modono
esbn:
```

```
        mov     pixelx,9
        mov     pixely,14
        jmp     modook
modono: print   1,23,nomodo,revers      ;modo no soportado, fin
        jmp     rinif2
modook: and     ah,18h          ;solo dejo los bits necesarios
        jz      rini12          ;si son 0's, es de color
        idvnt   vntmono,varea,50
        mov     aux,ax
        mov     si,offset bnflujo ;atributos B/N para menues
        jmp     rini13
rini12: idvn    vntcolor,varea,50
        mov     aux,ax
        mov     si,offset cflujo ;atributos de color para menues
rini13: mov     cx,5
        mov     di,offset colven
        push    es
        push    ds
        pop     es
        rep     movsb
        pop     es
        mov     ax,aux
        cmp     ax,0            ;todo bien?
        je      rini1b
        push    ax
        call    presen          ;no, presenta el programa
        call    pres0
        print   5,21,TecSal,bold ;da mensaje y sale
        pop     ax
        cmp     ax,1
        jne     rini11
        print   5,20,ErrAbrir,bold ;error al abrir o no hay
        jmp     rinif1
rini11: print   5,20,NoVnt,bold ;No hay espacio o no es de ventanas
        jmp     rinif1
rini1b:
        inimcr  archmcr         ;leo/creo archivo de macros predefinido
        cmp     ax,0            ;todo bien?
        jne     rinm1
        jmp     rinmf
rinm1:  cmp     ax,2            ;nuevo?
        jne     rinm2
        creamcr archmcr         ;lo creo
        jc      rinm2
        jmp     rinmf
rinm2:  call    presen          ;error
        call    pres0
```

```
        print   5,20,ErrMcr,bold
        print   5,21,TecSal,bold
        jmp     rinif1
rinimf: mov     StackLow,sp     ;todo bien con ventanas y macros
        mov     StackHi,ss      ;trato de cargar objetivo
        call    AcortaMiMem
        call    InstHandler63
rini2:  call    ReduceLevel
        jnc     rini3
        mov     ax,@data        ;No pudo bajar de nivel
        mov     ds,ax
        mov     es,videoseg
        lds     dx,Vector2F     ;restauro int 63h
        mov     ax,2563h
        int     21h
        mov     ax,@data
        mov     ds,ax
        mov     es,videoseg
        call    presen
        call    pres0
        print   5,20,NoChild,bold
        print   5,21,TecSal,bold
        jmp     rinif1
rini3:  call    LoadExe
        mov     ax,@data
        mov     ds,ax
        mov     es,videoseg
        jnc     rini3b
rini3m:
        call    presen              ;error al cargar programa, presento
        call    pres0
        print   5,21,TecSal,bold    ;doy mensaje y salgo
        cmp     FLoadExe,4
        jne     rini3m1
        print   5,20,MalParam,bold
        jmp     rinif
rini3m1:print   5,20,MalExec,bold
        jmp     rinif
rini3b:
        mov     sp,StackLow
        mov     ss,StackHi
        mov     ax,regds
        mov     xxxxseg,Eax         ;tal que inicie con Ds
        mov     dhexseg,eax
        mov     autoseg,eax
        mov     xxxxoff,0
        mov     dhexoff,0
```

```
        mov    autooff,0
        call   presen              ;presenta el programa
        call   pres0
        print  5,21,TecCont,bold
        cmp    FLoadExe,3          ;no param?
        je     rini31
        mov    siarch,0ffh
        print  5,20,ArchOK,bold
        jmp    rini32
rini31: print  5,20,NoParam,bold
rini32: getkey
      mov    si,offset l0ini       ;preparo y muestro linea 0
      call   mlin0
      limpln 23,normal
      print  0,23,l1ini,bmenu
rinif0: call preopc                ;prepara buffers MEMO1 y MEMO2
        call   pdebug
        call   delay
        retn

rinif: getkey
        call  rfinal               ;salida por errores desde nivel hijo
        jmp  rinif2
rinif1: getkey
rinif2: mov    ah,2                ;idem desde nivel padre
        mov    dl,7
        int    21h                 ;pitar
        mov    ah,4ch              ;y terminar
        int    21h

presen: mov    dl,0ah
        mov    cx,25
presl:  mov    ah,2                ;limpio pantalla
        push   cx
        int    21h
        pop    cx
        loop   presl
        mov    bx,offset vpro1
        vnt    'a'
        mov    bx,offset vpro2
        vnt    'a'
        retn
pres0:  posdir 0,19
        call   pres00
        posdir 0,22
pres00: mov    di,pddir
        mov    cx,79
```

```
        mov     al,0cdh
        mov     ah,bold
        rep     stosw
        retn

delay:  mov     ax,4
delay1: mov     cx,0ffffh
delay2: loop    delay2
        dec     ax
        jnz     delay1
        retn

rfinal: mov     di,0                ;limpio pantalla al salir
        mov     cx,2000
        mov     ax,0720h
        rep     stosw
        grabmcr                     ;graba macros
        cerrmcr
        lds     dx,Vector2F         ;restauro int 63h
        mov     ax,2563h
        int     21h
        mov     ax,@data
        mov     ds,ax
        mov     Stacklow,sp         ;salvo stack y subo de nivel
        mov     Stackhi,ss
        mov     dx,offset rfin1
        call    SetTerm
        mov     ah,4ch
        int     21h
rfin1:  mov     ax,@data            ;recupero stack y regreso a terminar
        mov     ds,ax
        mov     es,videoseg
        mov     ss,stackhi
        mov     sp,stacklow
        retn

rut0:
        jmp     salida

rf:                                 ;ayuda menu principal
        cmp     otroaux,0           ;ENTER, pasar
        je      rff
        cmp     mopaux,0
        jne     rff
        cmp     mopaux1,60          ;F2?
        jne     rff
        stodo
```

```
        mov     ax,@data
        mov     ds,ax
        mov     bx,ambiente
        mov     pb.ambiente1,bx
        mov     bx,offset parametros
        mov     pb.paramaddress,bx
        mov     pb.paramsegmento,ax
        mov     bx,offset fcbloque1
        mov     pb.fcb1offset,bx
        mov     pb.fcb1segmento,ax
        mov     bx,offset fcbloque2
        mov     pb.fcb2offset,bx
        mov     pb.fcb2segmento,ax
        mov     bx,offset pb
        push    ds
        mov     es,ax
        mov     di,pb.fcb1offset
        mov     si,pb.ambiente1
        inc     si
        mov     ax,2901h
        int     21h
        pop     es
        mov     di,pb.fcb2offset
        mov     ax,2901h
        int     21h
        push    bp
        mov     viejosp,sp
        mov     viejoss,ss

        mov     ah,62h          ;obtener dirección de mi PSP
        int     21h

        mov     es,bx
        mov     bx,2000h        ;liberar memoria para ejecutar otros procesos
        mov     ah,4ah
        int     21h
        jnc     simemoria1
        call    menserror
        jmp     biencorrido1
simemoria1:
        push    ds
        pop     es
        mov     bx,offset pb
        mov     dx,offset nombre
        mov     ax,4b00h
        int     21h
        jnc     biencorrido1
```

```
        call    menserror
bienconrido1:
        mov     ss,viejoss
        mov     sp,viejosp
        pop     bp
        mov     visible,0
        mmuestra 1,visible,ratonpresente
        mov     ah,2
        mov     bh,0
        mov     dh,24
        mov     dl,0
        int     10h
        rtodo
        call    pdebug
        jmp     salida
rff:    jmp     rdfun           ;otras funciones
raf:                    ;ayuda menu archivos
        cmp     otroaux,0       ;ENTER, pasar
        je      raff
        cmp     mopaux,0
        jne     raff
        cmp     mopaux1,60      ;F2?
        jne     raff
        mov     bx,offset vaya
        vnt     'a'
        limpln  24,normal
        getchar
        call    pdebug
        jmp     salida
raff:   jmp     rdfun           ;otras funciones

rdf:                    ;ayuda menu depurador
        cmp     otroaux,0       ;ENTER, pasar
        je      rdxf
        cmp     mopaux,0
        jne     rdxf
        cmp     mopaux1,60      ;F2?
        jne     rdxf
        mov     bx,offset vayd
        vnt     'a'
        limpln  24,normal
        getchar
        call    pdebug
        jmp     salida
rdxf:   jmp     rdfun           ;otras funciones
rdef:                   ;ayuda menu ensamblador
        cmp     otroaux,0       ;ENTER, pasar
```

```
        je      rdeff
        cmp     mopaux,0
        jne     rdeff
        cmp     mopaux1,60      ;F2?
        jne     rdeff
        mov     bx,offset vaye
        vnt     'a'
        limpln  24,normal
        getchar
        call    pdebug
        jmp     salida
rdeff:  jmp     rdfun           ;otras funciones
rdmf:                   ;ayuda menu memoria
        cmp     otroaux,0       ;ENTER, pasar
        je      rdmff
        cmp     mopaux,0
        jne     rdmff
        cmp     mopaux1,60      ;F2?
        jne     rdmff
        mov     bx,offset vaym
        vnt     'a'
        limpln  24,normal
        getchar
        call    pdebug
        jmp     salida
rdmff:  jmp     rdfun           ;otras funciones
rdrf:                   ;ayuda menu rupturas
        cmp     otroaux,0       ;ENTER, pasar
        je      rdrff
        cmp     mopaux,0
        jne     rdrff
        cmp     mopaux1,60      ;F2?
        jne     rdrff
        mov     bx,offset vayr
        vnt     'a'
        limpln  24,normal
        getchar
        call    pdebug
        jmp     salida
rdrff:  jmp     rdfun           ;otras funciones
rinfo:                  ;informacion
        lmppnt                  ;limpio pantalla
        mov             bx,offset vinfo ;muestro ayuda
        vnt     'a'
        limpln  23,revers
        limpln  24,normal
        print   1,23,TecCont,revers
```

```
        getchar
        call    pdebug
        mov     si,offset l0ini ;preparo y muestro linea 0
        call    mlin0
        limpln  23,normal
        print   0,23,l1ini,bmenu
rinifl1: jmp    salida

ran:                    ;recibe el nombre del Objetivo
        limpln  23,revers
        limpln  24,normal
        print   1,23,nombre?,revers
        mov     gdreset,1
rnl:    getline 1,24,archivo
        mov     gdreset,0
        cmp     gccod,4         ;ESC?
        je      rnfl
        cmp     gccod,1         ;valido?
        jne     rnl
        cmp     archivo+1,0     ;vacio?
        je      rnfl
        mov     si,offset archivo+1     ;vale
        push    es              ;paso a PSP del padre y a FileName
        mov     es,PSPparent
        mov     di,80h
        mov     cx,80
        rep     movsb
        mov     ax,@data
        mov     es,ax
        mov     di,offset FileName
        mov     si,offset archivo+2
        mov     cx,80
        rep     movsb
        pop     es

rnf:
      print     0,23,l1ini,bmenu
      push      es
      push      di
      push      cx
      push      ax
      mov       es,ax
      mov       di,offset TablaBrk ;primero borro buffer
      mov       cx,3096
      mov       al,0
      rep       stosb
      pop       ax
```

```
        pop     cx
        pop     di
        pop     es
        jmp     ral
rrnfl:
        print   0,23,l1ini,bmenu
        jmp     salida

ral:                    ;lee el Objetivo desde disco
        mov     error,0
        mov     StackLow,sp     ;salvo SS:SP
          mov   StackHi,ss
          push  ax
          xor   ax,ax
          mov   fs,ax
          mov   gs,ax
          pop   ax
          limpln 23,revers
          limpln 24,normal
          push  es
          mov   es,PSPparent
          cmp   byte ptr es:[80h],1
        ; pop   es
          jae   rrla
          print 1,23,nonomb,revers      ;aun no tengo nombre
          pop   es
          jmp   rrlf
rrla:     pop   es
           print 1,23,leo,revers ;leyendo...
          call  LoadExe
          mov   ax,@data
          mov   ds,ax
        mov     es,videoseg
        jnc     rrlb
rrlm:
        mov     siarch,0  ;con 0 indica que no se ha cargado archivo
        push    es
        push    ds
        pop     es
        mov     di,offset filename
        mov     cx,80
        mov     al,' '
        rep     stosb
        pop     es
        popf
        cld
        cmp     FLoadExe,4      ;errores en LoadExe
```

```
        jne     rrlm1
        limpln  23,revers
        print   1,23,MalParam,warning
        jmp     rrlb1
rrlm1:  print   1,23,MalExec,revers
        jmp     rrlf
rrlb:
        mov     siarch,0ffh   ;con ffh indica que se ha cargado archivo
        mov     sp,StackLow     ;lectura OK
        mov     ss,StackHi
        print   1,23,listo,revers
rrlb1:
        call    pdebug
        getchar
rrlf:   mov     si,offset l0ini ;preparo y muestro linea 0
        call    mlin0
        limpln  23,normal
        print   0,23,l1ini,bmenu
rinif22:jmp     salida
nomarch:
        mov     ax,ds
        mov     si,offset filename
        mov     bx,offset vnomarch
        vnt     'a'
        retn
mlin0: call     prelin          ;preparo linea 0
        mov     ax,ds
        mov     bx,offset vlin0
        mov     si,offset linea0
        vnt     'a'
        retn

pdebug:
        call    mreg            ;prepara y muestra: REGIST
        cmp     banens,1        ;Si BANENS=1,no mostrar codigo
        je      pdeb21
        mov     ax,regcs
        mov     segcod,ax
        mov     Eax,regip       ;si REGIP esta en pagina actual,
        cmp     ax,offcod       ;no cambiar de inicio de pagina
        jb      pdeb1
        cmp     ax,proxins
        jb      pdeb2
pdeb1:  mov     offcod,ax
pdeb2:  call    mcod            ;CODIGO
        call    nomarch
pdeb21:
```

```
        call    mmem            ;MEM
        call    mstk            ;stck  (stack)
        call    mexpc           ;EXPCOD
        call    mreg            ;prepara y muestra: REGIST
        call    mpsw            ;PSW1  (banderas)
        cmp     f5?,1           ;Presentar instrucciones completas?
        jne     copf3?          ;si f5?=1 si
        call    mcod
        call    nomarch
copf3?: cmp     f3?,1                   ;debo presentar ventana del coproc?
        jne     cop41                   ;no  f3?=0--> no presento
        call    rdf3m                   ;si  f3?=1--> lo presento
        mmuestra 0,visible,ratonpresente
        mov     ax,ds
        mov     bx,offset vcop0
        vnt     'a'
        mov     ax,ds
        mov     bx,offset vcop1         ;presento datos pila coprocesador
        mov     si,offset memocop1
        vnt     'a'
        mov     ax,ds
        mov     bx,offset vcop2         ;presento stat. y cont. del coproc.
        mov     si,offset memocop2
        vnt     'a'
        mmuestra 1,visible,ratonpresente
cop41:  cmp     FinNormal,1     ;acabo?
        jne     pdeb3
        mov     bx,offset vfnorm ;ventana de terminacion
        vnt     'a'
        getchar
        xor     FinNormal,1
        jmp     pdebug
pdeb3:  retn

funciones:
      limpln    23,normal
      print     0,23,l1ini,bmenu
rinif44:  retn

mcod:   push    ds              ;Preparo el buffer CODIGO con 13
        push    es              ;instrucciones desensambladas
        mov     si,offcod       ;y lo muestro en pantalla
        mov     di,offset codigo ;En PROXINS sale offset de prox.
        mov     ax,@data        ;instruccion
        mov     es,ax
        push    di
        mov     cx,14*58        ;limpio buffer
```

```
        mov   al,20h
        rep   stosb
        pop   di
        push  di          ;limpio TABRUP
        mov   di,offset tabrup
        mov   cx,13
        mov   al,0
        rep   stosb
        pop   di
        push  di
        mov   cx,segcod     ;guardo el valor del segmento que
        mov   al,ch         ;ira al inicio de todas las lineas
        hexasc
        mov   al,cl
        mov   cx,bx
        hexasc
        mov   ax,cx
        xchg  al,ah
        xchg  bl,bh
        mov   aux,ax
        mov   aux+2,bx
        pop   di
        mov   dl,14         ;13-DL=# de linea (0-12)
        mov   dh,0ffh       ;# linea a resaltarse
pcod0:  mov   ax,segcod
        call  pcr0
        cmp   ax,regcs      ;coincide el CS?
        jne   pcod01
        cmp   Esi,regip     ;coincide el IP?
        jne   pcod01
        mov   dh,14         ;si ambos coinciden, linea debe resaltarse
        sub   dh,dl
pcod01: mov   ax,aux        ;escribo segmento
        stosw
        mov   ax,aux+2
        stosw
        mov   ds,segcod
        call  desensdet     ;desensamblo linea
        mov   bx,@data
        mov   ds,bx
        add   si,cx
        add   di,54
        cmp   byte ptr [di-2],' '
        jne   pcod3         ;Si la linea no es muy larga
        cmp   al,0          ;pongo el tipo de instruccion.
        jne   pcod1         ;AL = 0 --> byte ptr
        mov   byte ptr [di-1],'B'
```

```
        jmp     pcod3
pcod1:  cmp     al,1            ;AL = 1 --> word ptr
        jne     pcod2
        mov     byte ptr [di-1],'W'
        jmp     pcod3
pcod2:  cmp     al,2            ;AL = 2 --> dword ptr
        jne     pcodq           ;AL = 255 --> nada
        mov     byte ptr [di-1],'D'
        jmp     pcod3
pcodq:  cmp     al,3
        jne     pcodt
        mov     byte ptr [di-1],'Q'
        jmp     pcod3
pcodt:  cmp     al,4
        jne     pcod3
        mov     byte ptr [di-1],'T'
pcod3:  cmp     dl,9
        jne     pcod31
        push    ax
        mov     ax,di
        mov     prevcod,ax
        pop     ax
pcod31: cmp     dl,2
        jne     cod6
        mov     proxins,si
cod6:   dec     dl
        jnz     pcod0
cod7:   pop     es
        pop     ds
        mov     ax,ds           ;ventana Codigo.
        mov     bx,offset vcod
        mov     si,offset codigo
        vnt     'a'
        mov     ax,ds
        mov     bx,offset vcod2
        push    ax
        mov     ax,prevcod
        mov     si,ax
        pop     ax
        vnt     'a'

codfin: cmp     dh,0ffh
        je      pcod5           ;no hay linea que resaltar
        add     dh,3
      mov       dl,11
        posdir dl,dh
        mov     di,pddir
```

```
        inc     di
        mov     cx,26           ;resalto linea CS:IP
        mov     al,07h
pcod4:  stosb
        inc     di
        loop    pcod4
pcod5:  call    pcr00           ;resalto bkpts
        retn

code?:  pusha
        cmp     f5?,1
        je      code0
        call    mreg            ;no
        call    mpsw
        call    mmem
        jmp     code1
code0:  mov     ax,ds           ;ventana Codigo.
        mov     bx,offset vcod
        mov     si,offset codigo
        vnt     'a'
        mov     ax,ds
        mov     bx,offset vcod2
        push    ax
        mov     ax,prevcod
        mov     si,ax
        pop     ax
        vnt     'a'
        call    nomarch
code1:  popa
        retn
pcr0:   ;veo si AX:SI pertenecen a TABLABRK y si es asi,
        ;pongo un 1 en la entrada respectiva de TABRUP si
        ;es punto activo o 2 si es inactivo
        stodo
        mov     di,offset TablaBrk
        mov     ch,0            ;CH=# del bkpt buscado
pcr01:  mov     bl,ch
        mov     bh,0
        mov     cl,4
        shl     bx,cl           ;multiplico por 16
        cmp     byte ptr[di+bx],0 ;veo si termine la tabla
        je      pcr03
        cmp     [di+bx+12],si   ;offset igual?
        jne     pcr02           ;no
        cmp     [di+bx+14],ax   ;si, segmento igual?
        jne     pcr02           ;no
        mov     al,0
```

```
        cmp     [di+bx+1],al    ;si, hay un bkpt. Desactivado?
        jne     pcr011
        mov     al,1            ;si
pcr011: inc     al              ;AL=1 activos, 2 inactivos
        mov     bl,13
        sub     bl,dl
        mov     di,offset TabRup
        mov     bh,0            ;BX+DI apunta a entrada en TABRUP
        mov     byte ptr[di+bx],al
        jmp     pcr03           ;salgo
pcr02:  inc     ch              ;preparo siguiente busqueda
        jnz     pcr01
pcr03:  rtodo
        retn
pcr00:  ;busco en TABRUP las lineas que tienen bkpts y las resalto.
        mov     nlinea,0
        mov     si,offset TabRup
pcr001: mov     bl,nlinea       ;para cada linea veo si debo resaltar
        mov     bh,0
        cmp     byte ptr[bx+si],0
        je      pcr002          ;no resalto
        mov     l,2h            ;resalto
        cmp     byte ptr[bx+si],1 ;es activo?
        jne     pcr003
        mov     dl,4h           ;si, video reverso; no, subrayado
pcr003: mov     ch,bl
        add     ch,4
        posdir  1,ch
        mov     di,pddir
        inc     di
        mov     al,dl
        call    rdcd01          ;resalto 4 casillas
pcr002: inc     nlinea
        cmp     nlinea,13
        jne     pcr001
        retn

prelin: mov     bx,di           ;preparo LINEA0
        mov     di,offset linea0
        push    es
        push    ds
        pop     es
        mov     al,' '          ;primero borro
        mov     cx,44
        rep     stosb
prl1:   mov     cx,44           ;escribo LINEA0
        mov     di,offset linea0
```

```
prl11: lodsb
       cmp    al,0
       je     prl2          ;salgo al encontrar un 0
       stosb
       loop   prl11         ;o escribir 44 bytes
prl2:
       pop    es
       retn
mreg:  mov    di,offset regist
       push   es            ;preparo buffer REGIST
       push   ds
       pop    es

       mov    Eax,RegIp
       push   ax
       push   cx
       mov    cl,16
       shr    eax,cl
       xchg   al,ah
       stosw
       pop    cx
       pop    ax
       xchg   al,ah
       stosw

     mov    Eax,RegAx
       push   ax
       push   cx
       mov    cl,16
       shr    eax,cl
       xchg   al,ah
       stosw
       pop    cx
       pop    ax
       xchg   al,ah
       stosw

       mov    eax,RegSp
       push   ax
       push   cx
       mov    cl,16
       shr    eax,cl
       xchg   al,ah
       stosw
       pop    cx
       pop    ax
       xchg   al,ah
```

```
        stosw
        mov     eax,RegBx
        push    ax
        push    cx
        mov     cl,16
        shr     eax,cl
        xchg    al,ah
        stosw
        pop     cx
        pop     ax
        xchg    al,Ah
        stosw

        mov     eax,RegBp       ;Leo un registro del usuario
        push    ax
        push    cx
        mov     cl,16
        shr     eax,cl
        xchg    al,ah
        stosw
        pop     cx
        pop     ax
        xchg    al,ah           ;pongo MSB antes de LSB
        stosw                   ;y guardo en el buffer.
        mov     eax,RegCx
        push    ax
        push    cx
        mov     cl,16
        shr     eax,cl
        xchg    al,ah
        stosw
        pop     cx
        pop     ax
        xchg    al,ah
        stosw
        mov     eax,RegSi
        push    ax
        push    cx
        mov     cl,16
        shr     eax,cl
        xchg    al,ah
        stosw
        pop     cx
        pop     ax
        xchg    al,ah
        stosw
        mov     Eax,RegDx
```

```
        push    ax
        push    cx
        mov     cl,16
        shr     eax,cl
        xchg    al,ah
        stosw
        pop     cx
        pop     ax
        xchg    al,ah
        stosw
        mov     eax,RegDi
        push    ax
        push    cx
        mov     cl,16
        shr     eax,cl
        xchg    al,ah
        stosw
        pop     cx
        pop     ax
        xchg    al,ah
        stosw
        mov     eax,Eflags
        push    cx
        mov     cl,16
        shr     eax,cl
        xchg    al,ah
        stosw
        pop     cx
        pop     ax
        xchg    al,ah
        stosw
        mov     ax,RegFs
        xchg    al,ah
        stosw
        mov     ax,RegGs
        xchg    al,ah
        stosw
        mov     ax,RegCs
        xchg    al,ah
        stosw
        mov     ax,RegDs        ;El orden de los registros es el
        xchg    al,ah           ;adecuado para la ventana REGISTRO.
        stosw
        mov     ax,RegEs
        xchg    al,ah
        stosw
        mov     ax,RegSs
```

```
        xchg    al,ah
        stosw
        pop     es
        mov     ax,ds           ;ventana Registro
        mov     bx,offset vreg
        mov     si,offset regist
        vnt     'h'
        retn

mpsw:   push    es              ;preparo buffer PSW1
        push    ds
        pop     es
        mov     ax,'0'          ;limpio el buffer
        mov     cx,8
        mov     di,offset psw1
        rep     stosb
        mov     di,offset psw1
        mov     bx,flags
        mov     cl,5
        rcl     bx,cl
        jnc     ppswo
        mov     ax,'1'
ppswo:  stosb                   ;O
        mov     ax,'0'
        rcl     bx,1
        jnc     ppswd
        mov     ax,'1'
ppswd:  stosb                   ;D
        mov     ax,'0'
        rcl     bx,1
        jnc     ppswi
        mov     ax,'1'
ppswi:  stosb                   ;I
        mov     ax,'0'
        rcl     bx,1
        rcl     bx,1
        jnc     ppsws
        mov     ax,'1'
ppsws:  stosb                   ;S
        mov     ax,'0'
        rcl     bx,1
        jnc     ppswz
        mov     ax,'1'
ppswz:  stosb                   ;Z
        mov     ax,'0'
        rcl     bx,1
        rcl     bx,1
```

```
        jnc     ppswa
        mov     ax,'1'
ppswa:  stosb                   ;A
        mov     ax,'0'
        rcl     bx,1
        rcl     bx,1
        jnc     ppswp
        mov     ax,'1'
ppswp:  stosb                   ;P
        mov     ax,'0'
        rcl     bx,1
        rcl     bx,1
        jnc     ppswc
        mov     ax,'1'
ppswc:  stosb                   ;C
        pop     es
        mov     ax,ds           ;Ventana Psw
        mov     bx,offset vpsw
        mov     si,offset psw1
        vnt     'a'
        retn
mmem:

mmemx:

mmem1:  mov     di,offset mem1
        mov     dx,ds
        push    es
        push    ds
        pop     es              ;ds=es
        mov     cx,52
        mov     eax,xxxxoff
        mov     si,ax
        mov     eax,xxxxseg
        xchg    ax,dx
        mov     ds,dx
mmem11: movsw
        dec     cx
        cmp     cx,0
        jne     mmem11
        xchg    ax,dx
        mov     ds,dx
        pop     es
        mov     ax,ds           ;ventana Opcmem1
        mov     bx,offset vopm1
        mov     si,offset memo1
        vnt     'a'
```

```
        mov   ax,ds          ;ventana Hexmem
        mov   bx,offset vhm
        mov   si,offset mem1
        vnt   'h'
        mov   ax,ds          ;ventana Ascmem
        mov   bx,offset vam
        mov   si,offset mem1
        vnt   'a'
        retn

mstk:   push  es
        mov   cx,4           ;preparo buffer STCK
        mov   si,14          ;segunda columna.
        mov   bx,offset stck
        mov   di,0
        mov   ebp,regsp
        mov   es,regss
pstk1:  mov   ax,es:[bp+di]
        xchg  al,ah
        mov   [bx+si],ax
        sub   si,4
        add   di,2
        loop  pstk1
        mov   cx,5           ;primera columna
        mov   si,16
pstk2:  mov   ax,es:[bp+di]
        xchg  al,ah
        mov   [bx+si],ax
        sub   si,4
        add   di,2
        loop  pstk2
        pop   es
        mov   ax,ds          ;Ventana Stack
        mov   bx,offset vstk
        mov   si,offset stck
        vnt   'h'
        retn
mexpc:  mov   bx,regcs       ;pongo en EXPCOD los valores de Cs e Ip
        mov   di,offset expcod
        push  bx             ;regcs a la pila
        push  bx
        mov   al,bh
        hexasc
        mov   ax,bx
        xchg  al,ah

        push  es
```

```
        push    ds
        pop     es
        stosw
        pop     es
        pop     bx
        mov     al,bl
        hexasc
        mov     ax,bx
        xchg    al,ah

        push    es
        push    ds
        pop     es
        stosw
        pop     es
        mov     ebx,regip
        push    bx              ;regip a la pila
        push    bx
        mov     al,bh
        hexasc
        mov     ax,bx
        xchg    al,ah
        push    es
        push    ds
        pop     es
        stosw
        pop     es
        pop     bx
        mov     al,bl
        hexasc
        mov     ax,bx
        xchg    al,ah
        push    es
        push    ds
        pop     es
        stosw                   ;valores de Cs e Ip almacenados
        pop     es
        mov     count,14
        mov     bx,offset codigo
        mov     si,offset codhex
        add     bx,4
        call    cod00
cod2:   mov     edx,regip
        cmp     ax,dx
        je      cod0
        add     bx,58 ;60
        call    cod00
```

```
        dec     count
        jz      coderr
        jmp     cod2
cod0:   add     bx,58 ;60
        mov     si,offset codhex2
        call    cod00
        sub     ax,codhex
        cmp     ax,14
        ja      coderr
        jmp     cod1
coderr: mov     ax,14
cod1:   mov     cx,ax
        mov     ax,@data        ;paso 14 bytes de codigo a EXPCOD+4
        mov     es,ax             ;es=@data
        mov     di,offset expcod+8
        pop     si              ;si=regip
        pop     ds              ;ds=regcs
        push    cx
cod5:   movsb                   ;ds:si --> es:di
        dec     di
        mov     bl,byte ptr es:[di]
        mov     al,bl
        hexasc
        mov     ax,bx
        xchg    al,ah
        push    ds
        push    es
        pop     ds
        stosw
        pop     ds
        dec     cx
        jnz     cod5
        pop     ax
        mov     cx,28
        sub     cx,ax
        mov     al,' '
        push    ds
        push    es
        pop     ds
cod51:  stosb
        loop    cod51           ;lleno con blancos lo restante
        pop     ds
        mov     ax,@data          ;(cs:ip --> @data:offset expcod+4)
        mov     ds,ax
        mov     es,videoseg
        mov     bx,offset vexpc     ;presento ventana
        mov     ax,ds
```

```
        mov   si,offset expcod
        vnt   'a'
        retn
cod00:  mov   al,[bx]       ;conversion del ascii a hexadecimal
        call  rgd03
        mov   cl,4
        shl   dh,cl         ;dh contiene ubicacion en gdtab del ascii

        mov   [si+1],dh
        mov   al,[bx+1]
        call  rgd03
        add   [si+1],dh     ;MSB transformado
        mov   al,[bx+2]     ;convierto LSB
        call  rgd03
        mov   cl,4
        shl   dh,cl
        mov   [si],dh
        mov   al,[bx+3]
        call  rgd03
        add   [si],dh       ;[si] = ascii convertido en hexadecimal
        mov   ax,[si]
        retn

preopc: mov   cx,104        ;preparo buffer MEMO1
        mov   di,offset memo1  ;en base a OPCSEG, OPCOFF,
        mov   al,' '        ;XXXXSEG y XXXXOFF.
        push  es
        push  ds
        pop   es
        rep   stosb         ;primero borro MEMO2
POPC2:  mov   di,offset memo1 ;escribo MEMO1
        mov   ebx,XXXXSEG   ;segmento
        mov   edx,XXXXOFF   ;offset
        mov   cx,13
        mov   bp,bx         ;guardo BX.
popc1J: mov   bx,bp         ;escribo direcciones
        mov   al,bh         ;de la primera ventana.
        hexasc
        mov   ax,bx
        xchg  al,ah
        stosw
        mov   bx,bp
        mov   al,bl
        hexasc
        mov   ax,bx
        xchg  al,ah
        stosw
```

```
        mov     al,dh
        hexasc
        mov     ax,bx
        xchg    al,ah
        stosw
        mov     al,dl
        hexasc
        mov     ax,bx
        xchg    al,ah
        stosw
        add     dx,8
        loop    popc1J
        pop     es
      retn

rdp:                    ;Corre un paso del programa
        cmp             error,0
        jne             finrdp
        cmp             siarch,0  ;hay archivo para depurar?
        jne     pasar    ;si, proceder
        mov             bx,offset vnoarchivo
        vnt     'a'     ;ventana de error de archivo no presente
        mov             ah,0
        int     16h
        call    pdebug
        jmp             salida  ;no, terminar
pasar:
        usrpro
        mov     word ptr xferaddr,offset YaEj
        mov     word ptr xferaddr+2,cs
        push    toggsub
        push    si
        push    ds                  ;nointerupcion
        push    ax
        mov     Esi,RegIp
        mov     ds,RegCs        ;DS:SI = apuntador a la instrucción
        mov     al,[si]
        cmp     al,0cdh         ;es una INT?
        pop     ax
        pop     ds
        pop     si
        jne     NoInterup
        push    ds
        push    si
        mov     esi,regip
        mov     ds,regcs
        mov     al,[si+1]
```

```
        pop     si
        pop     ds
        push    eax
        cmp     al,10h
        jne     int21?
        mov     eax,regax
        cmp     ah,00
        jne     int21?
        usrpro
        mov     bx,offset video
        vnt     'a'
        mov     ah,00
        int     16h
        usrpro
        pop     eax
        pop     toggsub
        jmp     yaEj                ;finalizar
int21?: pop     eax
      mov       toggsub,1
NoInterup:
        cmp     toggsub,0       ;ingreso a subrutinas,reps,loops?
        pop     toggsub
        je      rdp2
        jmp     expaso          ;no
rdp2:   jmp     exuniq          ;si
YaEj:   mov     ax,@data
        mov     ds,ax
        mov     es,videoseg
         usrpro
finrdp:
        mov             visible,0
        mmuestra 1,visible,ratonpresente
        call     erroric
        jmp     salida
erroric:
        cmp     error,1
        jne     nointe
        mov     bx,offset vinst
        jmp     short mostrare
nointe:
        cmp     error,2
        jne     noerrorc
        mov     bx,offset vcop
mostrare:
        vnt     'a'
        getchar
        getchar
```

```
noerrorc:
        call    pdebug
        retn

modvid:
        mov     ah,0fh
        int     10h
        cmp     al,videomod
        je      bienvid
        mov     al,videomod
        mov     ah,0
        int     10h
        mov     cl,24
        mov     dl,0ah
i_1z:
        pusha
        mov     ah,2            ;Uso rutina del DOS para escribir DL.
        int     21h
      popa
        dec     cl              ;cursor al fin de pantalla
        jnz     i_1z
        mov     ah,02
        mov     dx,1200h
        int     10h
bienvid:
        retn

rdc:
        mov     errorcj,0ffh
        cmp     error,0         ;está activado el indicador de error?
        jne     rdcf
        cmp     siarch,0  ;hay archivo para depurar?
        jne     correr    ;si
        mov     bx,offset vnoarchivo
        vnt     'a'     ;ventana de error de archivo no presente
        mov     ah,0
        int     16h
        call    pdebug
        mov     errorcj,0
        jmp     salida   ;no. terminar
correr:                 ;Corre hasta la ocurrencia de un evento
        limpln  23,revers
        limpln  24,normal
        mov     word ptr xferaddr,offset YaEjHas
        mov     word ptr xferaddr+2,cs
        cmp     toggcorre,0     ;Hasta direccion?
        je      rdcd
```

```
        jmp     rdcc
rdcd:   mov     ax,RegCs        ;Corrida hasta cierto punto
        mov     word ptr dhex1+6,ax
        xchg    al,ah           ;Hago que CS usuario sea seg. predef.
        push    ax              ;para lo que simulo condiciones
        hexasc
        pop     ax
        xchg    bl,bh
        mov     word ptr dhex1+2,bx
        mov     bufcorr,bx
        mov     al,ah
        hexasc
        xchg    bl,bh
        mov     word ptr dhex1+4,bx
        mov     bufcorr+2,bx
        mov     byte ptr dhex1+1,4
rdcd1:  mov     bx,offset vhas  ;ventana para recibir datos (CORRE HASTA
        mov     ax,ds           ;CIERTA DIRECCION)
        mov     si,offset bufcorr
        vnt     'a'
        print   1,23,123cd,revers
        mov     gdreset,1
        cmp     byte ptr bufcorr,'_'
        je      rdcd11
        mov     ax,'__'
        mov     bufcorr,ax
        mov     bufcorr+2,ax
        jmp     rdcd2
rdcd11: call    rdcd0           ;resalto campo de segmento
      getdata   10,20,dhex1,'h' ;recibo segmento
        call    funciones
        mov     al,atrprev      ;quito resaltado
        mov     di,dirprev
        call    rdcd01
        mov     gdreset,0
        cmp     gccod,0         ;dato valido?
        je      rdcd2
        cmp     gccod,7         ;ESC para abortar
        jne     rdcd12
        call    funciones
        jmp     rdcf
rdcd12: cmp     gccod,5         ;TAB para pasar
        jne     rdcd11
rdcd2:  mov     gdreset,1
rdcd21: call    rdcd00          ;resalto campo de offset
      getdata   17,20,dhex2,'h' ;recibo offset
        call    funciones
```

```
        mov   al,atrprev    ;quito resaltado
        mov   di,dirprev
        call  rdcd01
        mov   gdreset,0
        cmp   gccod,0       ;dato valido?
        je    rdcd3
        cmp   gccod,7       ;ESC para abortar
        jne   rdcd22
        call  funciones
        jmp   rdcf
rdcd22: cmp   gccod,5       ;TAB para pasar
        jne   rdcd21
        jmp   rdcd1
rdcd3:  usrpro              ;Pantalla de usuario<-->PRO
        mov   bx,word ptr dhex2+6
        mov   word ptr pointerlst,bx
        mov   ax,word ptr dhex1+6
        mov   word ptr pointerlst+2,ax
        jmp   exhasta
rdcc:   mov   bx,offset vcond ;ventana con las condiciones
        vnt   'a'
        print 1,23,l23cc,revers
        mov   nlinea,0      ;para cada linea...
rdcc1:  call  rdcc0         ;resalto linea actual
        getchar
        mov   al,atrprev    ;quito el resaltado
        mov   di,dirprev
        call  rdcc01
        cmp   gccod,1       ;ENTER?
        je    rdcc2
        cmp   gccod,4       ;ESC?
        jne   rdcc12
        call  funciones
        jmp   rdcf          ;abortar
rdcc12: cmp   gccod,5       ;especial?
        jne   rdcc1
        cmp   gcascii,72    ;si, arriba?
        jne   rdcc13
        dec   nlinea        ;si
        cmp   nlinea,0ffh
        jne   rdcc1
        mov   nlinea,2
        jmp   rdcc1
rdcc13: cmp   gcascii,80    ;abajo?
        jne   rdcc1
        inc   nlinea        ;si
        cmp   nlinea,3
```

```
        jne     rdcc1
        mov     nlinea,0
        jmp     rdcc1
rdcc2:  call    mexpc           ;restauro pantalla
        call    mstk
        cmp     nlinea,2        ;condicion de bandera?
        jne     rdcc21
        jmp     rdcc5
rdcc21: mov     bx,offset vrv   ;ventana para condicion
        vnt     'a'
        limpln  23,revers       ;registro = o <> xxxx
        print   1,23,l23rcmp,revers
        mov     igual+4,'='
        cmp     nlinea,0        ;igualdad?
        je      rdcc22
        mov     igual+4,0d8h    ;no, desigualdad
rdcc22: print   9,20,igual,normal
        mov     gdreset,1
rdcc221:getline 9,20,nomreg
        mov     gdreset,0
        cmp     gccod,4         ;ESC?
        jne     rdcc23
        call    funciones
        jmp     rdcf
rdcc23: cmp     gccod,1         ;recibi un reg?
        jne     rdcc221
        mov     si,offset nomreg+2 ;si recibi, es reg. normal?
        mov     cl,0            ;en CL enviare una bandera
        call    EncReg
        jnc     rdcc25
        mov     cl,1
        call    EncSreg         ;no, es de segmento?
        mov     cl,1
        jnc     rdcc25
        limpln  23,revers       ;no, error
        print   1,23,noesreg,revers
        getchar
        mov     ax,getword
        cmp     al,27           ;ESC?
        je      rdcc24
        jmp     rdcc21
rdcc24: jmp     rdcf
rdcc25: ;aqui ya tengo el registro, ahora recibo el valor
        push    cx
        mov     aux,ax
        limpln  23,revers
        print   1,23,l23cmpr,revers
```

```
        mov     gdreset,1
rdcc26: call    rdcc260         ;resalto el campo
        pop     cx
        cmp     cl,0            ;registro normal?
        jne     rdcc261
        test    byte ptr aux,18h  ;si, 8 o 16 bits?
        jnz     rdcc261
        getdata 17,20,dbyte,'h' ;8 bits
        jmp     rdcc262
rdcc261:
        test    byte ptr aux,16
        jnz     rup1j
      getdata   15,20,dhex1,'h'       ;16 bits
        jmp     rdcc262
rup1j:  mov     cual2,1
        getdata 15,20,dhex1e,'h'        ;32 bits
rdcc262:mov     gdreset,0
        cmp     gccod,7         ;ESC?
        jne     rdcc27
        jmp     rdcf            ;si,salgo
rdcc27: cmp     gccod,0         ;tengo dato?
        jne     rdcc26
        mov     ah,0            ;si, preparo llamada a ExCond
        cmp     cl,0            ;registro normal?
        je      rdcc28
      mov       ah,80h          ;no, de segm., bit 4 de AH = 1
rdcc28: mov     al,byte ptr aux
        add     ah,al
        cmp     cl,0            ;reg. normal?
        jne     rdcc281
        test    byte ptr aux,18h  ;si, de 8 o 16 bits?
        jnz     rdcc281
        mov     bl,dbyte+4      ;8 bits
        mov     bh,0
        jmp     rdcc282
rdcc281:
        test    byte ptr aux,16
        jnz     rdcc281j
        mov     bx,word ptr dhex1+6     ;16 bits
        jmp     rdcc282
rdcc281j:
        mov     ebx,dword ptr dhex1E+10 ;32 bits
rdcc282:
        pushad
        usrpro                  ;Pantalla de usuario<-->PRO
        popad
        cmp     nlinea,0        ;igualdad?
```

```
        je      rdcc3
        jmp     rdcc4
rdcc3:  mov     al,0            ;registro = xxxx
        jmp     excond
rdcc4:  mov     al,1            ;registro <> xxxx
        jmp     excond
rdcc5:  mov     bx,offset vcban ;condicion de banderas
        vnt     'a'
        mov     nlinea,0        ;para cada linea...
rdcc51: call    rdcc50          ;resalto opcion actual
        getchar
        mov     al,atrprev      ;quito resaltado
        mov     di,dirprev
        call    rdcc501
        cmp     gccod,1         ;ENTER?
        je      rdcc6
        cmp     gccod,4         ;ESC?
        jne     rdcc53
        call    funciones
        jmp     rdcf
rdcc53: cmp     gccod,5         ;especial?
        jne     rdcc5
        cmp     gcascii,72      ;arriba?
        jne     rdcc54
        dec     nlinea          ;si
        cmp     nlinea,0ffh
        jne     rdcc51
        mov     nlinea,15
        jmp     rdcc51
rdcc54: cmp     gcascii,80      ;abajo?
        jne     rdcc5
        inc     nlinea
        cmp     nlinea,16
        jne     rdcc51
        mov     nlinea,0
        jmp     rdcc51
rdcc6:
        usrpro                  ;Pantalla de usuario<-->PRO
        mov     al,2            ;ya tengo la condicion
        mov     ah,nlinea
        jmp     excond
YaEjHas:mov     ax,@data
        mov     ds,ax
        mov     es,videoseg
        usrpro                  ;P. de usuario<-->PRO
rdcf:
        call    pdebug
```

```
        call    modvid
        call    funciones
        mov     si,offset l0ini ;preparo y muestro linea 0
        call    mlin0
        call    errorlc
        mreseteo varraton1,visible,ratonpresente
        mov     visible,0
        mmuestra 1,visible,ratonpresente
        mov     errorcj,0
        jmp     salida
rdcd0:  posdir  10,20           ;resalto campo de segmento
rdcd0a: mov     di,pddir
        inc     di              ;calculo posicion
        mov     ah,es:[di]      ;atributo en ese lugar
        mov     atrprev,ah
        mov     dirprev,di
        mov     al,70h          ;reverso
rdcd01: mov     cx,4
rdcd02: stosb
        inc     di
        loop    rdcd02
        retn
rdcd00: posdir  17,20           ;resalto campo de offset
        jmp     rdcd0a          ;aqui incluido el RETN
rdcc0:  mov     cl,7            ;resalto uno de los tres casos
        mov     ch,nlinea       ;similar a rdcd0
        add     ch,18
        posdir  cl,ch
        mov     di,pddir
        inc     di
        mov     ah,es:[di]
        mov     atrprev,ah
        mov     dirprev,di
        mov     al,70h
rdcc01: mov     cx,17
        jmp     rdcd02          ;aqui ya se hace el RETN
rdcc50: mov     cl,19           ;resalto una de las 16 opciones
        mov     ch,nlinea
        add     ch,6
        posdir  cl,ch
        mov     di,pddir
        inc     di
        mov     ah,es:[di]
        mov     atrprev,ah
        mov     dirprev,di
        mov     al,70h
rdcc501:mov     cx,15
```

```
        jmp     rdcd02          ;aqui ya se hace el RETN
rdcc260:posdir  15,20           ;resalto campo de direccion
        mov     di,pddir
        inc     di
        mov     al,70h
        mov     cx,8
rdcd02j:stosb
        inc     di
        loop    rdcd02j
        retn

rdj:                    ;ejecuta hasta ruptura o final
        mov     errorcj,0ffh
        cmp     error,0
        jne     finejec
        cmp     siarch,0  ;hay archivo para depurar?
        jne     ejecutar  ;si
        mov     bx,offset vnoarchivo
        vnt     'a'       ;ventana de error de archivo no presente
        mov     ah,0
        int     16h
        call    pdebug
        mov     errorcj,0
        jmp     salida
ejecutar:
        usrpro
        mov     ax,offset rdj1
        mov     word ptr XferAddr,ax
        mov     word ptr XferAddr+2,cs
        jmp     ExTotal
rdj1:   mov     ax,@data
        mov     ds,ax
        mov     es,videoseg
        call    modvid
        usrpro
finejec:
        mov     si,offset l0ini ;preparo y muestro linea 0
        call    mlin0
        call    funciones
        call    erroric
        mreseteo varraton1,visible,ratonpresente
        mov     visible,0
        mmuestra 1,visible,ratonpresente
        mov     errorcj,0
        jmp     salida
rdra:                   ;activo/desactivo bkpt
        mov     brup,1          ;bandera BRUP=1 para activar/desactivar.
```

```
        jmp     rdrb1
rdrc:   mov     brup,2          ;crear brkpt, BRUP=2
        jmp     rdrb1
rdrb:   mov     brup,3          ;borro brkpt, BRUP=3
rdrb1:  mov     rupact,0        ;recibo nombre del Brkpt
        limpln  23,revers       ;RUPACT=ruptura actual-1
        limpln  24,normal
        print   1,23,l23bkp,revers
        mov     bx,offset vnom ;ventana para nombre
        mov     ax,ds
        mov     si,offset uline
        vnt     'a'
        mov     gdreset,1
rdb11a: call    rdr0            ;muestra ventana de bkpts
rdb11:  getline 12,19,nombrk    ;recibo nombre del brkpt
        mov     gdreset,0
        cmp     gccod,4         ;ESC?
        jne     rdb12
        jmp     rdrbf2
rdb12:  cmp     gccod,1         ;vale?
        jne     rdb13
        cmp     byte ptr nombrk[1],0 ;si, pero esta vacio?
        jne     rdb14
        jmp     rdrbf2          ;si,termino
rdb13:  cmp     gccod,5         ;especial?
        jne     rdb11
        cmp     gcascii,73      ;PGUP?
        je      rdb132
        cmp     gcascii,81      ;PGDN?
        jne     rdb11           ;no vale
        cmp     rupact,247      ;PGDN
        ja      rdb11
        add     rupact,8
        mov     al,16
        mov     cl,rupact
        mul     cl
        add     ax,offset TablaBrk
        mov     si,ax
        cmp     byte ptr [si],0
        jne     rdb11a
        sub     rupact,8
        jmp     rdb11
rdb132: cmp     rupact,8        ;PGUP
        jb      rdb11
        sub     rupact,8
        jmp     rdb11a
rdb14:  mov     cl,1            ;acceso  a bkpts solo por nombre
```

```
        mov     si,offset nombrk+2
        mov     al,brup
rdrb2:  cmp     al,1            ;activo/desactivo brkpt?
        jne     rdrb3
        call    TogBrkPt        ;si, toggle
        jc      rdrb21
        call    rdr0
        call    pdebug
        call    rdr0
        jmp     rdra            ;rdrbf1
rdrb21: limpln  23,revers
        print   1,23,noesbrk,revers
        getchar
        call    rdr0
        call    pdebug
        call    rdr0
        jmp     rdra            ;rdrbf
rdrb3:  cmp     al,2            ;creo brkpt?
        je      rdrb31
        jmp     rdrb4
rdrb31: limpln  23,revers       ;limpio lineas 23 y 24
        mov     ax,RegCs        ;bkpt en cierto punto
        mov     word ptr dhex1+6,ax
        xchg    al,ah           ;Hago que CS usuario sea seg. predef.
        push    ax              ;para lo que simulo condiciones
        hexasc
        pop     ax
        xchg    bl,bh
        mov     word ptr dhex1+2,bx
        mov     bufcorr,bx
        mov     al,ah
        hexasc
        xchg    bl,bh
        mov     word ptr dhex1+4,bx
        mov     bufcorr+2,bx
        mov     byte ptr dhex1+1,4
rup1:   mov     bx,offset vrdir ;ventana para recibir direccion
        mov     ax,ds
        mov     si,offset bufcorr
        vnt     'a'
        print   1,23,123cd,revers
        mov     gdreset,1
        cmp     byte ptr bufcorr,'_'
        je      rup1.1
        mov     ax,'__'
        mov     bufcorr,ax
        mov     bufcorr+2,ax
```

```
        mov     byte ptr[di+2],'*'
rdr021: mov     ax,ds:[si+bp+2] ;copio 10 bytes del nombre
        mov     [di+3],ax
        mov     ax,ds:[si+bp+4]
        mov     [di+5],ax
        mov     ax,ds:[si+bp+6]
        mov     [di+7],ax
        mov     ax,ds:[si+bp+8]
        mov     [di+9],ax
        mov     ax,ds:[si+bp+10]
        mov     [di+11],ax
        mov     al,ds:[si+bp+15]        ;muestro segmento
        hexasc
        xchg    bl,bh
        mov     [di+13],bx
        mov     al,ds:[si+bp+14]
        hexasc
        xchg    bl,bh
        mov     [di+15],bx
        mov     al,ds:[si+bp+13]        ;muestro offset
        hexasc
        xchg    bl,bh
        mov     [di+17],bx
        mov     al,ds:[si+bp+12]
        hexasc
        xchg    bl,bh
        mov     [di+19],bx
        inc     nlinea          ;veo si termine la ventana
        inc     dl
        add     di,21
        cmp     nlinea,8        ;llego a ultima linea?
        je      rdr03
        cmp     dl,0            ;ultima ruptura?
        je      rdr03
        jmp     rdr01
rdr03:  mov     bx,offset vrup  ;muestro la ventana
        mov     ax,ds
        mov     si,offset bufrup
        vnt     'a'
        retn

rdrm:                   ;muestra ventana de bkpts
        mov     rupact,0
        limpln  23,revers
        limpln  24,normal
        print   1,23,l23bkp,revers
rdrm11a:call    rdr0            ;muestra ventana de bkpts
```

```
rdrm11: getchar              ;recibo orden del teclado
        cmp    gccod,4       ;ESC?
        jne    rdrm12
        call   funciones
        jmp    rdrmf
rdrm12: cmp    gccod,5       ;especial?
        jne    rdrm11
rdrm13: cmp    gcascii,73    ;PGUP?
        je     rdrm132
        cmp    gcascii,81    ;PGDN?
        jne    rdrm11        ;no vale
        cmp    rupact,247    ;PGDN
        ja     rdrm11
        add    rupact,8
        mov    al,16
        mov    cl,rupact
        mul    cl
        add    ax,offset TablaBrk
        mov    si,ax
        cmp    byte ptr [si],0
        jne    rdrm11a
        sub    rupact,8
        jmp    rdrm11
rdrm132:cmp    rupact,8      ;PGUP
        jb     rdrm11
        sub    rupact,8
        jmp    rdrm11a
rdrmf:  call   mmem          ;restauro pantalla
        call   mreg
        call   mpsw
        jmp    salida

rdi:    limpln 23,revers
        limpln 24,normal
        print  1,23,instruc,revers
        mov    gdreset,1
rdi1:   getline 1,24,insbuff ;recibo nemotecnico
        mov    gdreset,0
        cmp    gccod,4
        jne    rdi2
        jmp    rdif          ;ESC
rdi2:   cmp    gccod,1
        jne    rdi1
        mov    bl,insbuff+1  ;ENTER, dato valido
        mov    bh,0
        mov    si,offset insbuff+2
        mov    byte ptr[bx+si],0
```

```
        mov     word ptr XferAddr,offset rdi3   ;direccion de retorno
        mov     word ptr XferAddr+2,cs
        call    funciones
        jmp     ExIns           ;ejecuto
rdi3:   mov     ax,@data
        mov     ds,ax
        mov     es,videoseg
        jnc     rdi4
        limpln  23,revers       ;nemotecnico incorrecto
        print   1,23,errens,revers
        jmp     rdif
rdi4:   call    pdebug          ;todo bien
        limpln  23,revers
        print   1,23,listo,revers
rdif:   getchar
        call    funciones
        jmp     salida


rdeb:                   ;ensambla y busca en el segmento de codigo
        limpln  0,revers
        limpln  1,revers
        print   10,0,ensw1,revers
        print   10,1,ensw2,revers
        limpln  23,revers
        limpln  24,normal
        print   1,23,ensbusc,revers ;indicaciones
        mov     gdreset,1
rdeba:  limpln  24,normal
rdeba1: getline 1,24,insbuff    ;recibo nemotecnico
        mov     gdreset,0
        cmp     gccod,4         ;ESC?
        jne     rdeba2
        call    funciones
        jmp     rdebf
rdeba2: cmp     gccod,1         ;Vale?
        jne     rdeba1
        mov     bl,insbuff[1]   ;BX = # de caracteres
        cmp     bl,0
        jne     rdeba3
        jmp     rdebf
rdeba3: xor     bh,bh
        mov     si,offset insbuff+2
        mov     byte ptr [si+bx],0 ;termino buffer con 0
        mov     bx,es
        mov     ax,ds
        mov     es,ax
```

```
        mov     di,offset InsEns
        call    assem
        mov     es,bx
        jnc     rdebx
        limpln  23,revers
        print   1,23,errens,revers
        getchar
        jmp     rdebf
rdebx:  push    RegIp
        mov     bx,cx           ;busco en el codigo
rdebx1: mov     es,RegCs        ;Inicio de busqueda
        mov     Edi,RegIp       ;en CS:IP del usuario.
        mov     si,offset InsEns
        mov     cx,0ffffh
        call    buscar
        cmp     cx,0            ;CX=0 si no encontro
        jne     rdeb1
        pop     RegIp           ;no encontro
        mov     es,videoseg
        limpln  23,revers
        print   1,23,NoEnsB,revers
        getchar
        jmp     rdebf
rdeb1:  mov     es,videoseg
        limpln  23,revers       ;si encontro,
        mov     offcod,di
        push    bx
        push    di
        call    mcod
        pop     di
        pop     bx
        print   1,23,sigo?,revers ;busco proxima?
rdeb11: getchar
        mov     ax,getword
        cmp     al,1bh
        jne     rdeb2
        jmp     rdeb3           ;ESC
rdeb2:  cmp     al,0dh
        jne     rdeb11
        inc     di              ;ENTER, sigo buscando
        mov     RegIp,Edi
        jmp     rdebx1
rdeb3:  pop     RegIp
rdebf:  mov     si,offset l0ini ;preparo y muestro linea 0
        call    mlin0
        limpln  23,normal
        print   0,23,l1ini,bmenu
```

```
rinif5: call   pdebug        ;actualizo pantalla
        call   funciones
        jmp    salida

rdml:                        ;lleno un area de memoria
        limpln 23,revers
        limpln 24,normal
        mov    bx,offset vll ;ventana para pedir datos
        mov    eax,autooff
        xchg   al,ah
        mov    bufm1+2,ax
        mov    eax,autoseg
        xchg   al,ah
        mov    bufm1,ax
        mov    ax,ds
        mov    si,offset bufm1
        vnt    'h'
        print  1,23,bytes?,revers
        mov    gdreset,1     ;recibo # de bytes
rdml1:  call   rdml0         ;resalto campo de #bytes
        getdata 20,18,dhex1,'h'
        mov    gdreset,0
        mov    al,atrprev
        mov    di,dirprev
        call   rdml01
        cmp    gccod,7
        jne    rdml2
        jmp    rdmlf         ;ESC
rdml2:  cmp    gccod,0
        jne    rdml1
        limpln 23,revers
        print  1,23,valor?,revers
        mov    gdreset,1     ;aqui ya hay el #bytes en DHEX1+6
rdml3:  call   rdml00
        getdata 22,19,dbyte,'h'
        mov    gdreset,0
        mov    al,atrprev
        mov    di,dirprev
        call   rdml001
        cmp    gccod,7
        je     rdmlf         ;ESC
        cmp    gccod,0
        jne    rdml3
rdml4:  push   es            ;aqui ya tengo el valor
        mov    es,autoseg
        mov    edi,autooff   ;dir1     ;ES:DI apunta al inicio del area
        mov    cx,word ptr dhex1+6
```

```
        mov     al,dbyte+4
        call    llenar          ;lleno el area con el valor de AL
        pop     es
        limpln  23,revers
        print   1,23,listo,revers
        call    pdebug
        getchar
        jmp     rdmlf1
rdmlf:  call    pdebug
rdmlf1: call    funciones
        jmp     salida
rdml0:  posdir  20,18           ;resalto campo de #bytes
        mov     di,pddir
        inc     di
        mov     al,es:[di]
        mov     atrprev,al
        mov     dirprev,di
        mov     al,70h
rdml01: mov     cx,4
rdml02: stosb
        inc     di
        loop    rdml02
        retn
rdml00: posdir  22,19
        mov     di,pddir
        inc     di
        mov     al,es:[di]
        mov     atrprev,al
        mov     dirprev,di
        mov     al,70h
rdml001:mov     cx,2
        jmp     rdml02          ;esto incluye el retn
rdmb:                   ;busca un string en segmento actual

        limpln  23,revers
        limpln  24,normal
        print   1,23,buscasc?,revers
rdmb1:  getchar                 ;espero respuesta H, A o ESC
        mov     ax,getword
        cmp     al,1bh
        jne     rdmb12
        jmp     rdmbf2          ;ESC
rdmb12: cmp     al,'A'
        je      rdmb2
        cmp     al,'a'
        je      rdmb2
        cmp     al,'H'
```

```
        je      rdmb13
        cmp     al,'h'
        jne     rdmb1           ;respuesta incorrecta
rdmb13: jmp     rdmb4
rdmb2:  mov     gdreset,1       ;Secuencia en ASCII
        limpln  23,revers
        print   1,23,busca?,revers
rdmb21: getline 1,24,insbuff    ;leo secuencia
        mov     gdreset,0
        cmp     gccod,4
        jne     rdmb22
        jmp     rdmbf2          ;ESC
rdmb22: cmp     gccod,1
        jne     rdmb21
        mov     bl,insbuff+1    ;vale, pero si esta vacio salgo
        cmp     bl,0
        jne     rdmb23
        jmp     rdmbf2
rdmb23: mov     bh,0            ;preparo BX para llamar a rutina de busqueda
        jmp     rdmb6
rdmb4:  limpln  23,revers       ;Secuencia en HEX
        print   1,23,busca?,revers
        mov     bx,0
rdmb41: mov     cl,bl
        mov     al,3
        mul     cl
        inc     al
        mov     cl,al
        mov     gdreset,1
rdmb42: call    rdmb0           ;resalto sitio actual
        getdata cl,24,dbyte,'h'
        mov     gdreset,0
        mov     al,atrprev      ;quito resaltado
        mov     di,dirprev
        call    rdmb01
        cmp     gccod,7
        jne     rdmb43
        jmp     rdmbf2          ;ESC
rdmb43: cmp     gccod,0
        jne     rdmb42
        cmp     dbyte+1,0       ;0 datos?
        je      rdmb5           ;si, pasar a buscar
        mov     si,offset insbuff+2 ;vale,guardo en INSBUFF+2
        mov     al,dbyte+4
        mov     [si+bx],al
        inc     bl
        cmp     bl,24
```

```
        jne     rdmb41
rdmb5:  cmp     bl,0
        jne     rdmb51
        jmp     rdmbf2
rdmb51: mov     insbuff+1,bl
rdmb6:  mov     cx,0ffffh       ;preparo llamada
        mov     bl,insbuff+1
        mov     bh,0
        mov     si,offset insbuff+2
        push    eax
        mov     eax,autoseg
        mov     es,ax
        pop     eax
        mov     edi,autooff
        call    buscar
        push    eax
        mov     ax,es
        mov     autoseg,eax
        mov     dhexseg,eax
        mov     xxxxseg,eax
        pop     eax
        mov     autooff,edi
        mov     dhexoff,edi
        mov     xxxxoff,edi
        mov     es,videoseg
        cmp     cx,0            ;CX=0 si no encontro
        jne     rdmb7
        limpln  23,revers       ;no encontro
        print   1,23,NoHayStr,revers
        inc     autooff
        inc     dhexoff
        inc     xxxxoff
        call    preopc
        call    mmem
        jmp     rdmbf
rdmb7:  call    preopc  ;encontro, muestro en ventana
        call    mmem
        limpln  23,revers
        print   1,23,sigo?,revers ;busco proxima?
rdmb72: getchar
        mov     ax,getword
        cmp     al,1bh
        je      rdmbf2          ;ESC, salgo
        cmp     al,0dh
        jne     rdmb72
        inc     autooff
        inc     dhexoff
```

```
        inc     xxxxoff
        jmp     rdmb6           ;enter, busco otra vez
rdmbf:  getchar
rdmbf2: call    funciones
        jmp     salida
rdmb0:  posdir  cl,24           ;resalto campo para byte actual
        mov     di,pddir
        inc     di
        mov     al,es:[di]
        mov     atrprev,al
        mov     dirprev,di
        mov     al,70h
rdmb01: stosb
        inc     di
        stosb
        retn
rdmc:   ;copia desde la ventana principal a otra area
        limpln  23,revers       ;limpio lineas 23 y 24
        limpln  24,normal
        mov     eax,autooff
        xchg    al,ah
        mov     bufm1+2,ax
        mov     eax,autoseg
        xchg    al,ah
        mov     bufm1,ax
        mov     eax,autoseg
        mov     word ptr dhex1+6,ax
        xchg    al,ah           ;Hago que DS usuario sea seg. predef.
        push    ax              ;para lo que simulo condiciones
        mov     bufm1+4,ax
        hexasc
        pop     ax
        xchg    bl,bh
        mov     word ptr dhex1+2,bx
        mov     bufcorr,bx
        mov     al,ah
        hexasc
        xchg    bl,bh
        mov     word ptr dhex1+4,bx
        mov     bufcorr+2,bx
        mov     byte ptr dhex1+1,4
rdmc1:  mov     bx,offset vcopia ;ventana para recibir direccion
        mov     ax,ds
        mov     si,offset bufm1
        vnt     'h'
        print   1,23,123cd,revers
        mov     gdreset,1
```

```
        cmp     byte ptr bufcorr,'_'
        je      rdmc10
        mov     ax,'__'
        mov     bufcorr,ax
        mov     bufcorr+2,ax
        mov     ax,0
        mov     bufm1+4,ax
        jmp     rdmc2
rdmc10: print   15,18,uline1,revers
rdmc11: call    rdmc0           ;resalto campo de segmento
        getdata 15,18,dhex1,'h' ;recibo segmento
        mov     al,atrprev      ;quito resaltado
        mov     di,dirprev
        call    rdmc01
        mov     gdreset,0
        cmp     gccod,0         ;dato valido?
        je      rdmc2
        cmp     gccod,7         ;ESC para abortar
        jne     rdmc12
        jmp     rdmcf
rdmc12: cmp     gccod,5         ;TAB para pasar
        jne     rdmc11
rdmc2:  mov     gdreset,1
rdmc21: call    rdmc00          ;resalto campo de offset
        getdata 20,18,dhex2,'h' ;recibo offset
        mov     al,atrprev      ;quito resaltado
        mov     di,dirprev
        call    rdmc01
        mov     gdreset,0
        cmp     gccod,0         ;dato valido?
        je      rdmc3
        cmp     gccod,7         ;ESC para abortar
        jne     rdmc22
        jmp     rdmcf
rdmc22: cmp     gccod,5         ;TAB para pasar
        jne     rdmc21
        jmp     rdmc1
rdmc3:  print   1,23,bytes?,revers ;recibo el # de bytes
        mov     bp,word ptr dhex1+6 ;libero dhex1
        mov     gdreset,1
        call    rdmc30          ;resalto campo de #bytes
rdmc31: getdata 20,19,dhex1,'h'
        mov     gdreset,0
        cmp     gccod,7
        je      rdmcf           ;ESC
        cmp     gccod,0
        jne     rdmc31          ;no vale
```

```
rdmc4:  mov    es,bp          ;aqui tengo todo listo, copio
        mov    di,word ptr dhex2+6 ;ES:DI-->destino
        mov    cx,word ptr dhex1+6 ;#bytes
        mov    esi,autooff
        mov    ds,autoseg     ;dir1+2      ;DS:SI-->origen
        call   copiar
        mov    ax,@data
        mov    ds,ax
        mov    es,videoseg
        limpln 23,revers
        print  1,23,listo,revers
        call   pdebug
        getchar
        jmp    rdmcf1
rdmcf:  call   pdebug
rdmcf1: call   funciones
        jmp    salida

rdmc0:  posdir 15,18          ;resalto campo de segmento
        mov    di,pddir
        inc    di
        mov    al,es:[di]
        mov    atrprev,al
        mov    dirprev,di
        mov    al,70h
rdmc01: jmp    rdml01         ;Esto incluye el RETN
rdmc00: posdir 20,18          ;resalto campo de offset
        mov    di,pddir
        inc    di
        mov    al,es:[di]
        mov    atrprev,al
        mov    dirprev,di
        mov    al,70h
        jmp    rdml01         ;Esto incluye el RETN
rdmc30: posdir 20,19          ;resalto campo de #bytes
        mov    di,pddir
        inc    di
        mov    al,es:[di]
        mov    atrprev,al
        mov    dirprev,di
        mov    al,70h
        jmp    rdml01         ;Esto incluye el RETN

rdfun:  cmp    otroaux,0      ;si se digito ENTER, mostrar funciones
        jne    rdfun1
        jmp    rdfm
rdfun1: cmp    mopaux,0       ;ASCII normal si AL=0
```

```
        je      rdf11
        jmp     rdf6
rdf11:  cmp     mopaux1,59      ;F1?
        jne     rdf12
cop451: call    rd_             ;editor en pantalla
        mov     f3?,0
        jmp     rdff
rdf12:  cmp     mopaux1,60      ;F2?
        jne     rdf13
        jmp     rdff
rdf13:  cmp     mopaux1,61      ;F3?
        jne     rdf14
        xor     f3?,1           ;complemento variable
        cmp     f3?,1
        je      cop42
        jmp     cop43
cop42:
        call    rdf3m
        mmuestra 0,visible,ratonpresente
        mov     ax,ds
        mov     bx,offset vcop0
        vnt     'a'

        mov     ax,ds
        mov     bx,offset vcop1         ;presento datos pila coprocesador
        mov     si,offset memocop1
        vnt     'a'
        mov     ax,ds
        mov     bx,offset vcop2         ;presento stat. y cont. del coproc.
        mov     si,offset memocop2
        vnt     'a'
        mmuestra 1,visible,ratonpresente
        stodo
        mov     getword,ax              ;espero se presione una tecla
        rtodo
        mov     ax,getword
        cmp     al,27                   ;esc?
        je      cop40
        cmp     ah,61                   ;f3?
        je      cop40
        cmp     ah,59                   ;f1?
        jne     cop44
        xor     f3?,1
        call    pdebug
        jmp     cop451                  ;transfiero control a rutina de F1
cop44:  cmp     ah,62                   ;f4?
        jne     cop45
```

```
        jmp    cop461          ;transfiero control a rutina de F4
cop45:  cmp    ah,65           ;f7?
        jne    cop46
        jmp    cop471          ;transfiero control a rutina de F7
cop46:  cmp    ah,66           ;f8?
        jne    cop47
        jmp    cop481          ;transfiero control a rutina de F8
cop47:  cmp    ah,67           ;f9?
        jne    cop49
        jmp    rdf19a          ;transfiero control a rutina de F9
cop49:  cmp    ah,68           ;f10?
        je     rdfa            ;transfiero control a rutina de F10
        jmp    rdff
cop40:  xor    f3?,1
cop43:  call   pdebug
        jmp    rdff

rdf14:  cmp    mopaux1,62      ;F4?
        jne    rdf15
cop461: usrpro                 ;Muestro pantalla de usuario
rdu1:   getchar                ;espero ESC para regresar
        mov    ax,getword
      cmp      ah,62           ;ESC?
        je     rdf1j
        cmp    al,27
        jne    rdu1
rdf1j:  usrpro                 ;vuelvo a pantalla de PRO
        jmp    rdff
rdf15:  cmp    mopaux1,63      ;F5?
        jne    rdf16
        xor    f5?,1
        call   pdebug          ;si f5?=1 --> debo presentar ventana con
                               ;instrucciones completas
                               ;si f5?=0, no
NocodE: jmp    rdff
rdf16:  cmp    mopaux1,64      ;F6?
        jne    rdf17
        stodo
        mov    ax,@data
        mov    ds,ax
        push   ax
        push   di
        push   si
        push   cx
        mov    bx,offset vnarch
        vnt    'a'
        mov    gdreset,1
```

```
getline         13H,0eH,Narch2      ;receptar el nombre del archivo a abrir
                 ;en Narch2
mov             cl,byte ptr narch2+1
cmp    cl,0             ;se ingresaron parámetros?
je     finf6            ;no, terminar
push   cx
mov    di,offset narch    ;
mov    al,' '           ;
mov    cx,40
push   es
push   ds
pop    es
rep    stosb
pop    es
pop    cx
mov    si,offset narch2+2
mov    di,offset narch    ;pasar los parámetros a Narch
push   es
push   ds
pop    es
rep    movsb
pop    es
pop    cx
pop    si
pop    di
pop    ax
mov    bx,ambiente          ;preparar para correr el editor
mov    pb.ambiente1,bx
mov    bx,offset narch1
mov    pb.paramaddress,bx
mov    pb.paramsegmento,ax
mov    bx,offset fcbloque1
mov    pb.fcb1offset,bx
mov    pb.fcb1segmento,ax
mov    bx,offset fcbloque2
mov    pb.fcb2offset,bx
mov    pb.fcb2segmento,ax
mov    bx,offset pb
push   ds
mov    es,ax
mov    di,pb.fcb1offset
mov    si,pb.ambiente1
inc    si
mov    ax,2901h
int    21h
pop    es
mov    di,pb.fcb2offset
```

```
        mov     ax,2901h
        int     21h
        push    bp
        mov     viejosp,sp
        mov     viejoss,ss
        mov     ah,62h              ;obtener dirección de mi PSP
        int     21h
        mov     es,bx
        mov     bx,2000h             ;acortar la memoria que utilizo
        mov     ah,4ah              ;liberar memoria
        int     21h
        jnc     simemoria            ;
        call    menserror            ;llamar a presentar mensaje de error
        jmp     biencorrido
simemoria:
        push    ds
        pop     es
        mov     bx,offset pb        ;
        mov     dx,offset nombre
        mov     ax,4b00h            ;ejecutar el editor
        int     21h
        jnc     biencorrido         ;hubo errores?
        call    menserror           ;si
biencorrido:
        mov     ss,viejoss          ;restaurar los punteros de pila
        mov     sp,viejosp
        pop     bp
        mov     visible,0
      mmuestra 1,visible,ratonpresente ;mostrar el puntero del ratón
        mov     ah,2
        mov     bh,0
        mov     dh,24
        mov     dl,0
        int     10h                 ;mover el cursor
        rtodo
        jmp     rd170
finf6:  pop     cx
        pop     si
        pop     di
        pop     ax
        rtodo
rd170:  call    pdebug
        jmp     rdff
rdf17:  cmp     mopaux1,65      ;F7?
        jne     rdf18
cop471: xor     toggcorre,1     ;corre hasta direccion/condicion.
        mov     al,6            ;indico en pantalla:
```

```
        mov   cl,toggcorre   ;Direcc o Condic
        mul   cl
        mov   si,ax
        add   si,offset opccorr      ; 'DireccCondic'
        mov   di,offset l1ini+47
        mov   ax,[si]
        mov   [di],ax
        mov   ax,[si+2]
        mov   [di+2],ax
        mov   ax,[si+4]
        mov   [di+4],ax
        print 0,23,l1ini,bmenu
        jmp   rdff
rdf18:  cmp   mopaux1,66     ;F8?
        jne   rdf19
cop481: xor   toggsub,1      ;permite/impide ingreso en
        mov   al,2           ;subrutinas, reps o loops.
        push  ax
        mov   ax,toggsub     ;indico en pantalla:
        mov   cl,al
        pop   ax
        mul   cl             ;Si o No Subr.
        mov   si,ax
        add   si,offset opcpaso      ; 'SINO'
        mov   di,offset l1ini+55
        mov   ax,[si]
        mov   [di],ax
        print 0,23,l1ini,bmenu
        jmp   rdff
rdf19:  cmp   mopaux1,67     ;F9?
        je    rdf19a
        jmp   rdf1a
rdf19a: mmuestra 0,visible,ratonpresente
        mov   bx,offset vf9
        mov   ax,ds
        mov   si,offset uline2
        vnt   'a'
        mmuestra 1,visible,ratonpresente
        mov   gdreset,1      ;recibo un decimal
rdf191: mov   cual2,1
        getdata 3ah,4,ddec,'d'
        mov   gdreset,0
        cmp   gccod,7
        jne   rdf192
        jmp   rdf19f         ;ESC
rdf192: cmp   gccod,0
        jne   rdf191
```

```
        posdir 47h,4
        mov    di,pddir
        mov    eax,dword ptr ddec+12
        push   ax
        rcr    eax,16
        xchg   al,ah          ;muestro equivalencia hex
        push   ax             ;en la pantalla
        hexasc
        mov    es:[di],bh
        mov    es:[di+2],bl
        pop    ax
        mov    al,ah
        hexasc
        mov    es:[di+4],bh
        mov    es:[di+6],bl
        pop    ax
        xchg   al,ah          ;muestro equivalencia hex
        push   ax             ;en la pantalla
        hexasc
        mov    es:[di+8],bh
        mov    es:[di+10],bl
        pop    ax
        mov    al,ah
        hexasc
        mov    es:[di+12],bh
        mov    es:[di+14],bl
        limpln 24,normal
        getchar
rdf19f: call   mreg
        call   mpsw
        jmp    rdff
rdf1a:  cmp    mopaux1,68     ;F10?
        je     rdfa
        jmp    rdff
rdfa:                         ;si
        limpln 24,normal
        mmuestra 0,visible,ratonpresente
        mov    bx,offset vasc1 ;muestro ventana de codigo ASCII
        mov    ax,ds
        mov    si,offset uno
        vnt    'a'
        mov    bx,offset vasc2
        vnt    'a'
        mmuestra 1,visible,ratonpresente
        getchar
        call   pdebug
        jmp    rdff
```

```
rdf6:  cmp    mopaux,'F'     ;ASCII normales
       je     rdfm           ;si es F o f muestro explico funciones
       cmp    mopaux,'f'
       jne    rdff
rdfm:  mov    bx,offset vfunc ;muestro ventana
       vnt    'a'
       limpln 24,normal
       getchar
       call   pdebug
rdff:  mov    mopaux,0
       mov    mopaux1,0
       jmp    salida

menserror:     ;Muestra mensajes de error que puedan ocurrir al correr
               ;el editor de texto(F6), SHell, Ayuda (F2) desde el menú
               ;principal
       jnc     bieneorrido
       shl     al,1
       mov     si,offset mensajes
       xor     bx,bx
       mov     bl,al
       mov     dx,[si+bx]
       mov     ah,9
       int     21h
       getchar
       retn

rd_:                    ;edicion en pantalla.
;Orden en que se atendera a las ventanas: 1 Registros, 2 Banderas,
;3 Codigo, 4 direccion memoria1, 5 direcciones memoria2, 6 memoria1
;hex, 7 memoria1 ascii, 8 memoria2 hex, 9 memoria2 ascii.
;Se usa para leer datos dos rutinas: GETCHAR y GETDATA, que al salir
;dan codigos sobre la forma en que se produjo la salida. En esta
;parte del programa el usuario tendra opciones generales como:
;      TAB          pasa a proxima ventana.
;      SHIFT TAB    pasa a ventana anterior.
;      ESC          sale del editor al menu.
;Existen opciones aplicables de forma diferente en cada rutina.

rd_1:  ;Cambios en ventana que muestra registros.
       ;Usar Flechas para moverse o para no alterar un registro.
       mov    vent,1
       mov    ventactual,0
       mov    dl,0          ;reg. actual = BP
       mov    cual,1
       mov    gdreset,1
       mov    si,offset l0reg ;LINEA0
```

```
        call    mlin0
        limpln  1,revers        ;lineas 1, 23 y 24
        limpln  23,revers
        limpln  24,normal
        print   15,1,11reg,revers
        print   3,23,123d_,revers
        print   3,24,124d_,normal
rd_a:
        push    f5?
        mov     f5?,0
        call    code?
        pop     f5?
        mov     bx,offset vr    ;ventana para cambios
        mov     ax,ds
        mov     si,offset uline
        vnt     'a'
rd_11:  call    rd_10           ;resalta reg. actual
        cmp     cual,0
        jNe     rd_111
        push    ax
        push    bx
        push    si
        push    dx
        mov     si,2
        mov     dh,4
        mov     bx,offset dhex1
        mov     al,'0'
RD_011: mov     [bx+SI],al
        inc     si
        dec     dh
        jnz     rd_011
        mov     [bx+1],0
        pop     dx
        pop     si
        pop     bx
        pop     ax
rd_b:   mov     bx,offset vr    ;ventana para cambios
        mov     ax,ds
        mov     si,offset uline
        vnt     'a'
getdata 15H,0eH,dhex1,'h'       ;(15H,0eH si en screen)
        jmp     rd_112
rd_111: push    ax
        push    bx
        push    si
        push    dx
        mov     si,2
```

```
        mov     dh,8
        mov     bx,offset dhex1E
        mov     al,'0'
RD_001: mov     [bx+si],al
        inc     si
        dec     dh
        jnz     rd_001
        mov     [bx+1],0
        pop     dx
        pop     si
        pop     bx
        pop     ax
rd_C:   mov     bx,offset vr    ;ventana para cambios
        mov     ax,ds
        mov     si,offset uline
        vnt     'a'
        getdata 11H,0EH,dhex1E,'h'
rd_112: mov     al,atrprev
        mov     di,dirprev
        mov     gdreset,0
        call    rd_102
        cmp     gccod,5
        jb      rd_12
        cmp     gccod,8
        jae     rd_11
        jmp     rd_1f
rd_12:  cmp     gccod,2         ;ABAJO?
        jne     rd_13
rd_121: inc     dl              ;siguiente registro
        cmp     dl,16           ;16 registros
        jne     rd_122
        mov     dl,0
rd_122: jmp     rd_11
rd_13:  cmp     gccod,1         ;ARRIBA?
        jne     rd_14
        dec     dl              ;registro anterior
        cmp     dl,0ffh
        jne     rd_131
        mov     dl,15
rd_131: jmp     rd_11

rd_14:  cmp     gccod,4         ;<---?
        jne     rd_15
        cmp     dl,12
        jb      rd_141
        sub     dl,2
        jmp     rd_142
```

```
rd_141: cmp    dl,6
        jb     rd_1411
        sub    dl,6
        jmp    rd_142
rd_1411:
        cmp    dl,4
        jb     rd_14111
        add    dl,10
        jmp    rd_142
rd_14111:
        add    dl,6
        jmp    rd_142
rd_142: jmp    rd_11
rd_15:  cmp    gccod,3        ;----->?
        jne    rd_16
        cmp    dl,12
        jnb    rd_151
        cmp    dl,6
        jnb    rd_11501
        add    dl,6
        jmp    rd_152
rd_11501:
        cmp    dl,10
        jb     rd_11502
        add    dl,2
        jmp    rd_152
rd_11502:
        sub    dl,6
        jmp    rd_152
rd_151: cmp    dl,14
        jb     rd_1511
        sub    dl,10
        jmp    rd_152
rd_1511:
        add    dl,2
        jmp    rd_152
rd_152: jmp    rd_11
rd_16:  mov    gdreset,1
        cmp    cual,1
        je     rd_1601
        mov    bx,word ptr dhex1+6
        jmp    rd_1611
rd_1601: mov   Ebx,Dword ptr dhex1E+10
rd_1611:
        mov    al,dl
```

```
        actuar
        dw      offset rd_ip,offset rd_sp,offset rd_bp,offset rd_si
        dw      offset rd_di,offset rd_cs,offset rd_ax
        dw      offset rd_bx,offset rd_cx,offset rd_dx
        dw      offset rd_flags,offset rd_ds,offset rd_fs
        dw      offset rd_es,offset rd_gs,offset rd_ss
rd_ip:  mov     regip,Ebx
        jmp     rd_161
rd_sp:  mov     regsp,ebx
        jmp     rd_161
rd_bp:  mov     regbp,Ebx
        jmp     rd_161
rd_si:  mov     regsi,Ebx
        jmp     rd_161
rd_di:  mov     regdi,ebx
        jmp     rd_161
rd_cs:  mov     regcs,bx
        jmp     rd_161
rd_ax:  mov     regax,Ebx
        jmp     rd_161
rd_bx:  mov     regbx,ebx
        jmp     rd_161
rd_cx:  mov     regcx,Ebx
        jmp     rd_161
rd_dx:  mov     regdx,Ebx
        jmp     rd_161
rd_flags:
        mov     eflags,Ebx
        mov     flags,bx
        jmp     rd_161
rd_ds:  mov     regds,bx
        jmp     rd_161
rd_fs:  mov     regfs,bx
        jmp     rd_161
rd_es:  mov     reges,bx
        jmp     rd_161
rd_gs:  mov     reggs,bx
        jmp     rd_161
rd_ss:  mov     regss,bx
rd_161: push    dx
        call    pdebug
        pop     dx
        jmp     rd_a
rd_1f:  mov     ax,regcs
        mov     segcod,ax
        mov     Eax,regip
        mov     offcod,ax
```

```
        call    mcod
        call    mpsw
        cmp     gccod,5         ;TAB?
        jne     rd_1f1
        call    mreg
        call    mpsw
        call    mmem
        jmp     rd_2
rd_1f1: cmp     gccod,6         ;SHIFT TAB?
        jne     rd_1f2
        call    mreg
        call    mpsw
        call    mmem
        jmp     rd_7
rd_1f2: jmp     rd_f            ;ESC.
rd_10:  cmp     dl,0
        jne     rd_esp
        mov     ch,02
        mov     cl,29h
        mov     cual,1
        mov     cual2,0
        jmp     rd_video
rd_esp: cmp     dl,1
        jne     rd_ebp
        mov     ch,3
        mov     cl,29h
        mov     cual,1
        mov     cual2,1
        jmp     rd_video
rd_ebp: cmp     dl,2
        jne     rd_esi
        mov     ch,4
        mov     cl,29h
        mov     cual,1
        mov     cual2,1
        jmp     rd_video
rd_esi: cmp     dl,3
        jne     rd_edi
        mov     ch,5
        mov     cl,29h
        mov     cual,1
        mov     cual2,1
        jmp     rd_video
rd_edi: cmp     dl,4
        jne     rdd_cs
        mov     ch,6
        mov     cl,29h
```

```
        mov     cual,1
        mov     cual2,1
        jmp     rd_video
rdd_cs: cmp     dl,5
        jne     rd_eax
        mov     ch,7
        mov     cl,29h
        mov     cual,0
        mov     cual2,0
        jmp     rd_video
rd_eax: cmp     dl,6
        jne     rd_ebx
        mov     ch,2
        mov     cl,36h
        mov     cual,1
        mov     cual2,1
        jmp     rd_video
rd_ebx: cmp     dl,7
        jne     rd_ecx
        mov     ch,3
        mov     cl,36h
        mov     cual,1
        mov     cual2,1
        jmp     rd_video
rd_ecx: cmp     dl,8
        jne     rd_edx
        mov     ch,4
        mov     cl,36h
        mov     cual,1
        mov     cual2,1
        jmp     rd_video
rd_edx: cmp     dl,9
        jne     rd_efl
        mov     ch,5
        mov     cl,36h
        mov     cual,1
        mov     cual2,1
        jmp     rd_video
rd_efl: cmp     dl,10
        jne     rdd_ds
        mov     ch,6
        mov     cl,36h
        mov     cual,1
        mov     cual2,1
        jmp     rd_video
rdd_ds: cmp     dl,11
        jne     rdd_fs
```

```
        mov   ch,7
        mov   cl,36h
        mov   cual,0
        mov   cual2,0
        jmp   rd_video
rdd_FS: cmp   dl,12
        jne   rdd_es
        mov   ch,6
        mov   cl,42h
        mov   cual,0
        mov   cual2,0
        jmp   rd_video
rdd_es: cmp   dl,13
        jne   rdd_gs
        mov   ch,7
        mov   cl,42h
        mov   cual,0
        mov   cual2,0
        jmp   rd_video
rdd_gs: cmp   dl,14
        jne   rdd_ss
        mov   ch,6
        mov   cl,4ah
        mov   cual,0
        mov   cual2,0
        jmp   rd_video
rdd_ss: mov   ch,7
        mov   cl,4ah
        mov   cual,0
        mov   cual2,0
rd_video:
        posdir cl,ch
        mov   di,pddir
        inc   di
        mov   al,07h
        mov   ah,es:[di]
        mov   atrprev,ah
        mov   dirprev,di
rd_102: cmp   cual,0
        je    rd_reg8
        stosb
        inc   di
        stosb
        inc   di
        stosb
        inc   di
        stosb
```

```
        inc     di
rd_reg8:
        stosb
        inc     di
        stosb
        inc     di
        stosb
        inc     di
        stosb
        retn
rd_2:   ;Cambios en banderas del microprocesador.
        ;Flechas --> y <-- para ir a otra bandera.
        mov     vent,1
        mov     ventactual,1
        mov     dl,0            ;bandera actual OF
        mov     si,offset l0ban ;linea 0
        call    mlin0
        limpln  1,revers        ;linea 1
        print   16,1,l1ban,revers
rd_2a:
        push    f5?
        mov     f5?,0
        call    code?
        pop     f5?
        mov     bx,offset vb    ;ventana para cambios
        vnt     'a'
rd_21:  call    rd_20           ;resalta bandera actual
        mov     al,atrprev
        mov     di,dirprev
        call    rgk0
        call    rd_201          ;quito resaltado
        cmp     getword,1c0dh   ;enter
        jne     rd_22
        mov     gdreset,1       ;dato valido.
        call    rd_210          ;altera variable Flags
        push    dx
        call    mpsw            ;actualiza pantalla
        pop     dx
        jmp     rd_2a
rd_22:
        cmp     getword,4d00h   ; "---->"
        jne     rd_23
        inc     dl              ;siguiente bandera
        cmp     dl,8
        jne     rd_221
        mov     dl,0
rd_221: jmp     rd_21
```

```
rd_23:
        cmp     getword,4b00h   ; " <--- "
        jne     rd_241
        dec     dl              ;bandera anterior
        cmp     dl,0ffh
        jne     rd_231
        mov     dl,7
rd_231: jmp     rd_221
rd_241: mov     ax,regcs
        mov     segcod,ax
        mov     Eax,regip
        mov     offcod,ax
        push    dx              ;preservar ultimo  dl
        call    mcod
        call    mmem
        call    mreg
        call    mpsw
        mov     bx,offset vb    ;ventana para cambios
        vnt     'a'
        pop     dx
        cmp     getword,0f09h   ;TAB
        jne     rd_25
        call    mreg
        call    mpsw
        jmp     rd_3
rd_25:
        cmp     getword,0F00H   ;sTAB
        jne     rd_26
        call    mreg
        call    mpsw
        jmp     rd_1
rd_26:  cmp     getword,011bh   ;ESC
        je      rd_f
        jmp     rd_21

rd_20:  mov     cl,64           ;resalto bandera actual
        add     cl,dl
        add     cl,dl
        mov     ch,3
        posdir  cl,ch
        mov     di,pddir
        inc     di
        mov     al,07h          ;video reverso
        mov     ah,es:[di]      ;atributo previo
        mov     atrprev,ah
        mov     dirprev,di
rd_201: stosb                   ;pone atributo (AL) en bandera
```

```
        add     di,159
        stosb
        retn

rd_210: mov     dh,dl           ;en CL obtengo la posicion
        add     dh,4            ;del bit a alterarse
        mov     cl,dh           ;empezando por el fin.
        cmp     dl,3
        jb      rd_2101
        inc     cl
        cmp     dl,5
        jb      rd_2101
        mov     cl,dl
        sub     cl,3
        add     cl,dh
rd_2101:mov     ax,8000h        ;un uno y quince ceros
        clc                     ;CF=0  (desactivar acarreo)
        rcr     ax,cl           ;rotar CL veces
        xor     flags,ax
        push    eax
        mov     ax,flags
        and     eax,0000ffffh
        and     eflags,0ffff0000h
        or      eflags,eax
        pop     eax
        retn

rd_3:   ;Recibir una direccion y desensamblar desde ella.
        ;PGUP y PGDN para cambiar de paginas.
        ;Flechas --> y <-- para pasar del un campo al otro sin alterarlo.
        mov     vent,1
        mov     ventactual,2
rd_30a: mov     dl,1            ;campo para el offset
        mov     dh,0            ;pagina 0
        mov     ax,offcod
        mov     tabcod,ax
        mov     ax,proxins
        mov     tabcod+2,ax
        mov     gdreset,1
        mov     si,offset l0cod ;linea 0
        call    mlin0

rd_3a:  push    dx
        call    mcod
        pop     dx
        call    code?
        Limpln  1,REVERS        ;linea 1
```

```
        cmp     dh,10
        jae     rd_3a1
        print   9,1,11cod,revers
        jmp     rd_3a2
rd_3a1: print   13,1,11cod1,revers
rd_3a2: mov     bx,offset vcc   ;ventana para dar la direccion
        mov     ax,ds
        mov     si,offset uline
        vnt     'a'
rd_31:
        call    rd_30           ;resalta campo actual, muestra # pagina
        push    ax
        push    dx
        push    cx
        push    bx
        push    si
        push    di
        push    es
        push    ds
        pop     es
        mov     dx,0
        mov     ax,offcod
        mov     bx,5957
        div     bx
        mov     cx,11
        mov     bx,ax
        mov     di,offset barra1
        mov     al,'ß'
        rep     stosb
        pop es
        mov     si,offset barra1
        mov     byte ptr[si+bx],'Ü'
        mov     bx,offset vbarra1
        mov     ax,ds
        vnt     'a'
        pop     di
        pop     si
        pop     bx
        pop     cx
        pop     dx
        pop     ax
        getdata 1fh,12h,dhex1,'h'
        mov     al,atrprev
        mov     di,dirprev
        mov     gdreset,0
        call    rd_301          ;quito resaltado
        cmp     gccod,0
```

```
        jne     rd_311
        mov     gdreset,1       ;dato valido
        mov     dh,0
        push    dx
        call    rd_310          ;altera variables y desensambla
        pop     dx
        jmp     rd_3a
rd_311: cmp     gccod,3
        jb      rd_31
rd_32:  cmp     gccod,3
        jne     rd_33
        xor     dl,1            ;otro campo
        jmp     rd_31
rd_33:  cmp     gccod,4
        jne     rd_34
        xor     dl,1            ;otro campo
        jmp             rd_31
rd_34:  mov     gdreset,1
        cmp     gccod,9
        jne     rd_35
        mov     ax,proxins      ;PGDN
        mov     offcod,ax
        push    dx
        call    mcod
        pop     dx
        cmp     dh,9            ;A la pagina 9 se regresa desde
        ja      rd_341          ;todas las paginas posteriores,
        inc     dh              ;no dejo que DH pase de 10.
rd_341: mov     al,2
        mul     dh
        mov     si,offset tabcod
        add     si,ax           ;SI apunta a la DH-esima pos. en TABCOD
        mov     ax,proxins
        mov     [si+2],ax
        jmp     rd_3a
rd_35:  cmp     gccod,8
        jne     rd_36
        cmp     dh,0            ;PGUP. No bajo de la pagina 0.
        jne     rd_351
        jmp     rd_3a
rd_351: dec     dh              ;pongo pagina de arriba
        mov     al,2
        mul     dh
        mov     si,offset tabcod
        add     si,ax
        mov     ax,[si]
        mov     offcod,ax
```

```
        push    dx
        call    mcod
        pop     dx
        jmp     rd_3a
rd_36:                  ;ventana Opcmem2
        call    mstk        ;ventana Stack
        mov     ax,regcs
        mov     segcod,ax
        mov     Eax,regip
        mov     offcod,ax
        call    mcod        ;ventana Codigo
        cmp     gccod,5
        jne     rd_37
        call    mexpc
        call    mmem
        call    mreg
        call    mpsw
        jmp     rd_5        ;TAB
rd_37:  cmp     gccod,6
        jne     rd_38
        call    mexpc
        call    mmem
        call    mreg
        call    mpsw
        jmp     rd_2        ;SHIFT TAB
rd_38:  jmp     rd_f        ;ESC
rd_30:  mov     al,dh       ;resalto area de segmento o de offset
        add     al,'0'      ;primero muestro # de pagina
        cmp     dh,10
        jb      rd_302
        mov     al,'X'      ;pagina 10 o superior
rd_302: mov     di,46
        stosb
        mov     al,5
        mul     dl
        mov     cl,al
        inc     cl
        mov     ch,3
        posdir  cl,ch
        mov     di,pddir
        inc     di
        mov     al,07h      ;video reverso
        mov     ah,es:[di]  ;atributo previo
        mov     atrprev,ah
        mov     dirprev,di
rd_301: stosb               ;Atributo AL en segmento u offset
        inc     di
```

```
        stosb
        inc     di
        stosb
        inc     di
        stosb
        retn
rd_310: mov     ax,word ptr dhex1+6     ;desensamblo desde dir. dada
        cmp     dl,0            ;Antes veo si lo alterado
        jne     rd_3101         ;fue el segmento o el offset
        mov     segcod,ax
        jmp     rd_312
rd_3101:mov     offcod,ax
        mov     tabcod,ax
rd_312: call    mcod
        mov     ax,proxins
        mov     tabcod+2,ax
        cmp     f5?,1
        je      rd_3fin
        call    mreg
        call    mpsw
        call    mmem
rd_3fin:retn

rd_5:   ;Cambios en las opciones de ventanas de memoria relativa.
        ;Flechas --> y <-- para pasar a otra de las opciones; arriba
        ;y abajo para ir a otra linea de la ventana; y PGUP y PGDN
        ;para pasar de Segmento a Offset o a la inversa.
        mov     vent,1
        mov     ventactual,3
        mov     dl,0
        limpln  1,revers
        print   2,1,11op2,revers
RD51A:
        mov     dh,0
        mov     si,offset l0op21
        mov     readerr,0
        call    mlin0
        mov     bx,offset vmemrel       ;ventana de memoria relativa
        mov     ax,ds
        mov     si,offset uline2
        vnt     'a'
        mov     gdreset,1
rd511:  call    rd510j          ;resalta linea actual
        push    ax
        push    dx
        push    cx
        push    bx
```

```
        push    si
        push    di
        push    es
        push    ds
        pop     es
        mov     dx,0
        mov     si,offset autooff
        mov     ax,[si]         ;offcod
        mov     bx,5957
        div     bx
        mov     cx,11
        mov     bx,ax
        mov     di,offset barra2
        mov     al,'▒'
        rep     stosb
        pop es
        mov     si,offset barra2
        mov     byte ptr[si+bx],'▒'
        mov     bx,offset vbarra2
        mov     ax,ds
        vnt     'a'
        pop     di
        pop     si
        pop     bx
        pop     cx
        pop     dx
        pop     ax
        getdata2  7h,0eh,dhexMR,'h'     ;regresa el buffer rotado y
                                        ;almacenado segmento y offset
        cmp     readerr,1       ;si ocurrio error en getdata2 buelvo al inicio
        je      rd51a           ;para que ingrese nuevamente datos, ya antes se
                                  ;se borro buffers dhex3 y dhexMr
        mov     al,atrprev      ;quito resaltado de linea y opción
        mov     di,dirprev
        call    rd5101j
        mov     gdreset,0
        cmp     gccod,0
        je      rd51110
        cmp     gccod,1         ;flecha arriba? navegacion en ventana relativa
        je      rd5550
        cmp     gccod,2         ;flecha abajo?
        je      rd5550
        cmp     gccod,8         ;PgUp?
        je      rd5550
        cmp     gccod,9         ;PgDn?
        jne     rd5554
rd5550: jmp     rd5551
```

```
error1: push   dx
        mov    ah,2
        mov    dl,7
        int    21h
        pop    dx
        print  0,23,overflow,revers
        call   error0
        jmp    rd51a
rd5551: push   eax
        mov    eax,autoseg
        mov    xxxxseg,eax
        pop    eax
rd5552: jmp    rd5553
error2: push   dx
        mov    ah,2
        mov    dl,7
        int    21h
        pop    dx
        print  0,23,overflow,revers
        call   error0
        jmp    rd51a
rd5553: push   eax
        mov    eax,autooff
rd5402: mov    xxxxoff,eax
        pop    eax
        jmp    rd53002
rd5554: cmp    gccod,5        ;TAB?
        jne    rd518
        call   mcod           ;restaurar la pantalla
        call   mreg
        call   mpsw
        call   mmem
        jmp    rd_6
rd518:  cmp    gccod,6        ;sTAB?
        jne    rd519
        call   mcod           ;restaurar la pantalla
        call   mreg
        call   mpsw
        call   mmem
        jmp    rd_3
rd519:  jmp    rd_f           ;ESC
rd51110:cmp    dhexseg,0
        jb     error3
        cmp    dhexseg,0ffffh
        ja     error3
        jmp    rd5301
error3: push   dx             ;overflow
```

```
        mov     ah,2
        mov     dl,7
        int     21h
        pop     dx
        print   0,23,overflow,revers
        call    error0
        jmp     rd51a
rd5301: push    eax
        mov     eax,dhexseg
        mov     xxxxseg,eax
        pop     eax
RD5300:
        cmp     dhexoff,0
        jb      error4
        cmp     dhexoff,0ffffh
        ja      error4
        jmp     rd5302
error4: push    dx              ;overflow
        mov     ah,2
        mov     dl,7
        int     21h
        pop     dx
        print   0,23,overflow,revers
        call    error0
        jmp     rd51a
error0: mov     bx,offset dhex3
        mov     dh,[bx]
        mov     al,0
        mov     [1+bx],al
        mov     si,2
        mov     al,'0'
rd10a:  mov     [si+bx],al
        inc     si
        dec     dh
        jnz     rd10a
        mov     bx,offset dhexMr
        mov     dh,[bx]
        mov     al,0
        mov     [bx+1],al
        mov     si,2
        mov     al,'0'
rd10b:  mov     [si+bx],al
        inc     si
        dec     dh
        jnz     rd10b
        mov     posini,0
        mov     posmedium,0
```

```
        mov     signopos,0
        mov     signobef,0
        mov     dhexseg,0
        mov     dhexoff,0
        mov     readerr,1
        call    mreg
        call    mpsw
        retn
rd5302: push    eax
        mov     eax,dhexoff
        cmp     eax,0ff98h
        jb      rd5403
        mov     eax,0ff98h      ;no se pase de 0fffffh en ventana de mem. rel.
        mov     autooff,eax
rd5403: MOV     xxxxoff,eax
        pop     eax

rd53002:call    mcod
        call    mstk
        call    mexpc
        call    preopc
        call    mmem
        mov     dhexseg,0h          ;borrar ultimos datos ingresados de
        mov     dhexoff,0h          ;seg. y off. en memoria relativa
        mov     signobef,0
        mov     signopos,0
        jmp     rd51a               ;y volver al inicio
                                        ;OJO  retorno con jmp y no con retn
rd510J: mov     cl,25h              ;resalta linea actual
        mov     ch,9h
        posdir  cl,ch
        mov     di,pddir
        inc     di
        mov     ah,es:[di]          ;atributo previo
        mov     atrprev,ah
        mov     dirprev,di
        mov     al,07h
rd5101j:mov     cx,9h
rd5102J:stosb
        inc     di
        loop    rd5102J
        call    mreg
        call    mpsw
        retn

rd_6:   ;Cambios en memoria directamente desde teclado. Las flechas -->
        ;y <-- no hacen cambio de linea. Arriba y Abajo no cambian de
```

```
        ;pagina, son para pasar de linea o ventana. PGUP y PGDN para pasar
        ;a otra pagina (+-64 en la ventana absoluta, +-8 en las relativas)
        ;;mov    rd6?,0          ;bandera para rd600 y rd700

        mov     vent,1
        mov     ventactual,4
        mov     dl,0            ;DL=# de linea (0-12)
        mov     dh,0            ;DH=# de columna (0-7)
        mov     si,offset l0hex ;linea 0
        call    mlin0
        limpln  1,REVERS        ;linea 1
        print   6,1,l1hex,REVERS
        mov     gdreset,1
rd6a:   pusha
        push    f5?
        mov     f5?,0
        call    code?
        pop     f5?
        popa
        mov     bx,offset vdh   ;dibujo la ventana
        mov     ax,ds
        mov     si,offset uline
        vnt     'a'
rd61:   call    rd60            ;resalto byte actual
        push    ax
        push    dx
        push    cx
        push    bx
        push    si
        push    di
        push    es
        push    ds
        pop     es
        mov     dx,0
        mov     si,offset autooff
        mov     ax,[si] ;offcod
        mov     bx,5957
        div     bx
        mov     cx,11
        mov     bx,ax
        mov     di,offset barra2
        mov     al,'▒'
        rep     stosb
        pop es
        mov     si,offset barra2
        mov     byte ptr[si+bx],'▀'
        mov     bx,offset vbarra2
```

```
        mov     ax,ds
        vnt     'a'
        pop     di
        pop     si
        pop     bx
        pop     cx
        pop     dx
        pop     ax
        getdata 14h,0eh,dbyte,'h' ;recibo entrada de teclado
        mov     al,atrprev      ;quito resaltados
        mov     di,dirprev
        mov     gdreset,0
        call    rd601
        cmp     gccod,0         ;dato valido?
        jne     rd611
        mov     gdreset,1       ;si
        call    rd610           ;altero memoria, actualizo pantalla
        inc     dh              ;paso al proximo byte
        cmp     dh,8            ;hasta llegar al fin de la ventana
        jne     rd6a
        mov     dh,0
        inc     dl
        cmp     dl,13
        jb      rd6a
        dec     dl
        mov     dh,7
        jmp     rd6a
rd611:  cmp     gccod,1         ;arriba?
        jne     rd612
        dec     dl              ;si
        cmp     dl,0ffh
        jne     rd6111
        mov     dl,12
rd6111: jmp     rd61
rd612:  cmp     gccod,2         ;abajo?
        jne     rd613
        inc     dl              ;si
        cmp     dl,13
        jne     rd6121
        mov     dl,0
rd6121: jmp     rd61
rd613:  cmp     gccod,3         ;-->?
        jne     rd614
        inc     dh              ;si
        cmp     dh,8
        jne     rd6131
        mov     dh,0
```

```
rd6131: jmp   rd61
rd614:  cmp   gccod,4       ;<--?
        jne   rd615
        dec   dh            ;si
        cmp   dh,0ffh
        jne   rd6141
        mov   dh,7
rd6141: jmp   rd61
rd615:  cmp   gccod,8       ;PGUP?
        je    rd616
        cmp   gccod,9       ;o PGDN?
        jne   rd617
rd616:  call  rd6160        ;si
        jmp   rd61
rd617:  call  mcod          ;restauro pantalla
        call  mstk
        call  mexpc
        call  mmem
        call  mpsw
        cmp   gccod,5       ;TAB?
        jne   rd62
        call  mreg
        call  mpsw
        jmp   rd_7          ;si
rd62:   cmp   gccod,6       ;SHIFT TAB?
        jne   rd63
        call  mreg
        call  mpsw
        jmp   rd_5          ;si
rd63:   jmp   rd_f          ;ESC
rd60:   mov   cl,dh         ;resalto byte actual. Primero
        mov   ch,dl         ;calculo posicion
        mov   al,3
        mul   cl
        add   al,2fh
        mov   cl,al
        add   ch,9
        posdir cl,ch
        mov   di,pddir
        inc   di
        mov   ah,es:[di]
        mov   atrprev,ah
        mov   dirprev,di
        mov   al,07h
rd601:  stosb               ;cambio el atributo
        inc   di
        stosb
```

```
        retn
rd610:  push    dx              ;altera memoria
rd6101: push    eax
        mov     eax,autooff
        mov     di,ax
        pop     eax
        mov     es,autoseg
        mov     al,8
        mul     dl
        add     al,dh
        mov     ah,0
        add     di,ax           ;ES:DI apunta a byte a cambiarse
        mov     al,dbyte+4
        stosb
        mov     es,videoseg
        push    dx
        call    mmemx           ;actualizo ventana de memoria
        pop     dx
rd610f: call    mcod
        call    mstk
        call    mexpc
        pop     dx
        retn
rd6160: mov     si,offset autooff       ;dir1  ;si, SI apunta a direccion a alterarse
        mov     cx,104          ;CX tiene el valor a sumar o restar
rd61602:cmp     gccod,8         ;PGUP?
        jne     rd61603
        sub     [si],cx         ;si
        push    eax
        mov     ax,[si]
        mov     xxxxoff,eax
        pop     eax
        jmp     rd61604
rd61603: add    [si],cx         ;PGDN
        push    eax
        mov     ax,[si]
        mov     xxxxoff,eax
        pop     eax
rd61604:push    dx
        call    preopc
        call    mmemx
        pop     dx
rd61605:retn

rd_7:   ;Cambios en memoria directamente desde teclado (ASCII). Las flechas
        ;--> y <-- no hacen cambio de linea. Arriba y Abajo no cambian de
        ;pagina, son para pasar de linea o ventana. PGUP y PGDN para pasar
```

```
        ;a otra pagina (+-64 en la ventana absoluta, +-8 en las relativas)
        mov   vent,1
        mov   ventactual,5
        mov   dl,0          ;DL=# de linea (0-12)
        mov   dh,0          ;DH=# de columna (0-7)
        mov   si,offset l0asc ;linea 0
        call  mlin0
        limpln 1,revers     ;linea 1
        print 6,1,11hex,revers
rd7a:   pusha
        push  f5?
        mov   f5?,0
        call  code?
        pop   f5?
        popa
        mov   bx,offset vda   ;dibujo la ventana
        vnt   'a'
rd71:   call  rd70          ;resalto ascii actual
        push  ax
        push  dx
        push  cx
        push  bx
        push  si
        push  di
        push  es
        push  ds
        pop   es
        mov   dx,0
        mov   si,offset autooff
        mov   ax,[si]       ;offcod
        mov   bx,5957
        div   bx
        mov   cx,11
        mov   bx,ax
        mov   di,offset barra2
        mov   al,'▒'
        rep   stosb
        pop   es
        mov   si,offset barra2
        mov   byte ptr[si+bx],'▀'
        mov   bx,offset vbarra2
        mov   ax,ds
        vnt   'a'
        pop   di
        pop   si
        pop   bx
        pop   cx
```

```
        pop     dx
        pop     ax
        getchar                 ;recibo entrada de teclado
        mov     al,atrprev      ;quito resaltado
        mov     di,dirprev
        stosb
        cmp     gccod,1         ;ENTER?
        jne     rd71x
        mov     gccod,0         ;si, simulo GCCOD=0
        mov     gcascii,13      ;Enter=^M
rd71x:  cmp     gccod,0         ;dato valido?
        jne     rd711
        call    rd710           ;si, alt. memoria, act. pantalla, -->.
        mov     bx,offset vda   ;dibujo la ventana ASCII
        vnt     'a'
        push    f5?
        mov     f5?,0
        call    code?
        pop     f5?
        inc     dh              ;paso al proximo byte
        cmp     dh,8            ;hasta llegar al fin de la ventana
        jne     rd71
        mov     dh,0
        inc     dl
        cmp     dl,13
        jb      rd71
        dec     dl
        mov     dh,7
        jmp     rd71
rd711:  cmp     gccod,5         ;especiales?
        je      rd711x
        jmp     rd717
rd711x: cmp     gcascii,72      ;arriba?
        jne     rd712
        dec     dl              ;si
        cmp     dl,0ffh
        jne     rd7111
        mov     dl,12
rd7111: jmp     rd71
rd712:  cmp     gcascii,80      ;abajo?
        jne     rd713
        inc     dl              ;si
        cmp     dl,13
        jne     rd7121
        mov     dl,0
rd7121: jmp     rd71
rd713:  cmp     gcascii,77      ;-->?
```

```
        jne     rd714
        inc     dh              ;si
        cmp     dh,8
        jne     rd7131
        mov     dh,0
rd7131: jmp     rd71
rd714:  cmp     gcascii,75      ;<--?
        jne     rd715
        dec     dh              ;si
        cmp     dh,0ffh
        jne     rd7141
        mov     dh,7
rd7141: jmp     rd71
rd715:  cmp     gcascii,73      ;PGUP?
        je      rd716
        cmp     gcascii,81      ;o PGDN?
        je      rd716
        jmp     rd71            ;tecla invalida
rd716:  call    rd7160          ;PGUP O PGDN
        jmp     rd71
rd717:  call    pdebug          ;restauro pantalla
        cmp     gccod,2         ;TAB?
        jne     rd72
        jmp     rd_1            ;si
rd72:   cmp     gccod,3         ;SHIFT TAB?
        jne     rd73
        jmp     rd_6            ;si
rd73:   jmp     rd_f            ;ESC
rd70:   mov     cl,dh           ;resalto byte actual. Primero
        mov     ch,dl           ;calculo posicion
        add     cl,47h
        add     ch,9
        posdir  cl,ch
        mov     di,pddir
        inc     di
        mov     ah,es:[di]
        mov     atrprev,ah
        mov     dirprev,di
        mov     al,07h
        stosb                   ;cambio el atributo
        retn
rd710:  push    dx              ;altera memoria
rd7101: push    eax
        mov     eax,autooff
        mov     di,ax
        pop     eax
        mov     es,autoseg      ;dir1+2
```

```
        mov   al,8
        mul   dl
        add   al,dh
        mov   ah,0
        add   di,ax          ;ES:DI apunta a byte a cambiarse
        mov   al,gcascii
        stosb
        mov   es,videoseg
        push  dx
        call  mmemx          ;actualizo ventana de memoria
        pop   dx
        jmp   rd710f
rd710f: call  mcod
        call  mstk
        call  mexpc
        pop   dx
        retn
rd7160: cmp   dl,8           ;ventana absoluta?
        mov   si,offset autooff      ;dir1  ;si, SI apunta a direccion a alterarse
        mov   cx,104         ;CX tiene el valor a sumar o restar
rd71602:cmp   gcascii,73     ;PGUP?
        jne   rd71603
        sub   [si],cx        ;si
        push  eax
        mov   ax,[si]
        mov   xxxxoff,eax
        pop   eax
        jmp   rd71604
rd71603: add  [si],cx        ;PGDN
        push  eax
        mov   ax,[si]
        mov   xxxxoff,eax
        pop   eax
rd71604:push  dx
        call  preopc
        call  mmemx
        pop   dx
rd71605:retn

rd_f:
        mov   vent,0
        mov   si,offset l0ini
        call  mlin0
        call  pdebug
        limpln 23,normal
        print 0,23,l1ini,bmenu
RINIF6: ret
```

```
;------------ RUTINAS COPROCESADOR ------------
RDF3M:
        fnstcw  contcon         ;cargar valores originales  status y control
        fnstsw  statcon
        fsave   mem_loc
        frstor  mem_loc
        mov     di,offset memocop1
        mov     esi,offset mem_loc      ;almaceno TAG
        add     esi,4
        mov     bx,[si]
        push    es
        push    ds
        pop     es
        push    bx
        mov     al,bh
        hexasc
        mov     ax,bx
        xchg    al,ah
        stosw
        pop     bx
        mov     al,bl
        hexasc
        mov     ax,bx
        xchg    al,ah
        stosw
        mov     esi,offset mem_loc      ;almaceno IPTR  (MSB)
        add     esi,9
        mov     bl,byte ptr[si]
        rcr     bl,4
        mov     al,bl
        hexasc
        mov     al,bl
        stosb
        mov     esi,offset mem_loc
        add     esi,6
        mov     bx,[si]
        push    bx
        mov     al,bh
        hexasc
        mov     ax,bx
        xchg    al,ah
        stosw
        pop     bx
        mov     al,bl
        hexasc
        mov     ax,bx
        xchg    al,ah
```

```
stosw
mov   esi,offset mem_loc     ;almaceno OPCODE
add   esi,9
mov   bl,byte ptr[si]
and   bl,7
mov   al,bl
hexasc
mov   al,bl
stosb
mov   esi,offset mem_loc
add   esi,8
mov   bl,[si]
mov   al,bl
hexasc
mov   ax,bx
xchg  al,ah
stosw
mov   esi,offset mem_loc     ;almaceno OPTR
add   esi,13
mov   bl,[si]
rcr   bl,4
mov   al,bl
hexasc
mov   al,bl
stosb
mov   esi,offset mem_loc
add   esi,10
mov   bx,[si]
push  bx
mov   al,bh
hexasc
mov   ax,bx
xchg  al,ah
stosw
pop   bx
mov   al,bl
hexasc
mov   ax,bx
xchg  al,ah
stosw
pop   es
mov   prevdic,di             ;almaceno puntero  de memoria1
push  si
mov   si,offset mem_loc
push  cx
mov   cx,[si+2]
and   ch,38h
```

```
        mov   cl,2
        shr   ch,cl
        mov   cl,ch
        ror   word ptr [si+4],cl
        pop   cx
        cmp   word ptr[si+4],0ffffh
        pop   si
        je    copvacio
      push    di
        mov   di,offset memocop1+17      ;memoria coprocesador
        push  es
        push  ds
        pop   es          ;es=ds
        mov   cx,312
        mov   al,' '
lazo2:  stosb             ;inicio borrando memocop1
        loop  lazo2
        pop   es
        pop   di
        mov   prevdic,di     ;contiene puntero para mem. coprocesador
        mov   esi,offset mem_loc  ;registro ST de la pila del coprocesador
        mov   ax,[si+4]
        and   ax,3h
        cmp   ax,3h
        jne   reg601
        add   prevdic,39
        jmp   reg60
reg601:
        add   esi,14        ;apuntar a st(0)
        call  save
reg60:  mov   esi,offset mem_loc ;almaceno st(1)
        mov   ax,[si+4]
        and   ax,0ch
        cmp   ax,0ch
        jne   reg611
        add   prevdic,39
        jmp   cop611
reg611: add   esi,24        ;apuntar a st(1)
        call  save
cop611: mov   esi,offset mem_loc  ;almaceno st(2)
        mov   ax,[si+4]
        and   ax,30h
        cmp   ax,30h
        jne   reg621
        add   prevdic,39
        jmp   cop621
reg621: add   esi,34        ;apuntar a st(2)
```

```
        call    save
cop621: mov   esi,offset mem_loc  ;almaceno st(3)
        mov   ax,[si+4]
        and   ax,0c0h
        cmp   ax,0c0h
        jne   reg631
        add   prevdic,39
        jmp   cop63
reg631:
        add   esi,44          ;apuntar a st(4)
        call  save
cop63:  mov    esi,offset mem_loc  ;almaceno st(4)
        mov   ax,[si+4]
        and   ax,300h
        cmp   ax,300h
        jne   reg641
        add   prevdic,39
        jmp   cop64
reg641: add   esi,54          ;apuntar a st(4)
        call  save
cop64:  mov    esi,offset mem_loc  ;almaceno st(5)
        mov   ax,[si+4]
        and   ax,0c00h
        cmp   ax,0c00h
        jne   reg651
        add   prevdic,39
        jmp   cop65
reg651: add   esi,64          ;apuntar a st(5)
        call  save
cop65:  mov    esi,offset mem_loc  ;almaceno st(6)
        mov   ax,[si+4]
        and   ax,3000h
        cmp   ax,3000h
        jne   reg661
        add   prevdic,39
        jmp   cop66
reg661: add   esi,74          ;apuntar a st(6)
        call  save
cop66:  mov    esi,offset mem_loc  ;almaceno st(7)
        mov   ax,[si+4]
        and   ax,0c000h
        cmp   ax,0c000h
        jne   reg671
        add   prevdic,39
        jmp   cop67
reg671: add   esi,84          ;apuntar a st(7)
        call  save
```

```
        jmp     cop67
save;   call    savest          ;guarda exponente y mantisa de ST(n) en ven_
                                ;tana para memoria1 del coprocesador
        call    expcop          ;analiza exponente actual de la pila del
                                ;coprocesador y lo convierte a decimal
        call    savemant        ;guarda exp. y mantisa en formato real en la
        ret                     ;ventana para memoria1 del coprocesador

cop67:  mov     di,offset memocop2      ;memoria para satatus y control
        mov     prevdic,di
        call    savecont        ;almacena palabra de control en memoria2 para
                                ;el coprocesador
        mov     di,prevdic
        call    savestat        ;almacena palabra de status en memoria2 para
                                ;el coprocesador
        retn


SAVEST: mov     bx,word ptr[si+8]       ;guarda exponente y mantisa de st(n)
        push    es
        push    ds
        pop     es
        mov     di,prevdic              ;apunta a siguiente byte en memoria1
        push    bx
        mov     al,bh
        hexasc                  ;convierto a ASCII
        mov     ax,bx
        xchg    al,ah
        stosw
        pop     bx
        mov     al,bl
        hexasc
        mov     ax,bx
        xchg    al,ah
        stosw
        mov     bx,word ptr[si+6]       ;almaceno mantisa
        push    bx
        mov     al,bh
        hexasc
        mov     ax,bx
        xchg    al,ah
        stosw
        pop     bx
        mov     al,bl
        hexasc
        mov     ax,bx
        xchg    al,ah
```

```
        stosw
        mov     bx,word ptr[si+4]
        push    bx
        mov     al,bh
        hexasc
        mov     ax,bx
        xchg    al,ah
        stosw
        pop     bx
        mov     al,bl
        hexasc
        mov     ax,bx
        xchg    al,ah
        stosw
        mov     bx,word ptr[si+2]
        push    bx
        mov     al,bh
        hexasc
        mov     ax,bx
        xchg    al,ah
        stosw
        pop     bx
        mov     al,bl
        hexasc
        mov     ax,bx
        xchg    ah,al
        stosw
        mov     bx,word ptr[si]
        push    bx
        mov     al,bh
        hexasc
        mov     ax,bx
        xchg    al,ah
        stosw
        pop     bx
        mov     al,bl
        hexasc
        mov     ax,bx
        xchg    al,ah
        stosw
        mov     prevdic,di      ;almaceno puntero final
        pop     es
        mov     prevdic,di
        retn
copvacio:push   es
        push    ds
        pop     es              ;es=ds
```

```
        mov    di,offset memocop1+17
        mov    cx,312
        mov    al,' '
lazo:   stosb
        loop   lazo
        pop    es
        jmp    cop67
EXPCOP: ;rutina que analiza el exponente y lo convierte a decimal
        ;sale con codigo de acuerdo al tipo de dato asi:
        ;copcop=0 si # es valido; copcop=2 si se trata de numero no valido
        ;(3ffffh < exp < -3ffffh); copcop=3 si # es un cero (exp=0000)
        mov    copcop,0
        mov    ax,word ptr[si+8]
        cmp    ax,0                ;analizo si dato es valido
        jne    cop100
        mov    copcop,3
        mov    signm,0
        jmp    fincop3
cop100: cmp    ax,0ffffh
        jne    cop101
        mov    copcop,2
        mov    signm,1         ;-NAN  si EXP < -16383
        jmp    fincop3
cop101: cmp    ax,7fffh
        jne    cop102
        mov    copcop,2
        mov    signm,0         ;+NAN   si EXP > 16383

cop102: test   ax,8000h        ;numero + o - ? (analizo signo mantisa)
        jz     cop10
        mov    signm,1         ;valor "-"
        jmp    cop11
cop10:  mov    signm,0         ;valor "+"
cop11:  and    ax,7fffh        ;elimono signo
        sub    ax,3fffh        ;exponente polarizado
        sub    ax,1ch          ;por la multiplicacion que se afectara a la
        cmp    ax,8000h        ;mantisa
        jb     cop12
        mov    signexp,1       ;exponente "-"
        xor    ax,0ffffh       ;calculo complemento al 2 para convertir el
        add    ax,1            ;exponente a un valor positivo
        jmp    cop13
cop12:  mov    signexp,0       ;exponente "+"
cop13:  mov    bx,10000
        mov    dx,0
        div    bx
        shl    eax,16
```

```
        mov     dms,eax
        mov     ax,dx           ;en DX el resto de la division
        mov     bx,1000
        mov     dx,0
        div     bx
        shl     ax,12
        add     dms,eax
        mov     ax,dx           ;DX contiene el resto de la division
        mov     bx,100
        mov     dx,0
        div     bx
        shl     ax,8
        add     dms,eax
        mov     ax,dx
        mov     bx,10
        mov     dx,0
        div     bx
        shl     ax,4
        add     dms,eax
        add     dms,edx         ;finalmente sumo el resto
fincop3: retn

SAVEMANT:       ;almacena exponente y mantisa ( valores reales )
        cmp     copcop,2        ;analizo tipo de dato
        jne     cop31
        push    esi             ; se trata de un "NAN"
        mov     di,prevdic
        push    es
        push    ds
        pop     es              ;ds=es
        cmp     signm,0         ;dato + o - ?
        jne     cop32
        mov     si,offset NANPOS
        mov     cx,19
        rep     movsb
        pop     es
        mov     prevdic,di
        pop     esi
        retn
cop32:  mov     si,offset NANNEG
        mov     cx,19
        rep     movsb
        pop     es
        mov     prevdic,di
        pop     esi
        retn
cop31:  cmp     copcop,3
```

```
        jne     cop30
        push    esi
        mov     di,prevdic      ;se trata de un cero
        mov     si,offset ZERO
        mov     cx,19
        push    es
        push    ds
        pop     es              ;ds=es
        rep     movsb
        pop     es
        mov     prevdic,di      ;guardo puntero de memoria1
        pop     esi
        retn
cop30:  ;se trata entonces de un dato valido, almaceno mantisa y exponente
        call    mantisa         ;regresa con valor convertido en decimal de
                                ;la mantisa en variable mantisf
        push    es
        push    ds
        pop     es              ; ds=es
        cmp     signm,0         ;signo de mantisa + o - ?
        jne     cop33
        mov     di,prevdic
        mov     al,'+'
        stosb
        mov     prevdic,di
        jmp     cop34
cop33:  mov     di,prevdic      ;# negativo
        mov     al,'-'
        stosb
        mov     prevdic,di
cop34:  mov     di,offset mantisf
        mov     bl,byte ptr[di+4]
        mov     al,bl
        hexasc
        mov     di,prevdic
        mov     al,bl
        stosb
        mov     prevdic,di
        mov     di,offset mantisf
        mov     bx,word ptr[di+2]
        push    bx
        mov     al,bh
        hexasc
        mov     di,prevdic
        mov     ax,bx
        xchg    ah,al
        stosw
```

```
        pop     bx
        mov     al,bl
        hexasc
        mov     ax,bx
        xchg    ah,al
        stosw
        mov     prevdic,di

        mov     di,offset mantisf
        mov     bx,word ptr[di]
        push    bx
        mov     al,bh
        hexasc
        mov     di,prevdic
        mov     ax,bx
        xchg    al,ah
        stosw
        pop     bx
        mov     al,bl
        hexasc
        mov     ax,bx
        xchg    ah,al
        stosw
        mov     al,'x'
        stosb
        mov     al,'2'
        stosb
        mov     al,'^'
        stosb
        mov     prevdic,di
        cmp     signexp,0
        je      cop350
        mov     di,prevdic          ;exponente negativo
        mov     al,'-'
        stosb
        mov     prevdic,di
        jmp     cop351
cop350: mov     di,prevdic
        mov     al,'+'
        stosb
        mov     prevdic,di
cop351: mov     eax,dms             ;dms contiene dato en decimal del exp.
        test    eax,0f0000h
        jnz     cop355              ;exponente de 5 dígitos
        test    eax,0f000h
        jnz     cop354              ;exponente de 4 dígitos
        test    eax,0f00h
```

```
        jnz     cop353          ;exponente de 3 dígitos
        test    eax,0f0h
        jnz     cop352          ;exponente de 2 dígitos
        mov     di,offset dms    ;exponente de 1 dígito
        mov     al,byte ptr[di]
        mov     di,prevdic
        hexasc
        mov     al,bl
        stosb
        mov     al,' '
        stosb
        stosb
        stosb
        stosb
        mov     prevdic,di
        pop     es
        retn
cop352: mov     di,offset dms
        mov     bl,byte ptr[di]
        mov     al,bl
        hexasc
        mov     ax,bx
        xchg    al,ah
        mov     di,prevdic
        stosw
        mov     al,' '
        stosb
        stosb
        stosb
        mov     prevdic,di
        pop     es
        retn
cop353: mov     di,offset dms
        mov     bl,byte ptr[di]
        mov     al,bl
        hexasc
        mov     ax,bx
        xchg    al,ah
        mov     di,prevdic
        stosw
        mov     prevdic,di
        mov     di,offset dms
        mov     al,byte ptr[di+1]
        mov     di,prevdic
        hexasc
        mov     al,bl
        stosb
```

```
        mov     al,' '
        stosb
        stosb
        mov     prevdic,di
        pop     es
        retn
cop354: mov     di,offset dms
        mov     bx,word ptr[di]
        push    bx
        mov     al,bl
        hexasc
        mov     ax,bx
        mov     di,prevdic
        xchg    al,ah
        stosw
        pop     bx
        mov     al,bl
        hexasc
        mov     ax,bx
        xchg    al,ah
        stosw
        mov     prevdic,di
        pop     es
        retn
cop355: mov     di,offset dms
        mov     bx,word ptr[di]
        push    bx
        mov     al,bh
        hexasc
        mov     ax,bx
        mov     di,prevdic
        xchg    al,ah
        stosw
        pop     bx
        mov     al,bl
        hexasc
        mov     ax,bx
        xchg    al,ah
        stosw
        mov     prevdic,di
        mov     di,offset dms
        mov     al,byte ptr[di+2]
        mov     di,prevdic
        hexasc
        mov     al,bl
        stosb
        mov     di,prevdic
```

```
        pop     es
        retn

MANTISA:
        clc
        mov     ebx,dword ptr[si+4]
        rcl     ebx,1
        mov     var2,2
        mov     cx,1
        mov     mantis,10000000h
cop20:  mov     eax,10000000h
        rcl     ebx,1
        jc      cop21                   ;es un cero o uno?
        push    eax                     ;es un cero
        push    ebx
        mov     eax,var2
        mov     ebx,2
        mul     ebx
        mov     var2,eax
        pop     ebx
        pop     eax
        jmp     cop22
cop21:  mov     edx,0
        div     var2            ;entonces es un " 1 "
        add     mantis,eax
        push    eax
        push    ebx
        mov     eax,var2
        mov     ebx,2
        mul     ebx
        mov     var2,eax
        mov     eax,var2
        pop     ebx
        pop     eax

cop22:  inc     cx
        cmp     cx,28           ;suficiente presicion practica
        jb      cop20
        mov     eax,mantis

        ;conversion a decimal de la mantisa
        mov     ebx,100000000
        mov     edx,0
        div     ebx
        mov     di,offset mantisf
        push    eax
        xor     eax,eax
```

```
mov   [di],eax          ;borro el contenido de mantisa
pop   eax
mov   byte ptr [di+4],al

mov   eax,edx       ;reciduo a edx
mov   edx,0
mov   ebx,10000000
div   ebx
shl   eax,28
add   [di],eax

mov   eax,edx
mov   edx,0
mov   ebx,1000000
div   ebx
shl   eax,24
add   [di],eax

mov   eax,edx
mov   edx,0
mov   ebx,100000
div   ebx
shl   eax,20
add   [di],eax

mov   eax,edx
mov   edx,0
mov   ebx,10000
div   ebx
shl   eax,16
add   [di],eax
mov   eax,edx
mov   edx,0
mov   ebx,1000
div   ebx
shl   eax,12
add   [di],eax

mov   eax,edx
mov   edx,0
mov   ebx,100
div   ebx
shl   eax,8
add   [di],eax

mov   eax,edx
mov   edx,0
```

```
        mov     ebx,10
        div     ebx
        shl     eax,4
        add     [di],eax
        add     [di],edx
        retn

SAVESTAT:                   ;guarda palabra de status de coprocesador
        mov     di,prevdic
        push    es
        push    ds
        pop     es          ;ds=es
        mov     bx,statcon
        push    statcon
        push    bx
        push    bx
        mov     al,bh
        hexasc
        mov     ax,bx
        xchg    al,ah
        stosw
        pop     bx
        mov     al,bl
        hexasc
        mov     ax,bx
        xchg    al,ah
        stosw
        pop     bx          ;bx contiene status
        mov     ax,'0'
        rcr     bx,1
        jnc     copie
        mov     ax,'1'
copie:  stosb               ;ie
        mov     ax,'0'
        rcr     bx,1
        jnc     copde
        mov     ax,'1'
copde:  stosb               ;de
        mov     ax,'0'
        rcr     bx,1
        jnc     copze
        mov     ax,'1'
copze:  stosb               ;ze
        mov     ax,'0'
        rcr     bx,1
        jnc     copoe
        mov     ax,'1'
```

```
copoe:  stosb                   ;oe
        mov     ax,'0'
        rcr     bx,1
        jnc     copue
        mov     ax,'1'
copue:  stosb                   ;ue
        mov     ax,'0'
        rcr     bx,1
        jnc     coppe
        mov     ax,'1'
coppe:  stosb                   ;pe
        mov     ax,'0'
        rcr     bx,2
        jnc     copes
        mov     ax,'1'

copes:  stosb                   ;es
        mov     ax,'0'
        rcr     bx,1
        jnc     copc0
        mov     ax,'1'
copc0:  stosb                   ;c0
        mov     ax,'0'
        rcr     bx,1
        jnc     copc1
        mov     ax,'1'
copc1:  stosb                   ;c1
        mov     ax,'0'
        rcr     bx,1
        jnc     copc2
        mov     ax,'1'
copc2:  stosb                   ;c2
        pop     ax
        rcr     ax,11
        and     ax,7
        cmp     ax,7
        jne     st6?
        mov     al,'7'
        stosb
        jmp     copc31
st6?:   cmp     ax,6
        jne     st5?
        mov     al,'6'
        stosb
        jmp     copc31
st5?:   cmp     ax,5
        jne     st4?
```

```
        mov    al,'5'
        stosb
        jmp    copc31
st4?:   cmp    ax,4
        jne    st3?
        mov    al,'4'
        stosb
        jmp    copc31
st3?:   cmp    ax,3
        jne    st2?
        mov    al,'3'
        stosb
        jmp    copc31
st2?:   cmp    ax,2
        jne    st1?
        mov    al,'2'
        stosb
        jmp    copc31
st1?:   cmp    ax,1
        jne    st0?
        mov    al,'1'
        stosb
        jmp    copc31
st0?:   mov    al,'0'
        stosb
copc31: mov    ax,'0'
        rcr    bx,4
        jnc    copc3
        mov    ax,'1'
copc3:  stosb
        mov    prevdic,di
        pop    es
        retn

SAVECONT:                       ;almaceno palabra de control
        mov    di,prevdic
        push   es
        push   ds
        pop    es               ;ds=cs
        push   contcon
        push   contcon
        mov    bx,contcon
        push   bx
        push   bx
        mov    al,bh
        hexasc
        mov    ax,bx
```

```
        xchg    al,ah
        stosw
        pop     bx
        mov     al,bl
        hexasc
        mov     ax,bx
        xchg    al,ah
        stosw
        pop     bx              ;bx contiene status
        mov     ax,'0'
        rcr     bx,1
        jnc     copim
        mov     ax,'1'
copim:  stosb
        mov     ax,'0'
        rcr     bx,1
        jnc     copdm
        mov     ax,'1'
copdm:  stosb
        mov     ax,'0'
        rcr     bx,1
        jnc     copzm
        mov     ax,'1'
copzm:  stosb
        mov     ax,'0'
        rcr     bx,1
        jnc     copom
        mov     ax,'1'
copom:  stosb
        mov     ax,'0'
        rcr     bx,1
        jnc     copum
        mov     ax,'1'
copum:  stosb
        mov     ax,'0'
        rcr     bx,1
        jnc     coppm
        mov     ax,'1'
coppm:  stosb
        pop     ax
        rcr     ax,8
        and     ax,3
        cmp     ax,3
        jne     pc2?
        mov     ax,'3'
        stosb
        jmp     coprc
```

```
pc2?:  cmp    ax,2
       jne    pc1?
       mov    ax,'2'
       stosb
       jmp    coprc
pc1?:  cmp    ax,1
       jne    pc0?
       mov    ax,'1'
       stosb
       jmp    coprc
pc0?:  mov    ax,'0'
       stosb
coprc: pop    ax
       rcr    ax,10
       and    ax,3
       cmp    ax,3
       jne    rc2?
       mov    ax,'3'
       stosb
       jmp    copic0
rc2?:  cmp    ax,2
       jne    rc1?
       mov    ax,'2'
       stosb
       jmp    copic0
rc1?:  cmp    ax,1
       jne    rc0?
       mov    ax,'1'
       stosb
       jmp    copic0
rc0?:  mov    ax,'0'
       stosb
copic0: rcr   bx,7
       jnc    copic
       mov    ax,'1'
copic: stosb
       mov    prevdic,di
       pop    es
       retn
shell:
       lmppnt
       stodo
       mov    ax,@data
       mov    ds,ax
       mov    bx,ambiente
       mov    pb.ambiente1,bx
       mov    bx,offset parametros1
```

```
        mov     pb.paramaddress,bx
        mov     pb.paramsegmento,ax
        mov     bx,offset fcbloque1
        mov     pb.fcb1offset,bx
        mov     pb.fcb1segmento,ax
        mov     bx,offset fcbloque2
        mov     pb.fcb2offset,bx
        mov     pb.fcb2segmento,ax
        mov     bx,offset pb
        push    ds
        mov     es,ax
        mov     di,pb.fcb1offset
        mov     si,pb.ambiente1
        inc     si
        mov     ax,2901h
        int     21h
        pop     es
        mov     di,pb.fcb2offset
        mov     ax,2901h
        int     21h
        push    bp
        mov     viejosp,sp
        mov     viejoss,ss
        mov     ah,62h          ;obtener dirección de mi PSP
        int     21h
        mov     es,bx
        mov     bx,2000h ;2300h     ;acortar la memoria que utilizo
        mov     ah,4ah
        int     21h
        jnc     simemoria2
        call    menserror
        jmp     biencorrido2
simemoria2:
        push    ds
        pop     es
        mov     bx,offset pb
        mov     dx,offset nombre1
        mov     ax,4b00h
        int     21h
        jnc     biencorrido2
        call    menserror
biencorrido2:
        mov     ss,viejoss
        mov     sp,viejosp
        pop     bp
        mov     visible,0
        mmuestra 1,visible,ratonpresente
```

```
        mov     ah,2
        mov     bh,0
        mov     dh,24
        mov     dl,0
        int     10h
        rtodo
        call    funciones
        call    pdebug
        mov     si,offset l0ini ;preparo y muestro linea 0
        call    mlin0
        jmp     salida
ensambla:
        mmuestra 0,visible,ratonpresente
        lmppnt
        stodo
        mov             ax,@data
        mov     ds,ax

        mov     bx,ambiente
        mov     pb.ambiente1,bx
        mov     pb1.ambiente1,bx

        mov     bx,offset narch1
        mov     pb.paramaddress,bx
        mov     bx,offset ml1
        mov     pb1.paramaddress,bx

        mov     pb.paramsegmento,ax
        mov     pb1.paramsegmento,ax

        mov     bx,offset fcbloque1
        mov     pb.fcb1offset,bx
        mov     bx,offset fcbloque3
        mov     pb1.fcb1offset,bx

        mov     pb.fcb1segmento,ax
        mov     pb1.fcb1segmento,ax

        mov     bx,offset fcbloque2
        mov     pb.fcb2offset,bx
        mov     pb.fcb2segmento,ax
        mov     bx,offset fcbloque4
        mov     pb1.fcb2offset,bx
        mov     pb1.fcb2segmento,ax

        mov     bx,offset pb
        push    ds
```

```
        mov     es,ax
        mov     di,pb.fcb1offset
        mov     si,pb.ambiente1
        inc     si
        mov     ax,2901h
        int     21h
        pop     es
        mov     di,pb.fcb2offset
        mov     ax,2901h
        int     21h

        mov     bx,offset pb1
        push    ds
        pop     es
        mov     di,pb1.fcb1offset
        mov     si,pb1.ambiente1
        inc     si
        mov     ax,2901h
        int     21h
        mov     di,pb1.fcb2offset
        mov     ax,2901h
        int     21h
        push    bp
        mov     viejosp,sp
        mov     viejoss,ss
        mov     ah,62h          ;obtener dirección de mi PSP
        int     21h
        mov     es,bx
        mov     bx,2000h        ;2300h    ;acortar la memoria que utilizo
        mov     ah,4ah
        int     21h
        jnc     simemoriae
        call    menserror
        jmp     biencorridoe

simemoriae:
        mov     dx,offset nombre2
        mov     bx,offset pb
simemoriae1:
        push    ds
        pop     es
        mov     ax,4b00h
        int     21h
        jnc     biencorridoe
        call    menserror
biencorridoe:
        cmp     cl,1bh
```

```
        je      yalink
        mov     ah,9
        mov     dx,offset mensens
        int     21h
        mov     ah,0
        int     16h
        mov     cl,al
        cmp     cl,1bh
        je      yalink
        mov     bx,2000h ;2300h     ;acortar la memoria que utilizo
        mov     ah,4ah
        int     21h
        mov     dx,offset nombre3
        mov     bx,offset pb1
        mov     cl,1bh
        jmp     simemoriae1
yalink:
        mov     ss,viejoss
        mov     sp,viejosp
        pop     bp
        mov     visible,0
        mmuestra 1,visible,ratonpresente
        mov     ah,2
        mov     bh,0
        mov     dh,24
        mov     dl,0
        int     10h
        rtodo
        mov     ah,9
        mov     dx,offset mensens1
        int     21h
        mov     ah,0
        int     16h
        call    funciones
        call    pdebug
        mov     si,offset l0ini ;preparo y muestro linea 0
        call    mlin0
        jmp     salida
redefvect:          ;Redefine vectores 6h y 7h    ;97/02/28
        push    ds
        push    es
        push    ax
        push    bx
        mov     ax,3506h
        int     21h
        mov     old6h,es
        mov     old6h+2,bx
```

```
        mov     ax,3507h
        int     21h
        mov     old7h,es
        mov     old7h+2,bx
        mov     ax,cs
        mov     ds,ax
        mov     dx,offset new6h
        mov     ax,2506h
        int     21h
        mov     dx,offset new7h
        mov     ax,2507h
        int     21h
        pop     bx
        pop     ax
        pop     es
        pop     ds
        retn

new6h   proc    far
                ;nueva rutina para la interrupción 6h
                ;instrucción no válida
        push    ax
        push    ds
        mov     ax,@data
        mov     ds,ax
        mov     error,1
        cmp     errorcj,0ffh
        je      finint1
        pop     ds
        pop     ax
        pop     ax
        inc     ax
        push    ax
        jmp     finint
finint1:
        pop     ds
        pop     ax
        pop     ax
        inc     ax
        push    ax
        int     3
finint:
        iret
        new6h       endp
```

```
new7h  proc   far
                ;nueva rutina para la interrupción 7h
                ;coprocesador no presente
        push   ax
        push   ds
        mov    ax,@data
        mov    ds,ax
        mov    error,2
        cmp    errorcj,0ffh
        je     finint1
        pop    ds
        pop    ax
        pop    ax
        inc    ax
        push   ax
        jmp    finint
        iret
new7h      endp

public  restorvec
restorvec proc  near
        mov    ds,old6h
        mov    dx,old6h+2
        mov    ah,25h
        mov    al,6
        int    21h
        mov    ds,old7h
        mov    dx,old7h+2
        mov    al,7h
        int    21h
        retn
restorvec endp


        end
```

MODULO DEPURA.ASM

```
;Ejecuta porciones de código.

        .model small
        .486
        .data

        public  XferAddr,PointerLst,Vector2F
        public  RegIp,RegCs,RegSp,RegSs,RegBp,flags,Eflags,regFS,regGS
        public  RegSi,RegDs,RegDi,RegEs,RegAx,RegBx,RegCx,RegDx
        public  TablaBrk,FLoadExe,FileSize,FileName
        public  FinNormal,PSPParent

MyLength        equ     3600h

Vector2F        dd      ?
ChildName       db      'Child486.exe',0
FileName        db      256 dup (0)

ParBlock        dw      0
PtrCommLine     dw      offset CommLine
CommSeg         dw      ?
                dw      6 dup (?)
ParBlockNew     dd      ?
CommLine        db      0,0,0
FileSize        dw      0
FilePtr         dw      0
ExeFlag         db      0
FinNormal       db      0
RegIp           dd      ?
RegCs           dw      ?
RegSp           dd      ?
RegSs           dw      ?
RegBp           dd      ?
RegDs           dw      ?
RegEs           dw      ?
RegDi           dd      ?
RegSi           dd      ?
RegDx           dd      ?
RegCx           dd      ?
RegBx           dd      ?
RegAx           dd      ?
flags           dw      ?
```

```
Eflags          dd    ?
regfs           dw    ?
reggs           dw    ?
RegIpInic       dd    ?
RegCsInic       dw    ?
RegSpInic       dd    ?
RegSsInic       dw    ?
RegBpInic       dd    ?
RegDsInic       dw    ?
RegEsInic       dw    ?
RegGsInic       dw    ?
RegFsInic       dw    ?
RegDiInic       dd    ?
RegSiInic       dd    ?
RegDxInic       dd    ?
RegCxInic       dd    ?
RegBxInic       dd    ?
RegAxInic       dd    ?
flagsInic       dw    ?
EflagsInic      dd    ?
ParSp           dw    ?
ParSs           dw    ?
PSPParent       dw    ?
PSPChild        dw    ?
startName       dw    ?
lenghtName      db    ?
NBytesFaltan    dw    ?
FHandle         dw    ?
rwArea          db    512 dup (?)
storeprv        dw    ?
storeprv2       db    256 dup (?) ;zona de preservación de inst. anterior
PrsrvPrg        db    32 dup (?)
PrsrvPsp        db    256 dup (?)
vector          dd    ?     ;area de preservación de vectores de int.
vector2         dd    ?
PointerLst      dd    ?     ;apuntador a la última inst. (en EXHASTA)
PointerNxt      dd    ?
XferAddr        dd    ?     ;dirección de transferencia tras la ejec.
XferProv        dd    ?
nada            db    30 dup (?)
PrCodIns        db    16 dup (?)
TipoCompEC      db    ?
RegCompEC       db    ?
ValCompEC       dd    ?
RetardoTF       db    ?     ;bandera indicadora de si se produce retardo en
                            ;la interrupción de trap flag.
FLoadExe        db    ?
```

```
corre?          db      0       ;indica si se está corriendo hasta detectar
                                ;una condición
TablaBrk        db      4096 dup (?)

        .code
        extrn   desens:proc,assem:proc
loadRegs        proc    near
;---------------------------------------------------------------
;Este procedimiento carga los registros con los valores de memoria.
;---------------------------------------------------------------
        mov     ax,@data
        mov     ds,ax
        pop     ax              ;extraer la dirección de retorno de esta
        mov     word ptr cs:retloadip,ax        ;rutina
        mov     word ptr cs:retloadcs,cs        ;ponerla en el Jmp final
        mov     bx,flags
        mov     Edi,RegIp
        mov     es,RegCs
        mov     al,es:[di]      ;opcode de la instrucción por ejecutar
        cmp     al,0fah         ;CLI?
        jne     NoCli
        and     bx,0fdffh       ;si, borrar I
        jmp     short PonFl
NoCli:  cmp     al,0fbh         ;STI?
        jne     NoSti
        or      bx,200h         ;si, set I
        jmp     short PonFl
NoSti:  mov     Edi,RegSp
        mov     es,RegSs
        cmp     al,9dh          ;POPF?
        jne     NoPopf
        mov     bx,es:[di]      ;si, obtener flags de stack
        and     bx,0feffh       ;borrar T
        jmp     short PonFl
NoPopf: cmp     al,0cfh         ;IRET?
        jne     PonFl
        mov     bx,es:[di+4]    ;si, obtener flags de stack
        and     bx,0feffh       ;borrar T
PonFl:  push    bx
        mov     flags,bx
        popf
        push    eax
        pushfd
      pop       eax
      mov       eflags,eax
        pop     eax
        mov     Eax,regAx
```

```
        mov     ebx,regBx
        mov     ecx,regCx
        mov     edx,regDx
        mov     ebp,regBp
        mov     esp,regSp
        mov     esi,regSi
        mov     edi,regDi
        mov     es,regEs
        mov     ss,regSs
        mov     ds,regDs
        db      0eah            ;JMP FAR
retloadip:
        dw      ?
retloadcs:
        dw      ?
loadRegs        endp

storeRegs       proc    near
;-----------------------------------------------
;Este procedimiento pone en memoria los registros.
;-----------------------------------------------
        pushf
        push    ds
        push    ax              ;(9/12/96)
        mov     ax,@data
        mov     ds,ax
        pop     ax              ;(9/12/96)
        mov     regAx,Eax
        mov     regBx,Ebx
        mov     regCx,Ecx
        mov     regDx,Edx
        mov     regBp,Ebp
        mov     regsp,esp
        push    eax
        xor     eax,eax
        mov     eax,regsp
        add     eax,12
        and     eax,0000ffffh
        push    eax
        mov     eax,regsp
        and     eax,0ffff0000h
        mov     regsp,eax
        pop     eax
        add     eax,regsp
        mov     regsp,eax
        pop     eax
        mov     regFs,fs
```

```
        mov     regGs,gs
        mov     regSi,Esi
        mov     regDi,Edi
        mov     regEs,es
        mov     regSs,ss
        pop     ax
        mov     regDs,ax
        pop     ax              ;AX=flags nuevas
        mov     bx,flags        ;BX=flags anteriores
        and     bx,200h         ;dejar solo la bandera de interrupción
        or      ax,bx           ;dejar el estado anterior de I en flags
        mov     flags,ax
        push    eax
        and     eax,0000ffffh
        and     eflags,0ffff0000h
        or      eflags,eax
        pop     eax
        retn
storeRegs       endp

        public  TestTrap
TestTrap        proc    near
;-------------------------------------------------------------------------
;Este procedimiento determina como se comporta la computadora en cuanto a si
;hay o no retardo en la interrupción 1, que debería producirse en cuanto sea 1
;la trap flag, pero que suele retardarse en una instrucción en ciertos
;microprocesadores. De ser este el caso, RetardoTF es 1, y de lo contrario 0.
;-------------------------------------------------------------------------
        push    ax
        push    bx
        push    dx
        push    ds
        push    es
        ;push   fs
        ;push   gs
        mov     ax,3501h        ;get vector 1
        int     21h
        mov     ax,@data
        mov     ds,ax
        mov     word ptr vector,bx
        mov     word ptr vector+2,es    ;preservar vector original
        mov     ax,cs
        mov     ds,ax
        mov     edx,offset AfterTest
        mov     ax,2501h        ;poner como vector la dirección de AFTERTEST
        int     21h
```

```
        xor     bx,bx           ;BX = 0
        pushf                   ;flags en el stack
        pop     ax              ;flags en AX
        or      ah,1            ;T = 1
        push    ax              ;flags cambiadas de vuelta al stack
        popf
        nop                     ;tras esta instrucción, debe producirse la int
        mov     bx,1            ;si hay retardo, se ejecutará esta instruceión
        nop
        nop
AfterTest:
        add     sp,6            ;restaurar stack, alterado al producirse la int
        mov     ax,@data
        mov     ds,ax
        mov     RetardoTF,0
        cmp     bx,0
        je      YaProbado
        inc     Byte Ptr RetardoTf
YaProbado:
        lds     dx,vector
        mov     ax,2501h        ;restaurar vector 1
        int     21h
        pop     es
        pop     ds
        pop     dx
        pop     bx
        pop     ax
        retn
TestTrap        endp

        public  ExUniq
ExUniq  proc    near
;-------------------------------------------------------------------------
;Este procedimiento coloca el código de SETTRPFLG inmediatamente antes de la
;instrucción que se desea ejecutar (preservando lo que estaba antes), ejecuta
;dicha instrucción, y luego restaura el código. Si la instrucción es una
;interrupción, el procedimiento es diferente: se simula dicha interrupción
;de forma que se recupera el control una vez ejecutada.
;
;Entrada:       RegIp
;               RegCs:          Deben contener la dirección absoluta de la
;                               instrucción que se quiere ejecutar.
;               RetardoTF:      Debe ser 1 si hay que considerar un retardo
;                               en INT 1 tras hacer T=1, 0 en caso contrario.
;               XFERADDR:       Debe contener la dirección absoluta donde debe
;                               transferirse el control una vez finalizado este
;                               procedimiento.
```

```
;
;Salida:        RegIp
;               RegCs:      Contienen la dirección de la siguiente
;                           instrucción, despues de la ejecutada.
;
;IMPORTANTE: Este procedimiento debe invocarse mediante un JMP, y NO
;mediante  un CALL.
;-------------------------------------------------------------------------
        cld
        mov   ax,@data
        mov   es,ax
        mov   ds,ax
        mov   cl,RetardoTF   ;bandera de retardo en CL
        mov   Esi,RegIp
        mov   ds,RegCs       ;DS:SI = apuntador a la instrucción
        mov   ax,ds
        cmp   ax,0a000h      ;ROM BIOS?
        jb    noRB
        mov   ax,@data
        mov   ds,ax
        jmp   dword ptr XferAddr     ;si, no hacer nada
noRB:   mov   al,[si]
        cmp   al,0cdh        ;es una INT ?
        je    SiInt
        jmp   NoInt
SiInt:  mov   al,[si+1]      ;número de la int.
        cmp   al,21h
        jne   NoInt21
SaltInt:jmp   ExPaso         ;siempre saltar la int 21
NoInt21:mov   ah,35h
        int   21h            ;get vector
        mov   dx,es
        cmp   dx,0a000h      ;ROM BIOS?
        jae   SaltInt        ;si, saltar
        mov   dx,@data
        mov   ds,dx
        mov   ParSp,sp
        mov   ParSs,ss
        mov   Esp,RegSp
        mov   ss,RegSs
        push  flags
        popf
        cli
        pushf
        push  RegCs          ;Simular la interrupción
        mov   Edx,RegIp
        add   dx,2
```

```
        push    dx
        mov     RegSp,ESp
        mov     RegSs,Ss
        mov     sp,ParSp
        mov     ss,ParSs
        mov     RegIp,Ebx       ;apuntar a la rutina
        mov     RegCs,es
        jmp     dword ptr xferaddr      ;finalizar

NoInt:
        cmp     al,8ch          ;mov sreg ?
        je      SiMSRg
        cmp     al,8eh
        jne     NoMSrg
SiMSRg: jmp     ExPaso
NoMSrg: cmp     al,0fh
        jne     SiMSRg1
        cmp     [si+1],0a0h
        je      SiMSrg
        cmp     [si+1],0a1h
        je      SiMSrg
        cmp     [si+1],0a8h
        je      SiMSrg
        cmp     [si+1],0a9h
        je      SiMSrg
        jne     nomrg1
SiMSRg1:
        and     al,0e6h
        cmp     al,6            ;push o pop sreg?
        je      SiMSrg
nomrg1:
        push    ds
        dec     si
        dec     si              ;apuntar a los dos bytes anteriores
        mov     dx,si           ;preservar este apuntador
        mov     edi,offset storeprv ;ES:DI apunta a donde se preserva el código
        movsb                   ;preservar código anterior a la instrucción
        movsb
        mov     ax,ds
        mov     es,ax
        mov     di,dx           ;ES:DI apunta a donde se colocara POPF
        mov     ax,909dh        ;código de POPF y NOP
        cmp     cl,0
        jne     AlterCod
        xchg    al,ah           ;si no hay retardo, poner el NOP antes del POPF
AlterCod:
        stosb                   ;colocar el código alterado
```

```
        mov     al,ah
        stosb
        mov     ax,3501h        ;get vector 1
        push    dx
        int     21h
        mov     ax,@data
        mov     ds,ax
        mov     word ptr vector,bx
        mov     word ptr vector+2,es    ;preservar vector original
        mov     ax,cs
        mov     ds,ax
        mov     edx,offset afterex
        mov     ax,2501h        ;poner como vector la dirección de AFTEREX
        int     21h
        pop     bx
        pop     ax
        mov     word ptr cs:dirip,bx
        mov     word ptr cs:dircs,ax    ;dirección del salto
        mov     ax,@data
        mov     ds,ax
        mov     ParSp,sp
        mov     ParSs,ss
        call    loadRegs        ;poner los valores de los registros
        pushf                   ;flags en el stack
        push    ax
        push    bp
        mov     bp,sp           ;apuntar BP al stack
        mov     ax,[bp+4]       ;flags en AX
        or      ah,1            ;T = 1
        mov     [bp+4],ax       ;flags cambiadas de vuelta al stack
        pop     bp
        pop     ax
        db      0eah            ;JMP FAR
dirip:  dw      ?
dircs:  dw      ?

afterex:                        ;aqui se pasa el control luego de ejecutar la

        push    ax              ;instrucción única por efecto de la int.
        push    bp
        push    ds
        mov     ax,@data
        mov     ds,ax
        mov     bp,sp
        mov     ax,[bp+6]       ;obtener la dirección de retorno,
        mov     word ptr pointernxt,ax
        mov     ax,[bp+8]
```

```
        mov     word ptr pointernxt+2,ax        ;preservar esta dirección
        pop     ds
        pop     bp              ;(9/12/96)
        pop     ax
        call    storeRegs       ;salvar registros
        cld             ; hace flag D=0 por lo tanto SI y DI incrementan
                        ;solo se lo usa en cadenas  JORGE(9/12/96)
                        ;no efecto con mi problema (eip=eax en parte alta)
        mov     ax,@data
        mov     ds,ax
        mov     sp,ParSp
        mov     Ss,ParSs
        mov     Edi,RegIp
        mov     es,RegCs        ;ES:DI = apuntador la instrucción ejecutada
        dec     di              ;apuntar al código alterado
        dec     di
        mov     esi,offset storeprv
        movsb                   ;restaurar código
        movsb
        lds     dx,vector
        mov     ax,2501h        ;restaurar vector 1
        int     21h
        mov     ax,@data
        mov     ds,ax
        mov     ax,word ptr pointernxt
        push    ax
        xor     eax,eax
        pop     ax
        mov     RegIp,eax
        mov     ax,word ptr pointernxt+2
        mov     RegCs,ax
        jmp     dword ptr xferaddr      ;finalizar
exuniq  endp

        public  exhasta
exhasta proc    near
;-------------------------------------------------------------------------
;Este procedimiento ejecuta desde la dirección de inicio indicada (inclusive)
;hasta la de finalización (exclusive) indicada.
;
;Entrada:       RegIp
;               RegCs:          Deben contener la dirección absoluta de la
;                               instrucción desde la que se quiere ejecutar.
;               POINTERLST:     Debe contener la dirección absoluta de la
;                               instrucción hasta la que se debe ejecutar.
;               XFERADDR:       Debe contener la dirección absoluta donde debe
;                               transferirse el control una vez finalizado este
```

```
;                       procedimiento.
;
;IMPORTANTE: Este procedimiento debe invocarse mediante un JMP, y NO
;mediante  un CALL.
;-------------------------------------------------------------------------
        cld
        mov     ax,@data
        mov     ds,ax
        mov     ah,62h          ;encontrar la dirección de mi PSP
        int     21h
        mov     PSPChild,bx     ;Preservarla, asumiendo que este es un child
        mov     edx,offset aftergo
        call    SetTerm         ;por si hay un EXIT, fijo dirección de term.
        les     si,pointerIst   ;ES:SI = apuntador a la última instrucción
        mov     ax,es
        cmp     ax,0a000h       ;ROM BIOS?
        jb      noRBh
        jmp     dword ptr XferAddr      ;si, no hacer nada
noRBh:  mov     al,es:[si]      ;preservar lo que hay alli
        mov     byte ptr storeprv,al
        mov     byte ptr es:[si],0cch   ;colocar un breakpoint (INT 3)
        mov     ax,3503h        ;get vector 3
        int     21h
        mov     ax,@data
        mov     ds,ax
        mov     word ptr vector,bx
        mov     word ptr vector+2,es    ;preservar vector original
        mov     ax,cs
        mov     ds,ax
        mov     edx,offset aftergo
        mov     ax,2503h        ;poner como vector 3 la dirección de AFTERGO
        int     21h
        mov     ax,@data
        mov     ds,ax
        mov     Eax,RegIp       ;preparar ejecución
        mov     word ptr cs:goip,ax
        mov     ax,RegCs
        mov     word ptr cs:gocs,ax
        mov     ParSp,sp
        mov     ParSs,ss
        call    loadRegs
        db      0eah            ;JMP FAR
goip:   dw      ?
gocs:   dw      ?

aftergo:                ;aqui se pasa el control luego del breakpoint
        call    storeRegs
```

```
        cld
        mov     ax,@data
        mov     ds,ax
        mov     sp,ParSp
        mov     ss,ParSs
        mov     ah,62h
        int     21h             ;obtengo dirección de PSP actual
        cmp     bx,PSPChild     ;es este un proceso hijo?
        je      EsHijo
        call    SavePspChild    ;no, preservar parámetros del PSP
        call    ReduceLevel     ;bajar de nivel
        mov     ax,@data
        mov     ds,ax
        call    RestPspChild    ;restaurar parámetros del PSP
        mov     FinNormal,1     ;mostrar mensaje de finalización de proceso
        jmp     short RestDir
EsHijo: mov     ax,word ptr pointerlst  ;poner la dirección de la nueva inst.
        push    ax
        xor     eax,eax
        pop     ax
        mov     RegIp,Eax               ;como resultado
        mov     ax,word ptr pointerlst+2
        mov     RegCs,ax
RestDir:                        ;ES:SI = pointer al breakpoint
        les     si,pointerlst
        mov     al,byte ptr storeprv
        mov     es:[si],al      ;restaurar
        lds     dx,vector
        mov     ax,2503h        ;restaurar vector 3
        int     21h
        mov     ax,@data
        mov     ds,ax
        jmp     dword ptr xferaddr      ;finalizar

exhasta endp

        public  expaso
expaso  proc    near
;-------------------------------------------------------------------------
;Este procedimiento ejecuta desde la dirección de inicio indicada (inclusive)
;hasta la siguiente intrucción (exclusive). Es útil para saltar subrutinas.
;
;Entrada:       RegIp
;               RegCs:          Deben contener la dirección absoluta de la
;                               instrucción desde la que se quiere ejecutar.
;               XFERADDR:       Debe contener la dirección absoluta donde debe
;                               transferirse el control una vez finalizado este
```

```
;                       procedimiento.
;
;IMPORTANTE: Este procedimiento debe invocarse mediante un JMP, y NO
;mediante un CALL.
;-------------------------------------------------------------------------
        cld
        mov     ax,@data
        mov     ds,ax
        mov     Esi,RegIp
        mov     ds,RegCs
        lodsb
        cmp     al,0f0h
        jb      noprf
        cmp     al,0f3h
        jbe     getlen
noprf:  dec     si
getlen: mov     ax,@data
        mov     es,ax
        mov     edi,offset nada
        call    desens
        mov     bx,es:[di]          ;obtener primeros caracteres del mnemo
        add     cx,si
        mov     ax,@data
        mov     ds,ax
        mov     word ptr pointerlst,cx
        mov     ax,RegCs
        mov     word ptr pointerlst+2,ax
        cmp     bl,'J'              ;es algun tipo de salto?
        jne     NoJmp               ;no
        jmp     ExUniq              ;si
NoJmp:  cmp     bl,'R'              ;es un RET ?
        jne     NoRet
        cmp     bh,'E'
        jne     NoRet
        cmp     byte ptr es:[di+2],'T'
        jne     NoRet               ;no
        jmp     ExUniq              ;si
NoRet:  cmp     bl,'I'              ;es un IRET ?
        jne     NoIret
        cmp     bh,'R'
        jne     NoIret              ;no
        jmp     ExUniq              ;si
NoIret: cmp     corre?,0ffh         ;si se está corriendo hasta una
        jne     irexhasta           ;condición se debe ingresar a los call
        cmp     bl,'C'              ;es un LOOP ?
        jne     irexhasta
        cmp     bh,'A'
```

```
        jne     irexhasta
        cmp     byte ptr es:[di+2],'L'
        jne     irexhasta
        cmp     byte ptr es:[di+3],'L'
        jne     irexhasta               ;no
        jmp     exuniq
        irexhasta:
        jmp     exhasta
expaso  endp

        public  exCond
exCond  proc    near
;-------------------------------------------------------------------------
;Este procedimiento ejecuta desde la dirección de inicio indicada (inclusive)
;en adelante hasta que se cumpla una condición determinada.
;
;Entrada:       RegIp
;           RegCs:      Deben contener la dirección absoluta de la
;                       instrucción desde la que se quiere ejecutar.
;           AL:         0 si la condición es igualdad de un registro
;                       con un valor
;                       1 si la condición es desigualdad entre un
;                       registro y un valor, en ambos casos:
;                       AH:     xxx0wReg para registros normales
;                               xxx1xxSr para registros de segmento
;                       BX:     valor
;                       2 si la condición es de flags, en este caso:
;                       AH:     xxxxCond
;           XFERADDR:   Dirección de terminación al finalizar
;Salida:        RegIp
;           RegCs:      Apuntan a la próxima intrucción tras cumplirse
;                       la condición
;NOTA: Este procedimiento debe invocarse mediante Jmp, y no Call.
;-------------------------------------------------------------------------
        mov     dx,@data
        mov     ds,dx
        mov     corre?,0ffh
        les     dx,XferAddr     ;preservar el XferAddr
        mov     word ptr XferProv,dx
        mov     word ptr XferProv+2,es
        mov     word ptr XferAddr+2,cs
        cmp     al,2                    ;condición de flag?
        jne     RegCond
        mov     edx,offset FlCondRet    ;si, fijar terminación
        mov     word ptr XferAddr,dx
        and     ah,0fh                  ;generar código de un Jcondition
        or      ah,70h                  ;cambiar el eódigo para que se produzca
```

```
        mov     byte ptr cs:condic,ah   ;un salto en la condición dada
        jmp     ExPaso
RegCond:
        mov     edx,offset RegCondRet
        mov     word ptr XferAddr,dx
        mov     TipoCompEC,al
        mov     RegCompEC,ah
        mov     ValCompEC,Ebx
        jmp     ExPaso
FlCondRet:
        mov     ax,@data
        mov     ds,ax
        cmp     FinNormal,1
        je      GoExitCond
        push    flags
        popf
Condic:
        jo      GoExitCond          ;el código de esta instrucción es
        jmp     ExPaso              ;alterado anteriormente
GoExitCond:
        jmp     ExitCond
RegCondRet:
        mov     ax,@data
        mov     ds,ax
        cmp     FinNormal,1
        je      GoExitCond
        mov     al,RegCompEC
        test    al,80h
        jz      RegNormal
        and     al,7
        cmp     al,0
        jne     RS1EC
        mov     dx,Reges
        jmp     WordReg0
RS1EC:  cmp     al,1
        jne     RS2EC
        mov     dx,Reges
        jmp     WordReg0
RS2EC:  cmp     al,2
        jne     RS3EC
        mov     dx,RegSs
        jmp     WordReg0
RS3EC:  cmp     al,3
        jne     RS4EC
        mov     dx,RegDs
        jmp     WordReg0
RS4EC:  cmp     al,4
```

```
        jne     RS5EC
        mov     dx,RegFs
        jmp     WordReg0
RS5EC:  mov     dx,Reggs

WordReg0:
        mov     ah,RegCompEC
        jmp     WordReg

RegNormal:
        mov     ah,al
        and     al,7
        cmp     al,0
        jne     R1EC
        mov     Edx,RegAx
        jmp     short RdyVal
R1EC:   cmp     al,1
        jne     R2EC
        mov     Edx,RegCx
        jmp     short RdyVal
R2EC:   cmp     al,2
        jne     R3EC
        mov     Edx,RegDx
        jmp     short RdyVal
R3EC:   cmp     al,3
        jne     R4EC
        mov     Edx,RegBx
        jmp     short RdyVal
R4EC:   cmp     al,4
        jne     R5EC
        mov     Edx,RegSp
        jmp     short RdyVal
R5EC:   cmp     al,5
        jne     R6EC
        mov     Edx,RegBp
        jmp     short RdyVal
R6EC:   cmp     al,6
        jne     R7EC
        mov     Edx,RegSi
        jmp     short RdyVal
R7EC:   mov     Edx,RegDi
RdyVal: test    ah,18h
        jnz     WordReg
        and     ah,8
        and     ValCompEC,0ffh          ;recortar a 8 bits el valor de comp.
        and     Edx,0ffh                ;eliminar MSByte del registro
        cmp     al,4
```

```
        jb      WordReg
        jne     S5EC
        mov     dl,byte ptr RegAx+1
        jmp     short WordReg
S5EC:   cmp     al,5
        jne     S6EC
        mov     dl,byte ptr RegCx+1
        jmp     short WordReg
S6EC:   cmp     al,6
        jne     S7EC
        mov     dl,byte ptr RegDx+1
        jmp     short WordReg
S7EC:   mov     dl,byte ptr RegBx+1
        jmp     dwordreg
WordReg:
        test    ah,16
        jnz     dwordreg
        and     edx,0ffffh
        and     valcompec,0ffffh
dwordreg:
        mov     Eax,ValCompEC
        cmp     byte ptr TipoCompEC,1
        je      Desig
        cmp             Eax,Edx             ;comparar el registro con el valor
        je      ExitCond
        jmp     ExPaso              ;ExUniq
Desig:
        cmp     Eax,Edx             ;comparar el registro con el valor
        jne     ExitCond
        jmp     ExPaso              ;ExUniq
ExitCond:
        mov     ax,@data
        mov     ds,ax
        les     dx,XferProv             ;restaurar XferAddr   (9/12/96)
        mov     word ptr XferAddr,dx
        mov     word ptr XferAddr+2,es
        mov     corre?,0
        jmp     dword ptr XferAddr

exCond  endp
```

```
        public  AddBrkPt
AddBrkPt        proc    near
;-------------------------------------------------------------------------
;Este procedimiento añade un breakpoint a la tabla, de no haber existido
;antes uno en la dirección indicada o con el nombre indicado, caso contrario,
;elimina dicho breakpoint para crear uno nuevo.
;
;Entrada:       DS:SI   apunta al nombre del breakpoint (10 caracteres
;                       siempre, forzados final)
;               DX:BX   dirección absoluta del breakpoint
;
;Salida:        CX      número del nuevo breakpoint
;               CY      set si hubo error, excediendo el límite de 255
;-------------------------------------------------------------------------
        xor     cl,cl           ;referencia por dirección a DelBrkPt
BorraB: call    DelBrkPt        ;busco y elimino de la tabla al breakpoint
        jnc     BorraB
        mov     cl,1
BorraN: call    DelBrkPt        ;eliminar también los de nombre igual
        jnc     BorraN
        push    ax
        push    si
        push    es
        push    di
        cld
        mov     ax,@data
        mov     es,ax
        mov     edi,offset TablaBrk
        mov     al,1                    ;número de breakpoint
SrchFt: cmp     byte ptr es:[di],0      ;fin de tabla?
        je      FinTab
        add     di,16
        inc     al
        jz      ErrAdBr         ;si se llega a 256, error
        jmp     short SrchFt
FinTab: stosb                   ;guardar en la tabla el número del brkpt
        mov     ah,al
        mov     al,1
        stosb                   ;marco activo al nuevo breakpoint
        mov     cx,10
        rep     movsb           ;poner el nombre en la tabla
        mov     cl,ah           ;parámetro de retorno
        xor     ch,ch
        mov     ax,bx           ;guardar dirección en tabla
        stosw
        mov     ax,dx
        stosw
```

```
        clc
        jmp     short FinAdBr
ErrAdBr:stc
FinAdBr:pop     di
        pop     es
        pop     si
        pop     ax
        retn
AddBrkPt        endp

        public  DelBrkPt
DelBrkPt        proc    near
;-------------------------------------------------------------------------
;Este Procedimiento elimina un breakpoint de la tabla.
;
;Entrada:       si CL = 0 entonces
;                   DX:BX   dirección absoluta del breakpoint por eliminar
;               si CL <> 0 entonces
;                   DS:SI apunta al nombre del breakpoint por eliminar
;
;Salida:        CY      si el breakpoint no estaba en la tabla
;-------------------------------------------------------------------------
        push    ax
        push    cx
        push    si
        push    di
        push    ds
        push    es
        cld
        mov     ax,@data
        mov     es,ax
        mov     edi,offset TablaBrk
SrchBrk:cmp     byte ptr es:[di],0
        je      FinTabla
        cmp     cl,0            ;referencia por nombre?
        jne     SrchNam         ;si
        cmp     es:[di+12],bx   ;no
        jne     NoEsEste
        cmp     es:[di+14],dx
        jne     NoEsEste
        jmp     Encontrado
SrchNam:push    si
        push    di
        push    cx
        inc     di
        inc     di              ;apunto al nombre en la tabla
        mov     cx,10
```

```
CmpNam: lodsb
        mov     ah,es:[di]
        inc     di
        cmp     ah,al
        jne     FinSrchNam
        loop    CmpNam
FinSrchNam:
        cmp     cx,0
        pop     cx
        pop     di
        pop     si
        je      Encontrado
NoEsEste:
        add     di,16
        jmp     short SrchBrk
FinTabla:
        stc
        jmp     short FinDlBrk
Encontrado:
        mov     si,di
        add     si,16
        mov     ax,es
        mov     ds,ax
MovTab: cmp     byte ptr [si],0
        je      YaBorrado
        lodsw
        dec     al
        stosw                   ;ajustar el número del breakpoint
        mov     cx,7            ;una vez encontrado el brkpt, eliminarlo
        rep     movsw           ;moviendo hacia atrás el resto de la tabla
        jmp     short MovTab
YaBorrado:
        movsb                   ;mover el fin de tabla
        clc
FinDlBrk:
        pop     es
        pop     ds
        pop     di
        pop     si
        pop     cx
        pop     ax
        retn
DelBrkPt        endp
```

```
        public  TogBrkPt
TogBrkPt        proc    near
;-------------------------------------------------------------------------
;Este Procedimiento invierte el estado de un breakpoint de la tabla.
;
;Entrada:       si CL = 0 entonces
;                   DX:BX   dirección absoluta del breakpoint
;              si CL <> 0 entonces
;                   DS:SI apunta al breakpoint
;
;Salida:        CY      si el breakpoint no estaba en la tabla
;-------------------------------------------------------------------------
        push    ax
        push    cx
        push    si
        push    di
        push    es
        cld
        mov     ax,@data
        mov     es,ax
        mov     edi,offset TablaBrk
EncBrk: cmp     byte ptr es:[di],0
        je      FinTablaB
        cmp     cl,0            ;referencia por nombre?
        jne     EncNam          ;si
        cmp     es:[di+12],bx   ;no
        jne     NoEsEsteB
        cmp     es:[di+14],dx
        jne     NoEsEsteB
        jmp     EncontradoB
EncNam: push    si
        push    di
        push    cx
        inc     di
        inc     di              ;apunto al nombre en la tabla
        mov     cx,10
CmpNom: lodsb
        mov     ah,es:[di]
        inc     di
        cmp     ah,al
        jne     FinEncNam
        loop    CmpNom
FinEncNam:
        cmp     cx,0
        pop     cx
        pop     di
        pop     si
```

```
        je      EncontradoB
NoEsEsteB:
        add     di,16
        jmp     short EncBrk
FinTablaB:
        stc
        jmp     short FinHabBrk
EncontradoB:
        xor     byte ptr es:[di+1],1    ;toggle el breakpoint
FinHabBrk:
        pop     es
        pop     di
        pop     si
        pop     cx
        pop     ax
        retn
TogBrkPt        endp

        public  ExTotal
exTotal proc    near
;---------------------------------------------------------------------------
;Este procedimiento ejecuta desde la dirección de inicio indicada (inclusive)
;hasta encontrar alguno de los breakpoints listado en la tabla. Si no hay
;breakpoints, se ejecuta hasta el fin del programa.
;
;Entrada:     RegIp
;             RegCs:        Deben contener la dirección absoluta de la
;                           instrucción desde la que se quiere ejecutar.
;             XFERADDR:     Debe contener la dirección absoluta donde debe
;                           transferirse el control una vez finalizado este
;                           procedimiento.
;             TablaBrk      Tabla de breakpoints. Cada uno es un registro
;                           de 16 bytes, y el final de la tabla es un 0.
;                           La estructura de cada registro es:
;
;                    Offset  Longitud      Contenido
;                      0       1       Numero (1 a 255)
;                      1       1       Estado (1 = hab, 0 = desh.)
;                      2      10       Nombre (forzado a 10)
;                     12       4       Dirección absoluta
;
;IMPORTANTE: Este procedimiento debe invocarse mediante un JMP, y NO
;mediante    un CALL.
;---------------------------------------------------------------------------
        cld
        mov     ax,@data
        mov     ds,ax
```

```
        mov     ah,62h          ;encontrar la dirección de mi PSP
        int     21h
        mov     PSPChild,bx     ;Preservarla, asumiendo que este es un child

        mov     ax,@data
        mov     ds,ax
        les     bx,XferAddr
        mov     word ptr XferProv,bx
        mov     word ptr XferProv+2,es
        mov     dword ptr XferAddr,offset AftPrimInst
        mov     word ptr XferAddr+2,cs

        jmp     ExUniq          ;ejecutar la primera instrucción por si habia
                                ;un breakpoint alli.

AftPrimInst:
        les     bx,XferProv
        mov     word ptr XferAddr,bx
        mov     word ptr XferAddr+2,es
        cmp     FinNormal,1
        jne     NoAcabo
        jmp     dword ptr XferAddr
NoAcabo:mov     edx,offset AfterGoTot
        call    SetTerm         ;por si hay un EXIT, fijo dirección de term.
        mov     esi,offset TablaBrk
        mov     edi,offset StorePrv2
Prsrv:  cmp     byte ptr [si],0
        je      FinPrsrv
        cmp     byte ptr [si+1],0
        je      NxtBrkPt        ;ignorarlo si esta deshabilitado
        mov     bx,[si+12]
        mov     es,[si+14]      ;obtener dirección del BrkPt
        mov     al,es:[bx]      ;preservar lo que hay alli
        mov     [di],al
        mov     byte ptr es:[bx],0cch   ;colocar un breakpoint (INT 3)
        inc     di
NxtBrkPt:
        add     si,16
        jmp     short Prsrv
FinPrsrv:
        mov     ax,3503h        ;get veetor 3
        int     21h
        mov     ax,@data
        mov     ds,ax
        mov     word ptr vector2,bx
        mov     word ptr vector2+2,es   ;preservar vector original
        mov     ax,cs
```

```
        mov     ds,ax
        mov     edx,offset AfterGoTot
        mov     ax,2503h        ;poner como vector 3 la dirección de AFTERGOTOT
        int     21h
        mov     ax,@data
        mov     ds,ax
        mov     Eax,RegIp       ;preparar ejecución
        mov     word ptr cs:goTotIp,ax
        mov     ax,RegCs
        mov     word ptr cs:goTotCs,ax
        mov     ParSp,sp
        mov     ParSs,ss
        call    loadRegs
        db      0eah            ;JMP FAR
GoTotIp:dw      ?
GoTotCs:dw      ?

AfterGoTot:                     ;aqui se pasa el control luego del breakpoint
        push    ax
        push    bp
        push    ds
        mov     ax,@data
        mov     ds,ax
        mov     bp,sp
        mov     ax,[bp+6]               ;obtener la dirección de retorno
        dec     ax                      ;ignorar el breakpoint
        mov     word ptr pointernxt,ax
        mov     ax,[bp+8]
        mov     word ptr pointernxt+2,ax        ;preservar esta dirección
        pop     ds
        pop     bp
        pop     ax
        call    storeRegs
        cld
        mov     ax,@data
        mov     ds,ax
        mov     sp,ParSp
        mov     ss,ParSs
        mov     ah,62h
        int     21h             ;obtengo dirección de PSP actual
        cmp     bx,PSPChild     ;es este un proceso hijo?
        je      EsHijo2
        call    SavePspChild    ;no, preservar parámetros del PSP
        call    ReduceLevel     ;bajar de nivel
        mov     ax,@data
        mov     ds,ax
        call    RestPspChild    ;restaurar parámetros del PSP
```

```
        mov     FinNormal,1     ;mostrar mensaje de fin de proceso
        jmp     short RestBrk
EsHijo2:
        mov     ax,word ptr pointernxt  ;poner la dirección de la nueva inst.
        push    ax
        xor     eax,eax
        pop     ax
        mov     RegIp,Eax               ;como resultado
        mov     ax,word ptr pointernxt+2
        mov     RegCs,ax
RestBrk:mov     edi,offset StorePrv2
        mov     esi,offset TablaBrk
Rstr:   cmp     byte ptr [si],0
        je      FinRstr
        cmp     byte ptr [si+1],0
        je      NxtBrk          ;ignorar el brkpt si esta deshabilitado
        mov     bx,[si+12]
        mov     es,[si+14]
        mov     al,[di]         ;restaurar lo alterado por los breakpoints
        mov     es:[bx],al
        inc     di
NxtBrk: add     si,16
        jmp     short Rstr
FinRstr:
        lds     dx,vector2
        mov     ax,2503h        ;restaurar vector 3
        int     21h

finj:
        mov     ax,@data
        mov     ds,ax

        jmp     dword ptr xferaddr      ;finalizar

exTotal endp

        public ExIns
ExIns   proc    near
;-------------------------------------------------------------------
;Este procedimiento ejecuta una instrucción no perteneciente al código.
;
;Entrada:       DS:SI   apunta a un asciiz que contiene el mnemotécnico
;                       de la instrucción por ejecutar
;               XFERADDR punto de transferencia al finalizar.
;Debe ejecutarse mediante JMP.
;-------------------------------------------------------------------
        push    ds
```

```
        push    si
        mov     ax,@data
        mov     ds,ax
        mov     es,ax
        mov     Esi,RegIp
        mov     ds,RegCs
        mov     edi,offset PrCodIns
        mov     es:[di+10],si
        mov     es:[di+12],ds   ;preservar RegCs,RegIp
        mov     cx,4
        rep     movsw           ;preservar lo actual en RegCs:RegIp
        mov     ax,es
        mov     ds,ax
        mov     Edi,RegIp
        mov     es,RegCs
        pop     si
        pop     ds
        call    assem           ;ensamblar, CX tiene la long. del cod. generado
        jc      FinExIns
        mov     ax,@data
        mov     ds,ax
        les     ax,XferAddr
        mov     word ptr XferProv,ax
        mov     word ptr XferProv+2,es
        mov     dword ptr XferAddr,offset AftExIns
        mov     word ptr XferAddr+2,cs
        push    cx
        jmp     ExUniq
AftExIns:
        pop     cx
        mov     ax,@data
        mov     ds,ax
        les     ax,XferProv
        mov     word ptr XferAddr,ax
        mov     word ptr XferAddr+2,es
        mov     ax,word ptr[PrCodIns+12]
        cmp     ax,RegCs
        jne     RestCod
        mov     ax,word ptr[PrCodIns+10]
        mov     dx,ax
        add     ax,cx
        cmp     Eax,RegIp
        jne     RestCod
        mov     RegIp,Edx
RestCod:mov     esi,offset PrCodIns
        mov     es,word ptr[PrCodIns+12]
        mov     di,word ptr[PrCodIns+10]
```

```
        mov     cx,4
        rep     movsw
        clc
FinExIns:
        jmp     dword ptr xferaddr
ExIns   endp

        public AcortaMiMem
AcortaMiMem     proc    near
;-------------------------------------------------------------------
;Este procedimiento reduce la memoria localizada para el proceso padre a
;MyLenght parrafos, para dejar memoria libre.
;       CY      set si hay error.
;-------------------------------------------------------------------
        mov     ax,@data
        mov     ds,ax
        mov     ah,62h          ;obtener dirección de mi PSP
        int     21h
        mov     PSPParent,bx
        mov     es,bx
        mov     bx,myLength     ;acortar la memoria que utilizo
        mov     ah,4ah
        int     21h
        retn
AcortaMiMem     endp

            public  InstHandler63
InstHandler63   proc    near
;-------------------------------------------------------------------
;Este procedimiento instala el Handler de servicio de la interrupción 63, de
;no estar ya instalado, y ademas contiene el handler.
;-------------------------------------------------------------------
        mov     ax,@data
        mov     ds,ax
        mov     ax,8000h        ;preguntar si ya esta instalado el handler
        mov     ax,3563h        ;no, hacer un getVector para preservar el
        int     21h             ;vector original
        mov     word ptr Vector2F+2,es
        mov     word ptr Vector2F,bx
        mov     dx,offset Handler2F
        mov     ax,cs
        mov     ds,ax
        mov     ax,2563h        ;SetVector de mi rutina
        int     21h
YaIns:  retn

Handler2F:
```

```
        cmp     ah,80h          ;este es el servicio a la interrupción
        je      mio
        push    ax
        push    ds
        mov     ax,@data
        mov     ds,ax
        mov     ax,word ptr Vector2f
        mov     word ptr cs:[Salto+1],ax
        mov     ax,word ptr Vector2f+2
        mov     word ptr cs:[salto+3],ax
        pop     ds
        pop     ax
Salto:  db      0eah
        dd      ?

mio:    cmp     al,0f8h         ;función reservada?
        jb      noReserv
        iret                    ;si, fin
noReserv:
        or      al,al           ;es un request de existencia?
        jnz     noRequest
        mov     al,0ffh         ;si, decir que ya esta instalado
        iret
NoRequest:
        jmp     ChildProc       ;devolver control
InstHandler63   endp

        public  SavePspChild
SavePspChild    proc    near
;-----------------------------------------------------------------------
;Este procedimiento preserva la segunda mitad del PSP del proceso Child.
;-----------------------------------------------------------------------
        push    ax
        push    cx
        push    si
        push    di
        push    ds
        push    es
        mov     si,80h
        mov     edi,offset PrsrvPsp
        mov     ax,ds
        mov     es,ax
        mov     ds,PSPChild
        mov     cx,40h
        rep     movsw
        pop     es
        pop     ds
```

```
        pop     di
        pop     si
        pop     cx
        pop     ax
        retn
SavePspChild    endp

        public  RestPspChild
RestPspChild    proc    near
;-------------------------------------------------------
;Este procedimiento restaura la segunda mitad del PSP del Child.
;-------------------------------------------------------
        mov     esi,offset PrsrvPsp
        mov     es,PSPChild
        mov     di,80h
        mov     cx,40h
        rep     movsw
        retn
RestPspChild    endp

        public reduceLevel
ReduceLevel     proc    near
;-------------------------------------------------------------------
;Este procedimiento ejecuta Child486.exe, que simplemente devuelve al control a
;ChildProc, de forma que se baja el nivel a procedimiento hijo. Restaura los
;registros a sus valores iniciales. Retorna CY=1 si por alguna razon no es
;posible correr Child486.exe.
;-------------------------------------------------------------------
        mov     ax,@data
        mov     ds,ax
        mov     es,ax
        mov     ax,PSPChild
        add     ax,10h
        mov     ds,ax
        xor     si,si
        mov     edi,offset PrsrvPrg
        mov     cx,10h              ;preservar lo que habia antes desde
        rep     movsw               ;PSPChild+10:0, 32 bytes
        mov     ax,es
        mov     ds,ax
        mov     CommSeg,ax
        mov     edx,offset ChildName
        mov     ebx,offset ParBlock
        mov     ax,4b00h
        mov     ParSs,ss
        mov     ParSp,sp        ;preservo Stack
        int     21h             ;EXEC!
```

```
        mov     ax,@data        ;El control no se transfiere si hay error, y
        mov     ds,ax           ;viene aqui, asi que recupero Stack y retorno.
        mov     ss,ParSs        ;Notese que CY es 1 en este punto, por el fallo
        mov     sp,ParSp        ;de la función del DOS.
        retn
ChildProc:
        mov     ax,@data
        mov     ds,ax
        mov     ss,ParSs
        mov     sp,ParSp        ;Recuperar Stack
        mov     ax,PSPChild
        add     ax,10h
        mov     es,ax
        mov     esi,offset PrsrvPrg
        xor     di,di
        mov     cx,10h
        rep     movsw           ;Restaurar lo alterado debido al EXEC
        mov     ah,62h          ;encontrar la dirección del PSP del child
        int     21h             ;process
        mov     PSPChild,bx     ;Preservarla
FinRedLev:
        mov     ax,RegGsInic
        mov     RegGs,ax
        mov     ax,RegFsInic
        mov     Regfs,ax
        mov     Eax,RegIpInic
        mov     RegIp,Eax
        mov     ax,RegCsInic
        mov     RegCs,ax
        mov     Eax,RegSpInic
        mov     RegSp,Eax
        mov     ax,RegSsInic
        mov     RegSs,ax
        mov     Eax,RegBpInic
        mov     RegBp,Eax
        mov     ax,RegDsInic
        mov     RegDs,ax
        mov     ax,RegEsInic
        mov     RegEs,ax
        mov     Eax,RegDiInic
        mov     RegDi,Eax
        mov     Eax,RegSiInic
        mov     RegSi,Eax
        mov     Eax,RegDxInic
        mov     RegDx,Eax
        mov     Eax,RegCxInic
        mov     RegCx,Eax
```

```
        mov     Eax,RegBxInic
        mov     RegBx,Eax
        mov     Eax,RegAxInic
        mov     RegAx,Eax
        mov     ax,flagsInic
        mov     flags,ax
        push    eax
        mov     eax,eflagsInic
        mov     eflags,eax
        pop     eax
        retn
ReduceLevel     endp

        public  SetTerm
SetTerm         proc    near
;-------------------------------------------------------------------
;Este procedimiento fija la dirección de terminación del proceso hijo.
;
;Entrada:       DX = dirección de terminación
;-------------------------------------------------------------------
        mov     ax,PSPChild
        mov     es,ax
        mov     es:[10],dx
        mov     es:[12],cs
        retn
SetTerm         endp


        public LoadExe
LoadExe         proc    near
;-------------------------------------------------------------------
;Este procedimiento carga el archivo cuyo nombre se pasó como parámetro
;a este programa en el PSP del proceso padre, al final del PSP del hijo.
;Luego calcula los valores iniciales de los seudoregistros, y pasa los
;siguientes parámetros al PSP del hijo. CY es 1 si se pasó un nombre de
;archivo incorrecto o si hubo problemas al cargar el archivo.
;Obtiene la posición del Load Module dentro del archivo que se carga.
;
;Ademas se retorna la siguiente información en FLoadExe:
;       0       si es un archivo .exe
;       1       si es un archivo .com
;       2       si es un archivo de datos
;       3       si no se se especificó un nombre de archivo
;       4       si el archivo especificado no existe
;       5       si no se pudo cargar el archivo .exe
;-------------------------------------------------------------------
        mov     ax,@data
```

```
        mov     ds,ax
        mov     ParSp,sp
        mov     ParSs,ss
        mov     ax,PSPParent
        mov     es,ax           ;ES apunta al PSP del padre.
        mov     di,81h          ;voy a buscar el nombre del archivo.
        mov     cx,7fh
        cmp     byte ptr es:[80h],0
        je      noParam
        mov     al,' '
        cld
        repe    scasb           ;encontrar el primer elemento diferente de ' '
        cmp     cx,0
        je      NoParam
        dec     di
        inc     cx
        mov     si,di
        mov     ax,es
        mov     ds,ax           ;DS apunta al PSP del padre
        mov     ax,@data
        mov     es,ax
        mov     edi,offset FileName
XferName:
        lodsb
        cmp     al,' '
        ja      noEndN
        jmp     endNombreArch
noEndN: stosb
        dec     cx
        jz      NotFound
        jmp     short xferName
NotFound:
        mov     byte ptr FLoadExe,4
        jmp     SinProgramaExe
NoParam:
        mov     byte ptr FLoadExe,3
        jmp     SinProgramaExe

LoadNoExe:
        mov     dx,@data
        mov     ds,dx
        mov     ds,PSPParent
        mov     si,cx
        lodsb
        cmp     al,'.'
        jne     NoCom
        lodsb
```

```
        and     al,0dfh
        cmp     al,'C'
        jne     NoCom
        lodsb
        and     al,0dfh
        cmp     al,'O'
        jne     NoCom
        lodsb
        and     al,0dfh
        cmp     al,'M'
        je      SiCom
NoCom:  mov     ds,dx
        mov     byte ptr FLoadExe,2
        jmp     short LoadFile
SiCom:  mov     ds,dx
        mov     byte ptr FLoadExe,1
LoadFile:
        mov     edx,offset FileName
        mov     ax,3d40h
        int     21h             ;abrir archivo
        jc      NotFound
        mov     FHandle,ax
        mov     bx,ax
        mov     dx,100h
        mov     cx,0ffffh
        mov     ds,PSPChild
        mov     ah,3fh
        int     21h             ;leer archivo
        jc      NotFound
        mov     bx,@data
        mov     ds,bx
        mov     FileSize,ax
        mov     bx,FHandle
        mov     ah,3eh
        int     21h             ;cerrar archivo

SinProgramaExe:
        mov     ax,@data
        mov     ds,ax
        mov     ss,ParSs
        mov     sp,ParSp
        mov     ExeFlag,0
        mov     FilePtr,0       ;posicionar pointer al inicio de archivo
        mov     ax,PSPChild
        mov     RegIp,100h
        mov     RegIpInic,100h
        cmp     FLoadExe,2
```

```
        jne     Ip100
        mov     RegIp,0
        mov     RegIpInic,0
        add     ax,10h
Ip100:  mov     RegCs,ax
        mov     RegCsInic,ax
        mov     RegDs,ax
        mov     RegEs,ax
        mov     RegSs,ax
        mov     RegFs,ax
        mov     RegGs,ax
        xor     Eax,Eax
        mov     RegAx,Eax
        mov     RegBx,Eax
        mov     RegCx,Eax
        mov     RegDx,Eax
        mov     RegSi,Eax
        mov     RegDi,Eax
        mov     RegSp,Eax
        mov     RegBp,Eax
        cmp     FLoadExe,3
        jb      NoErrLE
        je      BorraInCod
        stc
        jmp     finload
BorraInCod:
        mov     es,RegCs
        mov     Edi,RegIp
        mov     cx,16
        xor     ax,ax
        rep     stosw           ;borrar inicio del código
NoErrLE:clc
        jmp     finload

endNombreArch:
        mov     dl,ds:[80h]     ;Longitud de parámetros en PSP padre
        xor     dh,dh
        mov     ax,@data
        mov     ds,ax           ;Restauro DS
        inc     cx
        mov     NBytesFaltan,cx
        xor     al,al
        stosb
        dec     si
        mov     startName,si
        sub     si,81h
        sub     dx,si
```

```
        mov     lenghtName,dl
        mov     dx,ds
        mov     ax,PSPChild
        mov     es,ax           ;ES apunta al PSP del hijo
        mov     al,lenghtName
        mov     es:[80h],al
        mov     si,StartName
        mov     di,81h
        mov     cx,NBytesFaltan
        mov     ax,PSPParent
        mov     ds,ax           ;DS apunta al PSP del padre
        rep     movsb
        xor     al,al
Complet:cmp     di,100h
        jae     ListoPSPChild
        stosb
        jmp     short Complet
ListoPSPChild:
        mov     ds,dx
        mov     si,StartName
        sub     si,4            ;apuntar a la extensión del archivo
QueClase:
        mov     ds,PSPParent
        mov     cx,si
        lodsb
        cmp     al,'.'
        je      bienP
        jmp     LoadNoExe
BienP:  lodsb
        and     al,0dfh
        cmp     al,'E'
        je      BienE1
        jmp     LoadNoExe
BienE1: lodsb
        and     al,0dfh
        cmp     al,'X'
        je      BienX
        jmp     LoadNoExe
BienX:  lodsb
        and     al,0dfh
        cmp     al,'E'
        je      BienE2
        jmp     LoadNoExe
BienE2: mov     ds,dx
        mov     edx,offset FileName
        mov     ax,3d40h        ;Abrir archivo
        int     21h
```

```
        jnc     bien3
        jmp     NotFound
bien3:  mov     FHandle,ax
        mov     bx,ax
        mov     edx,offset rwArea
        mov     cx,512
        mov     ah,3fh
        int     21h             ;Leer los primeros 512 bytes del archivo
        jnc     bien4
        jmp     NotFound
bien4:  mov     ah,3eh
        mov     bx,FHandle      ;Cerrar el archivo
        int     21h
        jnc     bien5
        jmp     NotFound
bien5:  mov     esi,offset rwArea
        cmp     word ptr [si],5a4dh
        je      BienL           ;Verificar la marca del Linker
        mov     cx,@data
        mov     ds,cx
        mov     cx,StartName
        sub     cx,4
        jmp     LoadNoExe
BienL:  mov     ax,[si+10h]     ;preservar en memoria los valores de Stack y
        push    ax
        xor     eax,eax
        pop     ax
        mov     RegSp,Eax
        mov     ax,[si+14h]     ;va a controlar
        push    ax
        xor     eax,eax         ;una solucion es "xor   eax,eax"
        pop     ax
        mov     RegIp,EAx
        mov     RegIpInic,Eax
        mov     ax,[si+4]
        mov     cl,8
        shl     ax,cl
        add     ax,[si+2]       ;obtener la longitud del archivo
        mov     FileSize,ax
        mov     ax,[si+8]
        mov     FilePtr,ax      ;posición del Load Module en el archivo
        mov     bx,PSPChild
        add     bx,10h          ;poner al archivo al final del PSP hijo
        mov     word ptr ParBlockNew,bx
        mov     word ptr ParBlockNew+2,bx
        mov     ax,ds
        mov     es,ax
```

```
        mov     edx,offset FileName
        mov     ebx,offset ParBlockNew
        mov     ax,4b03h
        int     21h             ;EXEC, cargar el archivo sin ejecutarlo
        jnc     bien6
        jmp     NotFound
bien6:
        mov     ax,@data
        mov     ds,ax
        mov     ExeFlag,1
        mov     esi,offset rwArea
        mov     ax,word ptr ParBlockNew+2       ;Relocation factor
        mov     bx,[si+16h]
        add      bx,ax
        mov     RegCs,bx        ;Fijar nuevo valor de CS
        mov     RegCsInic,bx
        mov     bx,[si+0eh]
        add      bx,ax
        mov     RegSs,bx        ;Fijar Nuevo valor de SS
        mov     Eax,RegSp
        mov     RegBp,Eax
        mov     ax,PSPChild
        mov     RegEs,ax        ;Crear los seudoregistros
        mov     RegDs,ax
        xor     ax,ax
        mov     RegFs,ax         ;es aqui que se elije con que valor inicien
        mov     RegGs,ax         ;los registros
        xor     Eax,Eax
        mov     RegAx,Eax
        mov     RegBx,Eax
        mov     RegCx,Eax
        mov     RegDx,Eax
        mov     RegSi,Eax
        mov     RegDi,Eax
        mov     FLoadExe,0
        clc
finload:mov     Eax,RegIp
        mov     RegIpInic,Eax
        mov     ax,RegCs
        mov     RegCsInic,ax
        mov     Eax,RegSp
        mov     RegSpInic,Eax
        mov     ax,RegSs
        mov     RegSsInic,ax
        mov     Eax,RegBp
        mov     RegBpInic,Eax
        mov     ax,RegDs
```

```
        mov     RegDsInic,ax
        mov     ax,RegEs
        mov     RegEsInic,ax
        mov     ax,RegFs
        mov     RegFsInic,ax
        mov     ax,RegGs
        mov     RegGsInic,ax
        mov     Eax,RegDi
        mov     RegDiInic,Eax
        mov     Eax,RegSi
        mov     RegSiInic,Eax
        mov     Eax,RegDx
        mov     RegDxInic,Eax
        mov     Eax,RegCx
        mov     RegCxInic,Eax
        mov     Eax,RegBx
        mov     RegBxInic,Eax
        mov     Eax,RegAx
        mov     RegAxInic,Eax
        mov     ax,flags
        mov     flagsInic,ax
        push    eax
        mov     eax,eflags
        mov     eflagsInic,eax
        pop     eax
        retn
LoadExe endp
        end
```

MODULO DESENS.ASM

```
;Programa desensamblador.

        .model  small
        .486
        .data

segreg  db      'E,C,S,D,'
segreg1 db      'F,'
segreg2 db      'G,'

; ----------------------------o-----------------------------------
;
; Mnemotécnicos de las instrucciones del Coprocesador Matemático
;
; ----------------------------o-----------------------------------

oper0F00 db     'SLDT,STR,LLDT,???,VERR,VERW,'
oper0f01 db     'SGDT,SIDT,LGDT,LIDT,SMSW,???,LMSW,INVLPG,'
oper0f02 db     '???,???,LAR,LSL,'
oper0f06 db     'INVD,WBINVD,CLTS,'
oper0f90 db     'SETO,SETNO,SETB,SETNB,SETE,SETNE,SETBE,SETA,SETS,'
         db 'SETNS,SETP,SETNP,SETL,SETNL,SETLE,SETG,'
oper0fa3 db     'BT,BTS,BTR,BTC,'
oper0fa4 db     'SHLD,???,???,???,SHRD,IMUL,'
oper0fb0 db     'XADD,CMPXCHG,'
oper0fb2 db     'LFS,LGS,LSS,BTR,'
oper0fb3 db     'BSF,BSR,'
oper0fb6 db     'MOVZX,MOVSX,'
oper0fc8 db     'BSWAP,'
oper1   db    'ADD,OR,ADC,SBB,AND,SUB,XOR,CMP,'
oper2   db     'DAA,DAS,AAA,AAS,'
oper3   db     'INC,DEC,PUSH,POP,'
oper4   db     'O,NO,C,NC,Z,NZ,BE,A,S,NS,P,NP,L,GE,LE,G,'
oper5   db     'PUSH,POP,SAH,LAH,'
oper6   db     'ROL,ROR,RCL,RCR,SHL,SHR,???,SAR,'
oper7   db     'TEST,???,NOT,NEG,MUL,IMUL,DIV,IDIV,'
oper8   db     'CLC,STC,CLI,STI,CLD,STD,'
oper9   db     'INC,DEC,CALL,CALL FAR,JMP,JMP FAR,PUSH,???,'
operpi  db     'PUSH ,IMUL,'
operpba db      'PUSHA ,POPA ,BOUND,ARPL,'
operpp  db     'PUSH ,POP ,'
```

```
operlx  db    'TEST,XCHG,'
operm?  db    'MOV,???,'
operlea db    'LEA,'
operxax db    'XCHG AX,XCHG EAX,'
opercc  db    'CWD,CBW,CDQ,'
operjc  db    'JMP,CALL,'
operwai db    'WAIT,'
opercm  db    'CMPS,MOVS,'
opersto db    'STOS,'
opersl  db    'SCAS,LODS,'
operm   db    'RETN,'
operf   db    'RETF,'
operll  db    'LDS,LES,'
operi3  db    'INT 03,'
operint db    'INT ,'
operio  db    'INTO,'
operir  db    'IRET,'
operaa  db    'AAD,AAM,'
operxla db    'XLAT,'
operesc db    'ESC ,'
operjl  db    'JCXZ,LOOP,'
operoio db    'OUT ,IN ,OUT DX,'
operios db    'INS,OUTS,'
operlk  db    'LOCK,'
operpr  db    'REPE,REP,REPNE,'
operhlt db    'HLT,'
opercmc db    'CMC,'
opernop db    'NOP,'
operel  db    'ENTER,LEAVE,'

;Registros

reg1    db    'AL,CL,DL,BL,AH,CH,DH,BH,'
reg2    db    'AX,CX,DX,BX,SP,BP,SI,DI,'
reg3    db    'EAX,ECX,EDX,EBX,ESP,EBP,ESI,EDI,'
regtr   db    '???,???,???,TR3,TR4,TR5,TR6,TR7,'
regdr   db    'DR0,DR1,DR2,DR3,???,???,DR6,DR7,'
regcr   db    'CR0,???,CR2,CR3,???,???,???,???,'
modos   db    'SI+BX,DI+BX,SI+BP,DI+BP,SI,DI,BP,BX,'
escala  db    '+1*,+2*,+4*,+8*,'

; ------------------------------o-------------------------------------
;
; Mnemotécnicos de las instrucciones del Coprocesador Matemático
;
; ------------------------------o-------------------------------------
op000a  db    'FADD,FMUL,FCOM,FCOMP,FSUB,FSUBR,FDIV,FDIVR,'
```

```
op000b  db    'FADD st,FMUL st,FCOM,FCOMP,FSUBR st,FSUB st,'
        db    'FDIVR st,FDIV st,'
op001a  db    'FLD,FXCH st,FST,FSTP,FLDENV,FLDCW,FSTENV,FSTCW,'
op001b  db   'FNOP,FABS, , ,FTST,FXAM, , ,FLD1,FLDL2T,FLDL2E,FLDPI,'
        db    'FLDLG2,FLDLN2,FLDZ,'
op001c  db    'F2XM1,FYL2X,FPTAN,FPATAN,FXTRACT,FPREM1,'
        db    'FDECSTP,FINCSTP 'FPREM,FXL2XP1,FSQRT,'
        db    'FSINCOS,FRNDINT,FSCALE,FSIN,FCOS,'
op010a  db   'FIADD,FIMUL,FICOM,FICOMP,FISUB,FISUBR,FIDIV,FIDIVR,'
op010b  db    'FUCOMPP,'
op011a  db    'FILD, ,FIST,FISTP, ,FLD, ,FSTP,'
op011b  db    'FENI,FDISI,FCLEX,FINIT,FSETPM,'
op101a  db    'FLD, ,FST,FSTP,FRSTOR, ,FSAVE,FSTSW,'
op101b  db    'FFREE, ,FST,FSTP,FUCOM st,FUCOMP st,'
op110b  db    'FADDP,FMULP, ,FCOMPP,FSUBRP,FSUBP,'
        db    'FDIVRP,FDIVP,'
op111a  db    'FILD, ,FIST,FISTP,FBLD,FILD,FBSTP,FISTP,'

;Regsitros del coprocesador

regcop  db    'st(0),st(1),st(2),st(3),st(4),st(5),st(6),st(7),'


casos   db    40h,60h,70h,80h,84h,88h,8ch,90h
        db    91h,98h,9ah,9bh,9ch,0a0h,0a4h,0a8h
        db    0aah,0ach,0b0h,0c0h,0c2h,0c4h,0c6h,0c8h
        db    0cah,0cch,0cdh,0ceh,0cfh,0d0h,0d4h,0d6h
        db    0d7h,0d8h,0e0h,0e4h,0e8h,0eah,0ebh,0ech
        db    0f0h,0f1h,0f2h,0f4h,0f5h,0f6h,0f8h,0feh

;Direcciones de la rutina de cada caso

direcc  dw    offset case0to3f,       offset case40to5f
        dw    offset case60to6f,      offset case70to7f
        dw    offset case80to83,      offset case84to87
        dw    offset case88to8b,      offset case8cto8f
        dw    offset case90,           offset case91to97
        dw    offset case98to99,      offset case9a
        dw    offset case9b,          offset case9cto9f
        dw    offset casea0toa3,      offset casea4toa7
        dw    offset casea8toa9,      offset caseaatoab
        dw    offset caseactoaf,      offset caseb0tobf
        dw    offset casec0toc1,      offset casec2toc3
        dw    offset casec4toc5,      offset casec6toc7
        dw    offset casec8toc9,      offset casecatocb
        dw    offset casecc,          offset casecd
        dw    offset casece,          offset casecf
```

```
        dw      offset cased0tod3,      offset cased4tod5
        dw      offset cased6,          offset cased7
        dw      offset cased8todf,      offset casee0toe3
        dw      offset casee4toe7,      offset casee8toe9
        dw      offset caseea,          offset caseeb
        dw      offset caseectoef,      offset casef0
        dw      offset casef1,          offset casef2tof3
        dw      offset casef4,          offset casef5
        dw      offset casef6tof7,      offset casef8tofd

;Variables generales

memop   db      0  ;indica si se direcciona memoria
sib     dw      0  ;indica si hay el bite s-i-b para direccionamiento escalonado
l1      dw      0  ;indica si hay redefinición de segmento
l2      dw      0  ;indica si son datos de 32 bits
l3      dw      0  ;indica si es direccionamiento de 32 bits
l33     dw      0
add1    dw      ?
mne     db      ?  ;almacena C,E,D o S según la redefinicón del segmento
opcode  db      14 dup (?) ;contiene 14 bytes del código a desensamblar
astring dw      ?
bstring dw      ?
atastr  db      20 dup (?) ;almacena temporalmente el operando
                           ;correspondiente a REG
atbstr  db      20 dup (?) ;almacena temporalmente el operando
                           ;correspondiente a MOD y R/M

        .code

readarray       proc    near
;-------------------------------------------------------------------------
;Este procedimiento lee un elemento de un arreglo alfanumérico, de dirección
;especificada por DS:SI. El elemento se coloca desde ES:DI, y DI se acomoda
;para que apunte a la dirección siguiente al ultimo carácter del elemento.
;Los separadores entre elementos deben ser comas (,).
;
;Entrada:       CX = número del elemento en el arreglo.
;
;Salida:        DI = dirección del ultimo carácter transferido + 1
;-------------------------------------------------------------------------
        push    ax
        push    cx
        push    si
        or      cx,cx           ;primer elemento del arreglo?
        jz      encont          ;si
busccoma:                       ;buscar el elemento
```

```
        inc     si
        cmp     byte ptr[si],',' ;separador?
        jne     busccoma        ;no
        loop    busccoma        ;si, decrementar número de elemento
        inc     si              ;apuntar al elemento encontrado
encont:
        lodsb                   ;carácter en AL
        cmp     al,','          ;separador?
        je      finread         ;si, finalizar transferencia
        stosb                   ;no, transferir
        jmp     short encont
finread:
        pop     si
        pop     cx
        pop     ax
        retn
readarray       endp

        public convhex
convhex proc    near
;-----------------------------------------------------------------
;Este procedimiento genera un string de 2 caracteres desde ES:DI que
;representan hexadecimalmente al número en AL. DI se acomoda como en el
;procedimiento readarray.
;
;Entrada:       AL = número por convertir.
;
;Salida:        DI = dirección del ultimo carácter generado + 1
;-----------------------------------------------------------------
        push    ax
        push    cx
        mov     ah,al
        and     ah,0fh          ;4 lsb en AH
        mov     cl,4
        shr     al,cl           ;4 msb en AL
        or      ax,3030h        ;conv. a ascii
        cmp     al,'9'
        jbe     numal
        add     al,'A'-'9'-1
numal:  cmp     ah,'9'
        jbe     numah
        add     ah,'A'-'9'-1
numah:  stosw
        pop     cx
        pop     ax
        retn
convhex endp
```

```
conv2hex        proc    near
;-------------------------------------------------------------------
;Este procedimiento genera un string de 4 u 8 caracteres desde ES:DI que
;representan hexadecimalmente a la palabra apuntada por DS: SI+BX. DI se
;acomoda como en readarray.
;
;Salida:        DI = dirección del ultimo carácter generado + 1
;
;Usa:           convhex
;-------------------------------------------------------------------
        push    cx
        mov     cx,1
        cmp     12,0            ;es doble palabra
        je      OtraPalabra     ;no
        add     bx,2
        shl     cx,1
OtraPalabra:
        push    ax
        mov     ax,[si+bx]
primerapalabra:
        push    ax
        mov     al,ah
        call    convhex
        pop     ax
        call    convhex
        pop     ax
        sub     bx,2
        loop    otrapalabra     ;lazo a convertir la segunda palabra
        add     bx,2
        pop     cx
        retn
conv2hex        endp

cod3bit proc    near
;-------------------------------------------------------------------
;Este procedimiento obtiene el número de 3 bits de los bits 3, 4 y 5 de
;BL, y lo coloca en BL. Es de utilidad para, por ejemplo, obtener reg
;en el byte de modo de direccionamiento.
;
;Entrada:       BL
;
;Salida:        BL = (BL and 38H)/8
;-------------------------------------------------------------------
        push    cx
        and     bl,38h
        mov     cl,3
        shr     bl,cl
```

```
        pop     cx
        retn
cod3bit endp

copmem  proc    near
;---------------------------------------------------------------------
;Este procedimiento genera el string de modo de direccionamiento [...]
;desde  ES:DI para las instrucciones del coprocesador
;
;Entrada:       AL = tipo de operador (Q,B,D,Q,T)
;
;Salida:        CX = longitud presumida de instrucción
;              memop = 1 si se opera con memoria
;
;Usa:           modrm,addbstr
;---------------------------------------------------------------------
        push    bx
        push    ax
        mov     al,' '
        stosb
        mov     bx,offset atbstr
        mov     bstring,bx
        call    modrm           ;generar [...]
        call    addbstr         ;poner [...]
        pop     ax
        pop     bx
        retn

copmem  endp

copreg  proc    near
;---------------------------------------------------------------------
;Este procedimiento genera el string de registro del coprocesador
;desde ES:DI
;
;Entrada:       si = apunta al inicio del código de la instrucción
;
;Salida:        cx = longitud presumida de instrucción
;              al = tipo de operador
;
;Usa:           readarray

;---------------------------------------------------------------------
        push    bx
        mov     bx,l1
        add     bx,l2
        add     bx,l3
```

```
        mov     cl,[si+bx+1]  ;cargar la parte del código que indica
                                ;el registro
        and     cx,7
        push    si
        mov     si,offset regcop
        call    readarray       ;poner el registro
        pop     si
        mov     al,4
        mov     cx,2            ;longitud presumida de la
                                        ;instrucción
        pop     bx
        retn
copreg  endp


modrm   proc    near
;---------------------------------------------------------------------
;Este procedimiento genera el string de modo de direccionamiento [...]
;desde DS:bstring, y lo termina con 0.
;
;Entrada:       AL = tipo de operador (w)
;
;Salida:        CX = longitud presumida de instrucción
;               memop = 1 si se opera con memoria
;
;Usa:           convhex, conv2hex, readarray
;---------------------------------------------------------------------
        push    dx
        push    bx
        push    ax
        push    di
        push    es
        push    ds
        pop     es
        mov     di,bstring
        mov     bx,l1
        add     bx,l2
        add     bx,l3
        mov     dh,[si+bx+2]    ;segundo byte del opcode
        mov     bl,[si+bx+1]    ;primer byte del opcode
        mov     bh,bl
        and     bh,0c0h
        mov     cl,6
        shr     bh,cl           ;2 msb de dicho byte an bh (mod)
        mov     dl,bl
        and     dl,7            ;3 lsb de dicho byte en dl (r/m)
        cmp     l3,1
```

```
        je      Direcc32Bit
        cmp     dl,6
        jne     nodir           ;si no fuera modo directo
        or      bh,bh
        jnz     nodir           ;idem

        mov     memop,1         ;modo directo: mod=0, r/m=6
        mov     al,'['
        stosb                   ;poner la dirección entre corchetes
        mov     bx,l1           ;y terminar string con 0.
        add     bx,l2
        add     bx,l3
        add     bx,2
        push    l2
        mov     l2,0
        call    conv2hex
        pop     l2
        mov     al,']'
        xor     ah,ah
        stosw
        mov     cx,4            ;long. 4 bytes
        add     cx,l2
        jmp     finmodrm

nodir:  cmp     bh,3            ;registro a registro?
        jne     no3             ;no
        mov     cl,dl           ;si,leer de la tabla de registros
        xor     ch,ch
        push    si
        mov     si,offset reg1  ;poner nombre de registro en string
        or      al,al           ;instrucción de word?
        jz      primfila        ;no
        cmp     l33,1
        je      modrmex
        cmp     l2,0            ;es doble word?
        je      modrmex         ;si
        mov     si,offset reg3
        jmp     short  primfila
modrmex:
        mov     si,offset reg2  ;si
primfila:
        call    readarray
        xor     al,al
        stosb                   ;terminar con 0
        mov     cx,2
        add     cx,l2
        pop     si
```

```
        jmp     finmodrm
no3:    mov     memop,1         ;operación con memoria
        push    bx
        mov     al,'['
        stosb
        push    si
        mov     si,offset modos
        mov     cl,dl           ;leer un modo de direc. de la tabla
        xor     ch,ch
        call    readarray
        pop     si
        cmp     bh,1            ;mod=1?
        jne     mdno1           ;no
        mov     bx,l1           ;si, offset de un byte
        add     bx,l2
        add     bx,l3
        add     bx,2
        mov     bl,[si+bx]      ;obtener offset
        cmp     bl,127          ;hallar su signo
        jbe     positoffs
        mov     al,'-'
        neg     bl
        jmp     short addsgn
positoffs:
        mov     al,'+'
addsgn: stosb
        mov     al,bl
        call    convhex         ;incluir offset en el string
        jmp     short endselmodrm

mdno1:  cmp     bh,2            ;hay offset?
        jne     short endselmodrm
        mov     al,'+'          ;si, de 2 bytes
        stosb
        mov     bx,l1
        add     bx,l2
        add     bx,l3
        add     bx,2
        push    l2
        mov     l2,0
        call    conv2hex
        pop     l2
endselmodrm:
        mov     al,']'
        xor     ah,ah
        stosw                   ;acabar con 0
        pop     bx
```

```
        mov     cl,bh
        xor     ch,ch
        add     cx,2            ;long. = mod + 2
        add     cx,l2
        jmp     finmodrm

Direcc32Bit:                    ;Modo de direccionamiento de 32 Bits
        cmp     dl,5
        jne     NoDirecto       ;si no fuera modo directo
        or      bh,bh
        jnz     NoDirecto
        mov     memop,1         ;modo directo: mod=0, r/m=5
        mov     al,'['
        stosb                   ;poner la dirección entre corchetes
        mov     bx,l1           ;y terminar string con 0.
        add     bx,l2
        add     bx,l3
        add     bx,4
        push    l2
        mov     l2,0
        call    conv2hex
        sub     bx,2
        call    conv2hex
        pop     l2
        mov     al,']'
        xor     ah,ah
        stosw
        mov     cx,6            ;long. 6 bytes
        add     cx,l2           ;sumar uno debido al prefijo de doble palabra
        jmp     finmodrm

NoDirecto:
        cmp     bh,3            ;registro a registro?
        jne     NoRegaReg       ;no
        mov     cl,dl           ;si,leer de la tabla de registros
        xor     ch,ch
        push    si
        mov     si,offset reg1  ;poner nombre de registro en string
        or      al,al           ;instrucción de word?
        jz      NoInstWord      ;no
        cmp     l33,1
        je      NoModRmEx
        cmp     l2,0
        je      NoModRmEx
        mov     si,offset reg3
        jmp     short NoInstWord
NoModRmEx:
```

```
        mov     si,offset reg2     ;si
NoInstWord:
        call    readarray
        xor     al,al
        stosb                   ;terminar con 0
        mov     cx,2
        add     cx,l2
        pop     si
        jmp     finmodrm
NoRegaReg:
        mov     memop,1         ;operación con memoria
        push    bx
        mov     al,'['
        stosb
        mov     sib,0
        cmp     dl,4
        jne     NoHaySib
        mov     sib,1
        mov     dl,dh       ;Si hay S-i-b
        mov     bl,dl
        call    cod3bit
        and     dl,7
        and     dh,0c0h
        mov     cl,6
        shr     dh,cl
        push    si
        mov     si,offset reg3
        mov     cl,dl
        xor     ch,ch
        call    readarray
        cmp     dh,0            ;Si indice es 4 y escala no es 0 entonces
        je      NoIndef          ;la dirección efectiva queda indefinida
        cmp     bl,4
        jne     NoIndef
        mov     si,offset operm?
        mov     cx,1
        call    readarray
        jmp     short VerDesp
NoIndef:
        mov     si,offset escala
        mov     cl,dh
        call    readarray
        mov     si,offset reg3
        mov     cl,bl
        call    readarray
        pop     si
        jmp     short VerDesp
```

```
NoHaySib:
        push    si
        mov     si,offset reg3
        mov     cl,dl           ;leer un modo de direc. de la tabla
        xor     ch,ch
        call    readarray
        pop     si
VerDesp:
        cmp     bh,1            ;mod=1?
        jne     mdnoes1         ;no
        mov     bx,l1           ;si, offset de un byte
        add     bx,l2
        add     bx,l3
        add     bx,sib
        add     bx,2
        mov     bl,[si+bx]      ;obtener offset
        cmp     bl,127          ;hallar su signo
        jbe     positoffs1
        mov     al,'-'
        neg     bl
        jmp     short addsigno
positoffs1:
        mov     al,'+'
addsigno: stosb
        mov     al,bl
        call    convhex         ;incluir offset en el string
        jmp     endselmodrm1

mdnoes1:
        cmp     bh,2            ;hay offset?
        jne     endselmodrm
        mov     al,'+'          ;si, de 2 bytes
        stosb
        mov     bx,l1
        add     bx,l2
        add     bx,l3
        add     bx,sib
        add     bx,4
        push    l2
        mov     l2,0
        call    conv2hex
        sub     bx,2
        call    conv2hex
        pop     l2
endselmodrm1:
        mov     al,']'
        xor     ah,ah
```

```
        stosw               ;acabar con 0
        pop     bx
        mov     cl,bh
        xor     ch,ch
        add     cx,l2
        add     cx,2            ;long. = mod + 2
        add     cx,sib
        cmp     bh,2
        jne     finmodrm
        add     cx,2
finmodrm:
        pop     es
        pop     di
        pop     ax
        pop     bx
        pop     dx
        retn
modrm   endp



modregrm        proc    near
;-----------------------------------------------------------------
;Este procedimiento genera el string de modo de direccionamiento y su
;registro operador en el orden apropiado a partir de ES:DI y acomoda DI
;como en readarray. Utiliza DS:astring y DS:bstring
;
;Entrada:       AH = dw (par de bits de dirección y tipo de operador)
;
;Salida:        CX = longitud de instrucción
;               AL = w (tipo de operador)
;               memop = 1 si se ha operado con memoria.
;
;Usa:           modrm, cod3bit, readarray, order
;-----------------------------------------------------------------
        push    bx
        push    dx
        push    si
        push    es
        push    ds
        pop     cs
        mov     al,ah
        and     al,1            ;obtener w
        call    modrm           ;obtener [...]
        call    cod3l1inc       ;obtener cod del byte de modo de dir.
        mov     si,offset reg1  ;leer reg de la tabla segun w
        or      al,al
```

```
        jz      bringreg
        cmp     l2,0
        je      modregrmex
        mov     si,offset reg3
        jmp     short   bringreg
modregrmex:     mov     si,offset reg2
bringreg:
        push    cx
        push    di
        mov     di,astring
        mov     cl,bl
        xor     ch,ch
        call    readarray
        mov     dh,ah           ;obtener d en DH
        and     dh,2
        pop     di
        pop     cx
        pop     es
        call    order           ;poner en orden [...] y su operador
        add     ax,l2
        pop     si
        pop     dx
        pop     bx
        retn
modregrm        endp

detectmem proc near
;-----------------------------------------------------------------
;Este procedimiento extrae los bits 6 y 7 del segundo byte del opcode
;sirve para las instrucciones del coprocesador
;
;Salida:
;           dl = ((2do byte del opcode) and 0Ch)/64
;
;-----------------------------------------------------------------
        push    cx
        push    bx
        mov     bx,l1
        add     bx,l2
        add     bx,l3
        mov     dl,[si+bx+1]    ;bl = segundo byet del opcode
        mov     cl,6
        shr     dl,cl           ;dividir para 64
        pop     bx
        pop     cx
        retn
detectmem endp
```

```
modregrm1       proc    near
;-----------------------------------------------------------------
;Este procedimiento genera el string de modo de direccionamiento y su
;registro operador en el orden apropiado a partir de ES:DI y acomoda DI
;como en readarray. Utiliza DS:astring y DS:bstring
;
;Entrada:       AH = dw (par de bits de dirección y tipo de operador)
;
;Salida:        CX = longitud de instrucción
;               AL = w (tipo de operador)
;               memop = 1 si se ha operado con memoria.
;
;Usa:           modrm, cod3bit, readarray, order
;-----------------------------------------------------------------

        push    bx
        push    dx
        push    si
        push    es
        push    ds
        pop     cs
        mov     al,ah
        and     al,1            ;obtener w
        push    ax
        mov     l33,1
        call    modrm           ;obtener [...]
        mov     l33,0
        call    cod311inc       ;obtener cod del byte de modo de dir.
        mov     al,1
        mov     si,offset reg1  ;leer reg de la tabla segun w
        mov     al,1
        or      al,al
        pop     ax
        jz      bringreg1
        cmp     l2,0
        je      modregrmex1
        mov     si,offset reg3
        jmp     short   bringreg1

modregrmex1:
        mov     si,offset reg2
bringreg1:
        push    cx
        push    di
        mov     di,astring
        mov     cl,bl
```

```
        xor     ch,ch
        call    readarray
        mov     dh,ah           ;obtener d en DH
        and     dh,2
        pop     di
        pop     cx
        pop     cs
        call    order           ;poner en orden [...] y su operador
        pop     si
        pop     dx
        pop     bx
        retn
modregrm1       endp


order   proc    near
;-------------------------------------------------------------------------
;Este procedimiento establece el orden en que deben ir astring y bstring en
;el nemotécnico (a partir de ES:DI) segun el bit de dirección (DH) y los
;coloca, acomodando despues DI como en readarray.
;
;Entrada:       DH = bit de dirección
;
;Usa:    addastr, addbstr
;-------------------------------------------------------------------------
        push    ax
        mov     al,' '
        stosb                   ;poner un espacio
        or      dh,dh
        jz      bfirst
        call    addastr         ;agregar el string almacenado en atastr
        mov     al,','
        stosb
        call    addbstr         ;agregar el string almacenado en atbstr
        jmp     short finorder
bfirst: call    addbstr
        mov     al,','
        stosb
        call    addastr
finorder:
        pop     ax
        retn
order   endp

addstr  proc    near
;-------------------------------------------------------------------
;Este procedimiento coloca un string apuntado por DS:SI terminado en 0
```

```
;en ES:DI, acomodando DI como en readarray.
;-----------------------------------------------------------------
        push    si
        push    ax
xfer:   lodsb
        push    si
        push    di
        dec     si
        mov     di,si
        push    ax
        mov     al,0
        stosb
        pop     ax
        pop     di
        pop     si
        or      al,al           ;fin de string?
        jz      finaddstr       ;si
        stosb                   ;no, continuar
        jmp     short xfer
finaddstr:
        pop     ax
        pop     si
        retn
addstr  endp

;-----------------------------------------------------------------
;Los siguientes dos procedimientos colocan los strings apuntados por
;astring o bstring desde ES:DI, acomodando DI como en readarray.
;
;Usan:  addstr
;-----------------------------------------------------------------
addastr proc    near
        push    si
        mov     si,astring      ;apuntar a atastr
        call    addstr
        pop     si
        retn
addastr endp

addbstr proc    near
        push    si
        mov     si,bstring      ;apuntar a atbstr
        call    addstr
        pop     si
        retn
addbstr endp
```

```
accdata proc    near
;-------------------------------------------------------------------
;Este procedimiento genera un string de operación de acumulador con datos
;inmediatos, y lo coloca a partir de ES:DI, acomodando DI como en readarray.
;
;Entrada:       AL = w (bit de tipo de operador)
;
;Salida:        CX = longitud de instrucción
;
;Usa:           readarray, convhex, conv2hex
;-------------------------------------------------------------------
        push    bx
        push    dx
        push    si
        mov     dl,al           ;preservar w
        mov     al,' '
        stosb                   ;poner un espacio
        xor     cx,cx           ;leer string de acumulador (AL o AX)
        mov     si,offset reg1
        or      dl,dl           ;segun w
        jz      data1
        cmp     l2,0            ;es EAX
        je      accex           ;no
        mov     si,offset reg3  ;si
        jmp     short data1
acccx:  mov     si,offset reg2
data1:  call    readarray
        pop     si
        mov     al,','
        stosb                   ;poner una coma despues del acumulador
        mov     bx,l1
        add     bx,l2
        add     bx,l3
        inc     bx              ;SI+BX apunta al dato inmediato
        or      dl,dl           ;que es de 1 o 2 bytes segun w
        jz      dat1
        call    conv2hex
        jmp     short finaccdata
dat1:   mov     al,[si+bx]
        call    convhex
finaccdata:
        xor     cx,cx
        mov     al,dl   ;recuperar w
        cmp     l2,0
        je      longacc1
        mov     cl,2
        mov     al,2
```

```
longacc1:
        add     cl,2
        add     cl,al           ;longitud de instrucción
        pop     dx
        pop     bx
        retn
accdata endp

realadr proc    near
;---------------------------------------------------------------
;Este procedimiento calcula la dirección real de un salto relativo y la
;convierte en un string a partir de ES:DI, con DI come en readarray.
;
;Entrada:       AL = w
;               CX = longitud de instrucción
;
;Usa:           convhex
;---------------------------------------------------------------
        push    ax
        push    bx
        mov     bx,l1           ;apuntar al opcode
        or      al,al           ;desplazamiento de un byte?
        mov     al,[si+bx+1]
        cbw
        jz      dsp1b           ;si
        mov     ah,[si+bx+2]    ;no
dsp1b:  add     ax,add1
        add     ax,cx           ;dirección real en ax
        push    ax
        mov     al,' '
        stosb                   ;poner un espacio
        mov     al,ah           ;poner el string
        call    convhex
        pop     ax
        call    convhex
        pop     bx
        pop     ax
        retn
realadr endp

wob     proc    near
;---------------------------------------------------------------
;Este procedimiento coloca "W" o "B" en ES:DI e incrementa DI, segun w. Es
;util para completar instrucciones de string.
;
;Entrada:       AH = opcode
;
```

```
;Salida:      AL = w
;            CX = longitud de instrucción
;-------------------------------------------------------------------------
        mov   al,ah
        and   al,1          ;AL = w
        push  ax
        jnz   instw
        mov   al,'B'        ;Byte
        jmp   short adcar
instw:  mov   al,'W'        ;palabra
        cmp   l2,0
        je    adcar
        mov   al,'D'        ;Doble palabra
adcar:  stosb
        mov   cx,1          ;long = 1
        pop   ax
        add   ax,l2
        add   cx,l2
        retn
wob     endp

datadic proc   near
;-------------------------------------------------------------------------
;Este procedimiento genera el string de modo de direccionamiento [...] y
;un operador inmediato desde ES:DI. Acomoda DI como readarray .
;
;Entrada:     AH = opcode
;            DL = sw (bits de extension de signo y tipo de operador)
;
;Salida:      AL = w
;            CX = longitud de instrucción
;
;Usa:         modrm, addbstr, convhex
;-------------------------------------------------------------------------
        push   bx
        mov    al,ah
        and    al,1          ;obtener w
        mov    bx,offset atbstr
        mov    bstring,bx
        call   modrm         ;generar [...]
        push   ax
        mov    al,' '
        stosb                ;poner un espacio
        call   addbstr       ;poner [...]
        mov    al,','        ;poner una coma
        stosb
        mov    bx,l1         ;apuntar al opcode
```

```
        add     bx,l3
        add     bx,cx           ;apuntar a los datos
        mov     ax,[si+bx]
        cmp     l2,1            ;es doble palabra?
        jne     sinext          ;no
        push    ax
        add     bx,2
        mov     ax,[si+bx]
        add     cx,2
        mov     bl,al
        mov     al,ah
        call    convhex
        mov     al,bl
        call    convhex
        pop     ax
sinext: inc     cx
        cmp     dl,1
        jne     unbyte          ;s=0 y w=1 implica dato de 2 bytes
        inc     cx
        mov     bl,al
        mov     al,ah
        call    convhex
        mov     al,bl
unbyte: call    convhex
        pop     ax
        pop     bx
        retn
datadic endp

datret  proc    near
;-------------------------------------------------------------------------
;Este procedimiento decide si ciertas instrucciones requieren datos, y, de
;ser asi, los incluye como string desde ES:DI. Acomoda DI como readarray.
;
;Entrada:       AH = opcode
;
;Salida:        AL = w
;               CX = long. de instrucción
;
;Usa:           conv2hex
;-------------------------------------------------------------------------
        push    bx
        test    ah,1
        jz      haydat          ;si el lsb del opcode es 0, hay datos
        mov     cx,1
        jmp     short findatret
haydat: mov     al,' '          ;poner un espacio
```

```
        stosb
        mov     bx,l1
        inc     bx
        call    conv2hex        ;poner dato
        mov     cx,3
findatret:
        mov     al,1            ;w=1
        pop     bx
        retn
datret  endp

absoluto        proc    near
;---------------------------------------------------------------
;Este procedimiento pone un string de dirección absoluta desde ES:DI,
;acomodando DI como readarray.
;
;Salida:        AL = w
;               CX = long. de instrucción
;
;Usa:           conv2hex
;---------------------------------------------------------------
        push    bx
        mov     al,' '
        stosb                   ;poner un espacio
        mov     bx,l1
        add     bx,3            ;apuntar al segundo par de bytes
        call    conv2hex
        mov     al,':'
        stosb                   ;poner dos puntos
        sub     bx,2            ;apuntar al primer par de bytes
        call    conv2hex
        mov     cx,5
        mov     al,2
        pop     bx
        retn
absoluto        endp

cod3l1inc       proc    near
;---------------------------------------------------------------
;Este procedimiento encuentra el código de 3 bits del byte siguiente
;al opcode en BL. Es un procedimiento auxiliar.
;---------------------------------------------------------------
        mov     bx,l1
        add     bx,l2
        add     bx,l3
        mov     bl,[si+bx+1]
        call    cod3bit
```

```
        retn
cod3l1inc       endp

incognita       proc    near
;---------------------------------------------------------------
;Este procedimiento coresponde al desensamblado de un código
;totalmente inválido, y establece los parámetros correspondientes.
;---------------------------------------------------------------
        mov     si,offset operm?
        mov     cx,1            ;???
        call    readarray
        xor     al,al           ;terminador
        stosb
        mov     cx,1            ;long.
        mov     al,255
        retn
incognita       endp
        public  desens

desens  proc    near
;---------------------------------------------------------------
;Este es el procedimiento principal. Genera un string desde ES:DI
;terminado en 0, desensamblado de la instrucción apuntada por DS:SI.
;Preserva todos los registros que no transfieren parámetros.
;
;Salida:        CX = longitud de la instrucción
;               AL = tipo de operador:
;                       0 = byte        (B)
;                       1 = word        (D)
;                       2 = dword       (D)
;                       3 = qword       (Q)
;                       4 = T (temporal del coprocesador 80 bits)
;                       255 = indefinido
;---------------------------------------------------------------
        pushf
        push    bx
        push    dx
        push    bp
        push    ds
        push    si
        push    di
        cld
        mov     bx,si
        mov     ax,@data
        push    es
        push    di
        mov     es,ax           ;segmento de datos (para instrucciones desensambladas)
```

```
        mov     di,offset opcode ; direccion de datos temporal para transferir
                                 ;instruccion(codigo de maquina) a desensambalr
        mov     cx,14
        rep     movsb           ;transferir instrucción al segmento de datos
        mov     ds,ax
        mov     add1,bx         ;transferir a "add1" la dirección de la instrucción que
                                ;esta siendo desensamblada
        mov     si,offset opcode
        push    si
        mov     bx,0
        mov     al,[si]         ; transferir a "al" primer byte del codigo de maquina de
la instruccion
        mov     ah,al
        mov     l3,0
        cmp     ah,67h          ;es dirección extendida
        jne     NoDirec32Bit
        mov     l3,1
        add     bx,l3
        mov     al,[si+bx]
        mov     ah,al
NoDirec32Bit:
        mov     l2,0
        cmp     ah,66h          ;son registros extendidos
        jne     noext           ;no
        mov     l2,1
        add     bx,l2
        mov     al,[si+bx]
        mov     ah,al
noext:
        mov     mne,0
        mov     l1,0
        and     ah,0e7h
        mov     si,offset segreg
        cmp     ah,26h          ;prefijo de redef. de segmento CS,ES,DS,SS?
        je      short sipref
        mov     ah,al
        mov     si,offset segreg2
        cmp     ah,65h          ;prefijo de redfinición de segmento GS
        je      short sipref
        mov     si,offset segreg1
        cmp     ah,64h          ;prefijo de redefinición de segmento FS
        jne     proced          ;nopref
sipref:
        mov     l1,1            ;opcode desplazado un byte
        and     al,18h
        mov     cl,2
        shr     al,cl           ;número del segmento * 2
```

```
        xor     ah,ah
        add     si,ax
        mov     al,[si]
        mov     mne,al
proced: pop     si
        mov     mcmop,0
        mov     bx,l1
        add     bx,l2
        add     bx,l3
        mov     ah,[si+bx]      ;opcode en ah
        cmp     ah,0feh         ;primer byte es mayor a feh?
        jb      menfe           ;no
        pop     di
        pop     es
        push    di
        call    casefetoff
        jmp     short ajustar
menfe:  mov     di,offset casos ;ver en que caso está el primer byte del o
                                ;pcode
        mov     al,ah
        xor     bx,bx
buscop: scasb
        jb      encontrado
        inc     bx
        jmp     short buscop
encontrado:
        pop     di
        pop     es
        shl     bx,1
        add     bx,offset direcc
        push    di
        call    [bx]    ;saltar a la rutina de desensamblado correspondiente

ajustar:pop     di              ;incluir el prefijo de redefinición de
        cmp     l1,0            ;segmento de ser posible y necesario
        je      fin
        cmp     mcmop,0
        je      prpref
        push    ax
        push    cx
        push    di
        mov     al,'['
        mov     cx,30
        repne   scasb           ;buscar un [
        push    di
        xor     al,al
        repne   scasb           ;buscar un null
```

```
        pop     cx
        sub     cx,di           ;cx debe tener el número de caracteres que
        neg     cx              ;van a moverse a la derecha para que pueda
        inc     cx              ;entrar el prefijo de redefinición de seg.

        push    ds
        push    es
        pop     ds
        mov     si,di
        dec     si
        add     di,2            ;preparar el movimiento
        std                     ;hacia atrás
        rep     movsb           ;acomodar string
        cld
        pop     ds

        mov     word ptr es:[di-1],':S'
        mov     al,mne
        mov     es:[di-2],al
        pop     di
        pop     cx
        pop     ax
        inc     cx  ;long. de inst. incluye prefijo de redefinición de segmento
        jmp     short fin
prpref:
        mov     al,mne
        mov     es:[di],al
        mov     word ptr es:[di+1],':S'
        mov     byte ptr es:[di+3],0
        mov     al,255
        mov     cx,1
fin:    add     cx,13   ;long. de instrucción incluye prefijo de dirección
        pop     di
        pop     si
        pop     ds
        pop     bp
        pop     dx
        pop     bx
        popf
        retn
desens  endp



case0to3f       proc    near
        cmp     ah,0fh          ;es el caso 0Fh
        jne     nocaso0f        ;no
```

```
        mov   bx,1
        add   bx,l1
        add   bx,l2
        add   bx,l3
        mov   al,[si+bx]      ;primer byte del opcode
        mov   ah,al
        mov   bl,[si+bx+1]    ;segundo byte del opcode
        mov   bh,bl
        call  cod3bit         ;obtener los bits 5,6,7 en bh
        push  ax
        mov   al,1
        push  si
        inc   si
        push  es
        push  ds
        pop   es
        mov   dx,offset atastr
        mov   astring,dx
        mov   dx,offset atbstr
        mov   bstring,dx
        mov   ah,3
        call  modrm           ;hallar el operando correspondiente a MOD y R/M
        pop   es
        pop   si
        pop   ax
        cmp   ah,2
        jae   casof23
        push  si              ;casos 0f00h a 0f01h
        mov   si,offset oper0F00 ;instrucciones SLDT,STR,LLDT,VERR,VERW
        cmp   ah,0
        je    escero
        mov   si,offset oper0f01 ;instrucciones SGDT,SIDT,LGDT,LIDT,SMSW
                                 ;LMSW,INVLPG
escero: push  cx
        mov   cl,bl
        xor   ch,ch
        call  readarray       ;leer el mnemotécnico
        pop   cx
        pop   si
        mov   ax,' '
        stosb
        mov   ax,1
        call  addbstr         ;agrarar el operando
        jmp   fin0f1

casof23:
        cmp   ah,6            ;casos 0F02h y 0F03h
```

```
        jae     casof6
        mov     cl,ah
        xor     ch,ch
        push    si
        mov     si,offset oper0f02   ;instrucciones LAR,LSL
        call    readarray
        pop     si
        mov     ah,3
        push    si
        inc     si
        call    modregrm    ;coloca desde ES:DI los operandos correspondientes
        pop     si
        jmp     fin0f1

casof6: cmp     ah,20h
        jae     casof20     ;caso 0F06h
        mov     cl,ah
        and     cx,3
        push    si
        mov     si,offset oper0f06  ;instrucciones INVD,WBINVD,CLTS
        call    readarray
        pop     si
        mov     cx,1                ;longitud, no tiene operandos
        jmp     fin0f1

casof20:
        cmp     ah,80h
        jae     casof80             ;caso 0F20h
        push    si
        push    cx
        mov     cx,0
        mov     si,offset operm?    ;instrucción MOV con registros especiales
        call    readarray
        push    ax
        mov     ax,' '              ;Poner un espacio luego del mnemotécnico
        stosb
        pop     ax
        pop     cx
        pop     si
        push    si
        push    ax
        and     ah,4
        cmp     ah,4
        jne     notr
        mov     si,offset regtr     ;leer registros TR
        jmp     short leeee
notr:   mov     ah,al
```

```
        and     ah,1
        cmp     ah,0
        jne     nocr
        mov     si,offset regcr    ;leer registros CR
        jmp     short leeee
nocr:   mov     ah,al
        mov     si,offset regdr    ;leer registros DR
leeee:  pop     ax
        push    di
        push    es
        push    ds
        pop     es
        mov     di,astring         ;obtener primer operando
        mov     cl,bl
        xor     ch,ch
        call    readarray
        mov     di,bstring
        mov     cl,bh
        and     cx,7
        push    si
        mov     si,offset reg3     ;segundo operando
        call    readarray
        pop     si
        pop     es
        mov     dh,ah              ;obtener d en DH
        and     dh,2
        pop     di
        call    order              ;poner en orden los operandos
        mov     cx,2
        pop     si
        mov     al,2
        jmp     fin0f1

casof80:
        cmp     ah,90h
        jae     casof90            ;caso 0F80h
        push    si
        inc     si
        call    case70to7f         ;instrucciones de saltos condicionales de 16
                                   ;y 32 bits
        pop     si
        mov     al,1
        jmp     fin0to3f

casof90:
        cmp     ah,0a0h
        jae     casofa0
```

```
        push    si              ;caso 0F90h
        mov     si,offset oper0f90  ;instrucciones SETO,SETNO,SETB,SETNB,SETE

;SETNE,SETBE,SETA,SETS,SETNS,SETP,SETNP
                                    ;SETL,SETNL,SETLE,SETG,'
        push    cx
        mov     cl,ah
        and     cx,15
        call    readarray       ;poner el Mnemotécnico
        pop     cx
        pop     si
        mov     ax,' '
        stosb
        push    si
        push    es
        push    ds
        pop     es
        mov     ax,0
        inc     si
        call    modrm           ;leer operando correspondiente a MOD y R/M
        pop     es
        pop     si
        call    addbstr         ;poner el operando
        jmp     fin0f1

casofa0: cmp    ah,0b0h
        jae     casofb0
        push    ax              ;caso 0FA0h
        and     ah,7
        cmp     ah,2
        pop     ax
        jae     mayora2
        push    si
        mov     si,offset operpp ;instrucciones PUSH,POP con segmentos GS y FS
        mov     cl,ah
        and     cx,7
        call    readarray
        mov     si,offset segreg
        mov     bl,ah
        call    cod3bit
        mov     cl,bl
        xor     ch,ch
        call    readarray       ;leer F o G a cl
        mov     al,'S'
        stosb
        mov     cx,1            ;longitud de a instrucción de un byte
        mov     al,1
```

```
        pop     si
        jmp     fin0f1

mayora2:
        push    ax
        and     ah,7
        cmp     ah,3
        pop     ax
        jne     mayora3
        push    si
        and     ah,8
        mov     cx,3
        shr     ah,cl
        mov     cl,ah
        mov     si,offset   oper0fa3    ;instrucciones BT,BTS,BTR,BTC
        call    readarray
        pop     si
        mov     ah,1
        push    si
        inc     si
        call    modregrm            ;poner operandos
        pop     si
        jmp     fin0f1

mayora3:
        push    ax
        push    si
        mov     al,ah
        and     ah,0ah
        mov     cl,ah
        shr     cl,1
        xor     ch,ch
        mov     si,offset oper0fa4  ;instrucciones SHLD,SHRD,IMUL
        call    readarray
        pop     si
        mov     ah,1
        cmp     al,0afh
        jne     noimul
        mov     ah,3
noimul: push    si
        inc     si
        call    modregrm        ;poner operandos
        pop     si
        pop     ax
        mov     al,1
        add     ax,12
        push    ax
```

```
        mov     al,','
        stosb
        and     ah,7
        cmp     ah,4
        jne     no4
        mov     bx,1
        add     bx,11
        add     bx,cx
        mov     al,[si+bx]
        call    convhex         ;poner el tercer operando
        inc     cx
        jmp     short   fin0f2
no4:    cmp     ah,5
        jne     no5
        mov     ax,'lc'
        stosw
        jmp     short   fin0f2
no5:    dec     di
fin0f2: pop     ax
        jmp     fin0f1

casofb0:
        cmp     ah,0c8h
        jae     casofc8
        and     ah,0FH      ;Caso 0Fb0h
        cmp     ah,2
        jae     maque2
        mov     cl,4
        and     al,16
        shr     al,cl
        mov     cl,al
        xor     ch,ch
        push    si
        mov     si,offset oper0fb0    ;instrucciones XADD,CMPXCHG
        call    readarray
        pop     si
        and     ah,1
        push    si
        inc     si
        call    modregrm              ;poner operandos
        pop     si
        jmp     fin0f1

maque2: cmp     ah,6
        jae     maque6
        and     ah,3
        mov     cl,ah
```

```
        xor     ch,ch
        push    si
        mov     si,offset oper0fb2    ;instrucciones LFS,LGS,LSS,BTR
        call    readarray
        pop     si
        cmp     ah,3
        jne     no03
        mov     al,1
        jmp     proce1
no03:   mov     ah,3
        push    si
        inc     si
        call    modregrm      ;poner operandos
        pop     si
        jmp     fin0f1

maque6: push    ax
        and     ah,6
        cmp     ah,6
        pop     ax
        jne     noes6
        mov     al,ah
        and     ah,8
        mov     cl,3
        shr     ah,cl
        mov     cl,ah
        xor     ch,ch
        push    si
        mov     si,offset oper0fb6    ;instrucciones MOVZX,MOVSX
        call    readarray
        pop     si
        mov     ah,al
        and     ah,3
proce:
        push    si
        inc     si
        call    modregrm1             ;poner operandos
        pop     si
        jmp     fin0f1

noes6:  cmp     ah,0ch
        jae     noesaob
        push    ax
        push    cx
        cmp     ah,0bh
        jnc     noesb
        and     ah,3
```

```
        mov     cl,ah
        jmp     short si0b
noesb:  and     bl,3
        mov     cl,bl
si0b:   xor     ch,ch
        push    si
        mov     si,offset oper0fa3      ;instrucciones BSF,BSR
        call    readarray
        pop     si
        mov     al,' '
        stosb
        pop     cx
        pop     ax
        cmp     ah,0bh
        jne     no0b
        mov     ah,1
        dec     di
        jmp     short proce1
no0b:
        call    addbstr                 ;poner operando
        mov     al,','
        stosb
        mov     bx,1
        add     bx,11
        add     bx,cx
        mov     al,[si+bx]
        call    convhex                 ;poner segundo operando
        inc     cx
        mov     al,1
        add     ax,12
        jmp     short fin0f1
noesaob:
        and     ah,3
        mov     cl,ah
        push    si
        mov     si,offset oper0fb3      ;instruciones BSF,BSR
        call    readarray
        pop     si
        mov     ah,3
proce1: push    si
        inc     si
        call    modregrm                ;poner operandos
        pop     si
        jmp     short   fin0f1

casofc8:
        xor     cx,cx
```

```
        mov     si,offset oper0fc8      ;instrucción BSWAP
        call    readarray
        mov     si,offset reg3          ;poner operandos
        mov     al,' '
        stosb
        mov     cl,ah
        and     cx,7
        call    readarray
        mov     ax,1
        add     ax,l2
        mov     cx,1
        add     cx,l2
fin0f1: add     cx,1
        jmp     fin0to3f

nocaso0f:mov    bl,ah           ;opcode
        call    cod3bit         ;obtiene b its 3,4,5 en bl(bits 0,1,2)
        and     ah,7
        cmp     ah,5
        ja      mayque5         ;bifurcar si superior (de 0 a 5 son sumas)
        mov     cl,bl
        xor     ch,ch
        push    si
        mov     si,offset oper1
        call    readarray       ;lee add,or,adc,sbb,and,sub,xor,cmp de acuerdo
        pop     si              ;a cl
        cmp     ah,3            ;de 0 a 3 trabaja con un registro y direccionam
        ja      mayque3
        mov     dx,offset atastr
        mov     astring,dx
        mov     dx,offset atbstr
        mov     bstring,dx
        call    modregrm        ;coloca operandos
        jmp     short fin0to3f
mayque3:                        ;reg con inmediato
        mov     al,ah
        and     al,1            ;al=w
        call    accdata ;genera string de datos de acc con operando inmediato
        jmp     short fin0to3f
mayque5:
        cmp     ah,6
        ja      mayque6         ;valor 6 es push
        push    si
        cmp     bl,3
        ja      codmq3
        mov     si,offset operpp  ;para leer push
        xor     cx,cx
```

```
        call    readarray
        mov     si,offset segreg    ; para leer registro de segmento E o C
        mov     cl,bl
        call    readarray
        mov     al,'S'
        stosb
        jmp     short fin6
codmq3: mov     si,offset segreg
        mov     cl,bl
        and     cx,3
        call    readarray           ;para leer reg de seg S,D
        mov     ax,':S'
        stosw
fin6:   mov     cx,1
        mov     al,1
        pop     si
        jmp     short fin0to3f
mayquc6:
        push    si
        cmp     bl,3
        ja      cm3
        mov     si,offset operpp
        mov     cx,1                ;pop
        call    readarray
        mov     si,offset segreg
        mov     cl,bl
        call    readarray           ;leer E,S o D  de acuerdo a cl
        mov     al,'S'
        stosb
        jmp     short fin7

cm3:    mov     si,offset oper2
        mov     cl,bl
        and     cx,3
        call    readarray           ;dAA,das,aaa,aas de auerdo a cl
fin7:   mov     cx,1
        mov     al,1
        pop     si
fin0to3f:
        push    ax
        xor     al,al
        stosb
        pop     ax
        retn
case0to3f       endp

case40to5f      proc    near
```

```
        mov   bl,ah
        and   bl,18h
        mov   cl,3
        shr   bl,cl
        mov   cl,bl
        xor   ch,ch
        mov   si,offset oper3   ;instrucciones INC,DEC,PUSH,POP
        call  readarray
        cmp   l2,1              ;registro extendido?
        jne   short noregext    ;no
        mov   si,offset reg3
        jmp   short  careg
noregext:
        mov   si,offset reg2
careg:  mov   al,' '
        stosb
        mov   cl,ah
        and   cx,7
        call  readarray         ;poner operando
        xor   al,al
        stosb
        mov   ax,1
        add   ax,l2
        mov   cx,1
        add   cx,l2
        retn
case40to5f   endp

case60to6f   proc   near
        mov   dx,offset atastr
        mov   astring,dx
        mov   dx,offset atbstr
        mov   bstring,dx
        cmp   ah,68h
        jb    mena68maya6b
        cmp   ah,6bh
        ja    mena68maya6b
        mov   cl,ah
        and   cx,1
        push  si
        mov   si,offset operpi    ;instrucciones PUSH,IMUL
        call  readarray
        pop   si
        mov   cx,1
        mov   dl,ah
        and   ah,1
        jnz   simodregrm
```

```
        add     cx,l2
        mov     al,0
        test    dl,2
        jnz     short   nomodregrm      ;solo tiene un operando
        mov     al,1
        add     ax,l2
        jmp     short   nomodregrm      ;solo tiene un operando
simodregrm:
        add     ah,2
        call    modregrm    ;poner primeros operandos
        push    ax
        mov     ah,al
        mov     al,','      ;poner una coma
        stosb
        pop     ax
nomodregrm:
        and     dl,2
        push    ax
        add     dl,1
        mov     bx,l1           ;apuntar al opcode
        add     bx,l3
        add     bx,cx           ;apuntar a los datos
        mov     ax,[si+bx]
        and     dl,2
        jnz     unbyte1
        cmp     l2,1            ;datos de 32 bits?
        jne     sinext1         ;no
        push    ax
        add     bx,2
        mov     ax,[si+bx]
        add     cx,2
        mov     bl,al
        mov     al,ah
        call    convhex         ;poner tercer operando de 32 bits
        mov     al,bl
        call    convhex
        pop     ax
sinext1:
        inc     cx
        mov     bl,al
        mov     al,ah
        call    convhex
        mov     al,bl
unbyte1:
        inc     cx
        call    convhex
        pop     ax
```

```
        jmp     short  fin61

mena68maya6b:
        cmp     ah,64h
        jae     maque3
        mov     cl,al
        and     cx,3
        push    si
        mov     si,offset operpba  ;instrucciones PUSHA,POPA,BOUND,ARPL
        call    readarray
        pop     si
        mov     cx,1
        test    ah,2
        jz      es01
        mov     ah,1
        call    modregrm           ;poner operandos
es01:   jmp     fin61

maque3:
        cmp     ah,6ch
        jb      desconocido        ;instrucción no reconocida
        push    si
        mov     si,offset operios  ;instrucciones INS,OUTS
        and     al,2
        shr     al,1
        mov     cl,al
        xor     ch,ch
        call    readarray          ;leer el mnemotécnico
        pop     si
        and     ah,1
        call    wob          ;poner el carácter de palabra, doble palabra
        jmp     short  fin61
desconocido:
        call    incognita    ;instrucción no reconocida

fin61:  push    ax
        xor     al,al
        stosb
        pop     ax
        retn

case60to6f      endp

case70to7f      proc   near
        mov     al,'J'    ;instrucciones JO,JNO,JC,JNC,JZ,JNZ,JBE,JA,JS,JNS,JP,J
                          ;JNP,JL,JGE,JLE,JG
        stosb
```

```
        push    si
        mov     si,offset oper4
        mov     cl,ah
        and     cx,15
        call    readarray   ;leer el mnemotécnico de acuerdo al valor en cl
        pop     si
        cmp     ah,80h
        jb      esb
        mov     al,1
        mov     cx,4
        jmp     short esw
esb:    xor     al,al
        mov     cx,2
esw:    call    realadr     ;calcular la dirección real del salto
        push    ax
        xor     ax,ax
        stosb
        pop     ax
        retn
case70to7f      endp

case80to83      proc    near
        call    cod3l1inc
        mov     cl,bl
        xor     ch,ch
        push    si
        mov     si,offset oper1 ;instrucciones ADD,OR,ADC,SBB,AND
                                ;SUB,XOR,CMP
        call    readarray
        pop     si
        mov     dl,ah
        and     dl,3            ;sw
        call    datadic         ;poner dato directo
        add     ax,12
        push    ax
        xor     al,al
        stosb
        pop     ax
        retn
case80to83      endp

case84to87      proc    near
        mov     cl,ah
        and     cx,2
        shr     cl,1
        push    si
        mov     si,offset opertx    ;instrucciones TEST,XCHG
```

```
        call    readarray
        pop     si
        and     ah,1
        or      ah,2            ;obtener dw
        mov     dx,offset atastr
        mov     astring,dx
        mov     dx,offset atbstr
        mov     bstring,dx
        call    modregrm        ;poner operandos
        push    ax
        xor     al,al
        stosb
        pop     ax
        retn
case84to87      endp

case88to8b      proc    near
        push    si
        mov     si,offset operm?
        xor     cx,cx           ;mov
        call    readarray       ;poner el mnemotécnico
        pop     si
        and     ah,3
        mov     dx,offset atastr
        mov     astring,dx
        mov     dx,offset atbstr
        mov     bstring,dx
        call    modregrm        ;poner operandos
        push    ax
        xor     al,al
        stosb
        pop     ax
        retn
case88to8b      endp

case8cto8f      proc    near
        push    si
        test    ah,1
        jnz     leapop
        call    cod311inc
        mov     si,offset operm?
        xor     cx,cx           ;mov
        cmp     bl,5
        jbe     valido
        inc     cx              ;instrucción no reconocida
valido: call    readarray
        mov     si,offset segreg
```

```
        shl     bl,1
        add     si,bx
        push    es
        push    di
        push    ds
        pop     es
        mov     di,offset atastr    ;preparar espacio para primer operando
        mov     astring,di
        movsb
        mov     al,'S'
        stosb
        mov     di,offset atbstr    ;preparar espacio para segundo operando
        mov     bstring,di
        pop     di
        pop     es
        mov     al,1
        pop     si
        call    modrm               ;leer operando correspondiente a MOD y R/M
        mov     dh,ah
        and     dh,2            ;d
        call    order               ;ordenar los operandos
        jmp     short fin8cto8f
leapop: test    ah,2
        jz      eslea
        call    cod311inc           ;obtener el bl los bits 5,6,7 del segundo
        or      bl,bl               ;byte del opcode
        jz      espop
        mov     si,offset operm?
        mov     cx,1                ;???
        call    readarray           ;instrucción no válida
        mov     al,' '
        stosb
        jmp     short inclmod
espop:  mov     si,offset operpp
        mov     cx,1                ;pop
        call    readarray
inclmod:pop     si
        mov     dx,offset atbstr
        mov     bstring,dx
        mov     al,1
        call    modrm               ;determinar el operando
        call    addbstr
        jmp     short fin8cto8f
eslea:  mov     si,offset operlea
        xor     cx,cx           ;lea
        call    readarray
        mov     dx,offset atastr    ;prepara los espacios temporales para
```

```
        mov     astring,dx          ;los operandos
        mov     dx,offset atbstr
        mov     bstring,dx
        mov     ah,3
        pop     si
        call    modregrm            ;ubicar los operandos
fin8cto8f:
        push    ax
        xor     al,al
        stosb
        pop     ax
        retn
case8cto8f      endp


case90  proc    near
        mov     si,offset opernop
        xor     cx,cx               ;nop
        call    readarray           ;sin operandos
        xor     al,al
        stosb                       ;poner cero a continuación
        mov     al,1                ;poner tipo de instrucción
        mov     cx,1                ;longitud de instrucción 1byte
        retn
case90  endp

case91to97      proc    near
        mov     si,offset operxax
        xor     cx,cx           ;xchg ax
        add     cx,l2
        call    readarray       ; mnemotécnico
        mov     ax,','          ;poner una coma
        stosb
        mov     cl,ah
        and     cx,7
        mov     si,offset reg2      ;poner segundo registro operando
        cmp     l2,0
        je      espalabra
        mov     si,offset reg3
espalabra:      call    readarray
        xor     al,al
        stosb
        mov     al,1
        add     ax,l2
        mov     ex,1
        add     cx,l2
        retn
```

```
case91to97    endp

case98to99    proc   near
        mov   si,offset opercc
        xor   cx,cx              ;instrucción CWD
        mov   al,ah              ;
        cmp   l2,0
        je    wd
        mov   cx,2
        and   al,1    ;w
        jnz   wd
        inc   cx                 ;instrcución CBW
wd:     call  readarray
        mov   cx,1
        add   cx,l2
        push  ax
        xor   al,al
        stosb
        pop   ax
        add   ax,l2
        retn
case98to99    endp

casc9a  proc  near
        push  si
        mov   si,offset operjc
        mov   cx,1               ;instrucción CALL
        call  readarray
        pop   si
        call  absoluto           ;poner dirección absoluta del call
        push  ax
        xor   al,al              ;poner un 0 al final de la instrucción
        stosb
        pop   ax
        retn
case9a  endp

case9b  proc  near
        mov   si,offset operwai
        xor   cx,cx         ;wait
        call  readarray
        xor   al,al
        stosb
        mov   al,255
        mov   cx,1
        retn
case9b  endp
```

```
case9cto9f    proc   near
        mov    si,offset oper5       ;instrucciones PUSH,POP;LAH,SAH
        mov    cl,ah
        and    cx,3
        call   readarray
        cmp    l2,1
        je     esppd
        mov    ax,46h        ;trabajo con  el registro de banderas
        stosw
        jmp    noesppd
esppd:
        mov    ax,'DF'       ;trabajo conel registro de banderas extendido
        stosw
        mov    al,' '
        stosb
noesppd:
        mov    al,1
        mov    cx,1          ;longitud de la instrucción
        add    cx,l2         ;sumar el prefijo de datos de 32 bits
        retn
case9cto9f    endp

casea0toa3    proc   near
        push   si
        mov    memop,1             ;trabajo con memoria
        mov    si,offset opernn?
        xor    cx,cx          ;MOV del acumulador con memoria direccionada
        call   readarray       ;direccionada directaemnte
        pop    si
        mov    al,' '          ;poner un espacio
        stosb
        mov    al,ah
        and    al,1
        mov    dx,offset reg1        ;registro tipo byte
        jz     prfil
        mov    dx,offset reg2        ;registro tipo palabra
        cmp    l2,0
        je     prfil
        mov    dx,offset reg3        ;registro extendido
prfil:  push   ax
        test   ah,2
        jz     apr
        call   ponb              ;poner primero la localidad de memoria
        mov    al,','           ;instrucción del tipo mov [####],ax
        stosb
        call   pona              ;poner el registro
        jmp    short fina0toa3
```

```
apr:    call    pona            ;poner primero el registro
        mov     al,','          ;instrucción del tipo mov ax,[####]
        stosb
        call    ponb            ;poner la dirección de memoria
fina0toa3:
        xor     al,al
        stosb
        pop     ax
        mov     cx,3
        add     cx,l2
        add     ax,l2
        retn
pona:   xor     cx,cx           ;pone el registro en la línea de instrucción
        push    si
        mov     si,dx           ;dx apunta al arreglo de nombres de registros
        call    readarray
        pop     si
        retn
ponb:   mov     al,'['          ;pone la dirección de memoria del tipo [####]
        stosb                   ;el número entre corchetes puede se byte, word
        mov     bx,l1           ;o doble word
        add     bx,l2
        add     bx,l3
        inc     bx
        push    l2
        mov     l2,0
        call    conv2hex        ;convertir número hexadecimal en ASCII
        pop     l2
        mov     al,']'
        stosb
        retn
casea0toa3      endp

casea4toa7      proc    near
        mov     si,offset operem
        xor     cx,cx           ;instrucción CMPS
        test    ah,2
        jnz     escmps
        inc     cx              ;instrucción MOVS
escmps: call    readarray
        call    wob             ;poner el tipo de datos que manejan
        push    ax
        xor     al,al
        stosb
        pop     ax
        retn
casea4toa7      endp
```

```
casea8toa9    proc   near
        push   si
        mov    si,offset opertx
        xor    cx,cx          ;instrucción TEST
        call   readarray
        mov    al,ah
        and    al,1
        pop    si
        call   accdata        ;acumulador con dato inmediato
        push   ax
        xor    al,al
        stosb
        pop    ax
        retn
casea8toa9    endp

caseaatoab    proc   near
        mov    si,offset opersto
        xor    cx,cx          ;STOS
        call   readarray
        call   wob            ;tipo de operandos con los que trabaja
        push   ax
        xor    al,al          ;almacenar el cero al final de la instrucción
        stosb
        pop    ax
        retn
caseaatoab    endp

caseactoaf    proc   near
        mov    si,offset opersl
        xor    cx,cx          ;instrucción SCAS
        test   ah,2
        jnz    esscas
        inc    cx             ;instrucción LODS
esscas: call   readarray
        call   wob            ;tipo de datos
        push   ax
        xor    al,al
        stosb
        pop    ax
        retn
caseactoaf    endp

caseb0tobf    proc   near
        push   si
        mov    si,offset operm?
        xor    cx,cx          ;instricción MOV registro con dato inmediato
```

```
        call    readarray
        mov     al,' '
        stosb
        mov     al,ah
        and     al,8
        mov     cl,3
        shr     al,cl
        mov     si,offset reg1    ;registro de 1 byte
        jz      esr1
        mov     si,offset reg2    ;registro de palabra
        cmp     l2,0
        je      esr1
        mov     si,offset reg3    ;registro de doble palabra
esr1:   mov     cl,ah
        and     cx,7
        push    ax
        call    readarray         ;se lee el eregistro
        mov     al,','
        stosb
        pop     ax
        pop     si
        push    ax
        mov     bx,l1
        add     bx,l2
        add     bx,l3
        mov     cl,al
        or      al,al
        jz      es1b
        cmp     l2,0
        je      es1w
        add     cx,2
        mov     al,[si+bx+4]    ;agregar el dato inmediato sea byte,
        call    convhex         ;palabra o doble palabra
        mov     al,[si+bx+3]
        call    convhex
es1w:   mov     al,[si+bx+2]
        call    convhex
es1b:   mov     al,[si+bx+1]
        call    convhex
        xor     al,al
        stosb
        pop     ax
        add     cx,2
        add     cx,l2
        add     ax,l2   ;ax entrega el tipo de datos con el que se trabaja
        retn
caseb0tobf      endp
```

```
casec0toc1      proc    near
        call    cod3l1inc    ;obtener bits 5,6,7 de segundo byte del opcode
        mov     cl,bl        ;en los bytes 0,1,2 de bl
        xor     ch,ch
        push    si
        mov     si,offset oper6   ;instrucciónes ROL,ROR,RCL,RCR,SHL,SHR,SAR

        call    readarray
        pop     si
        mov     al,' '
        stosb
        mov     al,ah
        and     al,1
        mov     dx,offset atbstr
        mov     bstring,dx
        call    modrm        ;obtener el primer operando: registro o memoria
        call    addbstr      ;poner el primer operando
        push    ax
        mov     al,','       ;poner un acoma
        stosb
        mov     bx,cx
        mov     al,[si+bx]
        call    convhex  ;agrgar el segundo operando que es un dato inmediato

        xor     al,al
        stosb
        pop     ax       ;poner cero al final
        inc     cx
        add     ax,l2
        retn

casec0toc1      endp

casec2toc3      proc    near
        push    si
        mov     si,offset operrn
        xor     cx,cx    ;instrucción RETN
        call    readarray
        pop     si
        call    datret   ;poner dirección de retorno si es que es necesaria
        push    ax
        xor     al,al
        stosb
        pop     ax
        retn
casec2toc3      endp
```

```
casec4toc5    proc   near
        push   si
        mov    si,offset operll
        xor    cx,cx          ;instrucción LDS
        test   ah,1
        jnz    eslds
        inc    cx             ;instrucción LES
eslds:  call   readarray
        pop    si
        mov    ax,offset atastr
        mov    astring,ax
        mov    ax,offset atbstr
        mov    bstring,ax
        mov    ah,3
        call   modregrm       ;obtener y colocar operandos
        push   ax
        xor    al,al
        stosb
        pop    ax
        retn
casec4toc5    endp

casec6toc7    proc   near
        call   cod3l1inc
        push   si
        mov    si,offset operm?
        xor    cx,cx          ;instrucción MOV de dato inmediato a memoria
        or     bl,bl
        jz     esmov
        inc    cx             ;???
esmov:  call   readarray
        pop    si
        mov    dl,ah
        and    dl,1           ;sw
        call   datadic        ;poner el operando y el dato inmediato
        add    ax,l2
        push   ax
        xor    al,al
        stosb
        pop    ax
        retn
casec6toc7    endp

casec8toc9    proc   near
        push   si
        mov    si,offset operel    ;instrucciones ENTER y LEAVE
        mov    cl,ah
```

```
        and     cx,1
        call    readarray
        pop     si
        cmp     cl,1
        je      finc8c9
        mov     al,' '
        stosb
        mov     bx,1
        call    conv2hex
        mov     al,','          ;poner el dato inmediato
        stosb
        mov     al,[si+3]       ;poner segundo dato inmediato
        call    convhex
        mov     cx,4            ;longitud de la instrucción 4 bytes
finc8c9:
        mov     al,1
        retn
casec8toc9      endp

casecatocb      proc    near
        push    si
        mov     si,offset operrf
        xor     cx,cx           ;instrucción RETF
        call    readarray
        pop     si
        call    datret          ;Poner dato inmediato tipo doble palabra si
        push    ax              ;es necesario
        xor     al,al
        stosb
        pop     ax
        retn
casecatocb      endp

casecc  proc    near
        mov     si,offset operi3
        xor     cx,cx           ;Instrucción INT 3
        call    readarray
        xor     al,al
        stosb
        mov     al,2            ;intrucción tipo doble word
        mov     cx,1            ;longitud de la instrucción un byte
        retn
casecc  endp

casecd  proc    near
        push    si
        mov     si,offset operint
```

```
        xor     cx,cx           ;Istrucción INT
        call    readarray
        pop     si
        mov     bx,11
        mov     al,[si+bx+1]
        call    convhex         ;poner el número de la instrucción
        xor     al,al
        stosb
        mov     al,2
        mov     cx,2            ;longitud de la instriucción
        retn
casecd  endp

casece  proc    near
        mov     si,offset operio
        xor     cx,cx           ;instrucción
        call    readarray
        xor     al,al
        stosb
        mov     al,2
        mov     cx,1
        retn
casece  endp

casecf  proc    near
        mov     si,offset operir
        xor     cx,cx           ;instrucción IRET
        call    readarray
        xor     al,al
        stosb
        mov     al,2
        mov     cx,1
        retn
casecf  endp

cased0tod3      proc    near
        call    cod311inc
        mov     cl,bl
        xor     ch,ch
        push    si
        mov     si,offset oper6   ;Instrucciones ROL,ROR,RCL,RCR,SHL,SHR,SAR
        call    readarray
        pop     si
        mov     al,' '
        stosb
        mov     al,ah
        and     al,1
```

```
        mov     dx,offset atbstr
        mov     bstring,dx
        call    modrm           ;obtener el primer operando
        call    addbstr         ;poner el primer operando
        push    ax
        mov     al,','
        stosb
        test    ah,2
        jz      es1
        mov     ax,'LC'         ;poner el registro cl
        stosw
        jmp     short find0tod3
es1:    mov     al,'1'          ;poner 1
        stosb
find0tod3:
        xor     al,al
        stosb
        pop     ax
        add     ax,l2
        retn
cased0tod3      endp

cased4tod5      proc    near
        push    si
        mov     si,offset operaa
        xor     cx,cx           ;instrucción AAD
        test    ah,1
        jnz     esaad
        inc     cx              ;Instrucción AAM
esaad:  call    readarray
        pop     si
        mov     bx,l1
        mov     bl,[si+bx+1]
        cmp     bl,10    ;si segundo byte es igual a diez entonces es normal
        je      norm
        mov     al,' '
        stosb
        mov     al,bl
        call    convhex  ;caso contrario ponemos el dato a continuación
norm:   xor     al,al
        stosb
        mov     cx,2
        retn
cased4tod5      endp

cased6  proc    near
        call    incognita ;No hay esta instrucción
```

```
        retn
cased6  endp

cased7  proc    near
        mov     si,offset operxla
        xor     cx,cx           ;instrucción XLAT sin operandos
        call    readarray
        xor     al,al
        stosb
        mov     cx,1
        retn
cased7  endp

cased8todf      proc    near
        and     ah,7    ;**** Instrucciones del coprocesador matemático
        call    cod3l1inc ;obtiene bits 3,4,5 en bl del segundo byte
                        ;del opcode
        call    detectmem ;obtiene bits 6,7 en dl del segundo byte del opcode
        cmp     ah,0
        jne     notcero
        mov     ci,bi       ;casos ESC000
        xor     ch,ch
        cmp     dl,3
        je      nohaymem
        push    si
        mov     si,offset op000a ;instrucciones FADD,FMUL,FCOM,FCOMP,FSUB,
                                 ;FSUBR,FDIV,FDIVR
        call    readarray
        pop     si
        mov     al,2
        call    copmem          ;poner operando: dirección de memoria
        jmp     find8
nohaymem:
        push    si
        mov     si,offset op000b  ;instrucciones FADD st,FMUL st,FCOM,FCOMP,
                                  ;FSUBR st,FSUB st,FDIVR st,FDIV st,'
        call    readarray
        pop     si
        mov     al,','
        cmp     bl,2
        jb      nocomp
        cmp     bl,4
        jae     nocomp
        mov     al,' '
nocomp:
        stosb
        call    copreg          ;poner operando: registro del coprocesador
```

```
        jmp     find8

notcero: cmp    ah,1
        jne     notuno
        push    bx              ;casos ESC001
        mov     bx,l1
        add     bx,l2
        add     bx,l3
        mov     dh,[si+bx+1]
        cmp     dh,0d0h
        pop     bx
        jae     nohaymem1
        mov     cl,bl
        xor     ch,ch
        push    si
        mov     si,offset op001a  ;instrucciones FLD,FXCH st,FST,FSTP,FLDENV,
                                  ;FLDCW,FSTENV,FSTCW
        call    readarray
        pop     si
        cmp     dl,3
        je      poneregcop
        mov     al,2
        call    copmem            ;poner operando: memoria
        cmp     bl,4
        jb      no255
        mov     al,1
        test    bl,1
        jnz     no255
        mov     al,255
        jmp     find8
poneregcop:
        mov     al,','
        stosb
        call    copreg    ;poner operando: registro del coprocesador
no255:  jmp     Find8

nohaymem1:
        push    si
        mov     si,offset op001c  ;instrucciones F2XM1,FYL2X,FPTAN,FPATAN,
                                  ;FXTRACT,FPREM1,FDECSTP,FINCSTP,FPREM,
                                  ;FXL2XP1,FSQRT,FSINCOS,FRNDINT,FSCALE,
                                  ;FSIN,FCOS; no tiene operandos
        cmp     dh,0f0h
        jae     esf
        mov     si,offset op001b  ;instrucciones FNOP,FABS,FTST,FXAM,FLD1,
                                  ;FLDL2T,FLDL2E,FLDPI,FLDLG2,FLDLN2,FLDZ
esf:    mov     cl,dh             ;no tienen operandos
```

```
        and     cx,0fh
        call    readarray
        pop     si
        mov     cx,2
        mov     al,4
        jmp     find8
notuno: cmp     ah,2
        jne     notdos
        mov     cl,bl
        xor     ch,ch
        cmp     dl,3
        je      nohaymem2
        push    si
        mov     si,offset op010a   ;instrucciones FIADD,FIMUL,FICOM,FICOMP,
                                   ;FISUB,FISUBR,FIDIV,FIDIVR
        call    readarray
        pop     si
        mov     al,2
        call    copmem             ;agregar dirección de memoria
        jmp     find8
nohaymem2:
        mov     cx,0
        push    si
        mov     si,offset op010b   ;instrucción FUCOMPP
        call    readarray          ;sin operandos
        pop     si
        mov     cx,2
        mov     al,4
        jmp     find8

notdos: cmp     ah,3
        jne     nottres
        cmp     dl,3
        je      nohaymem3
        mov     cl,bl
        xor     ch,ch
        push    si
        mov     si,offset op011a   ;instrucciones FILD,FIST,FISTP,FLD,FSTP
        call    readarray
        pop     si
        mov     al,2
        call    copmem             ;agregar dirección de memoria
        cmp     bl,5
        jb      find8
        mov     al,4
        jmp     find8
nohaymem3:
```

```
        push    bx
        mov     bx,l1
        add     bx,l2
        add     bx,l3
        mov     cl,[si+bx+1]
        pop     bx
        and     cx,7
        push    si
        mov     si,offset op011b  ;intrucciones FENI,FDISI,FCLEX,FINIT,FSETPM
        call    readarray         ;sin operandos
        pop     si
        mov     cx,2
        mov     al,255
        jmp     find8

nottres:
        cmp     ah,4
        jne     notcuatro
        mov     cl,bl
        xor     ch,ch
        cmp     dl,3
        je      nohaymem4
        push si
        mov     si,offset op000a  ;instrucciones FADD,FMUL,FCOM,FCOMP,FSUB,
                                  ;FSUBR,FDIV,FDIVR
        call    readarray         ;con operandos de memoria de 64 bits
        pop     si
        mov     al,3
        call    copmem            ;poner dirección de memoria
        jmp     find8

nohaymem4:
        push    si
        mov     si,offset op000b   ;instrucciones FADD,FMUL,FCOM,FCOMP,
                                   ;FSUBR,FSUBt,FDIVR,FDIV
        call    readarray          ; de la forma Mnemotécnico st(i),st
        pop     si
        sub     di,2
        call    copreg             ;poner registro del coprocesador
        mov     al,','
        stosb
        mov     ax,'ts'            ;poner a continuación st
        stosw
        mov     al,4
        jmp     find8

notcuatro:
```

```
        cmp     ah,5
        jne     notcinco
        cmp     dl,3
        je      nohaymem5
        mov     cl,bl
        xor     ch,ch
        push    si
        mov                             si,offset       op101a      ;instrucciones
FLD,FST,FSTP,FRSTOR,FSAVE,FSTSW
        call    readarray
        pop     si
        mov     al,3    ;indica 64 bits
        call    copmem  ;poner dirección de memoria de 64 bits o más
        cmp     bl,4
        jb      find8
        mov     al,255  ;al indica má de 64bits
        cmp     bl,7
        jne     find8
        mov     al,1    ;indica 16 bits
        jmp     find8
nohaymem5:
        mov     cl,bl
        xor     ch,ch
        push    si
        mov     si,offset op101b ;instrucciones FFREE,FST,FSTP
                                 ;FUCOM st,FUCOMP st
        call    readarray
        pop     si
        mov     al,' '
        cmp     bl,4
        jb      nofucom
        mov     al,','
nofucom:
        stosb
        call    copreg      ;poner registro del coprocesador
        jmp     short find8
notcinco:
        cmp     ah,6
        jne     notseis
        cmp     dl,3
        je      nohaymem6
        mov     cl,bl
        xor     ch,ch
        pusb    si
        mov     si,offset op010a ;instrucciones FIADD,FIMUL,FICOM,FICOMP,
                                 ;FISUB,FISUBR,FIDIV,FIDIVR
        call    readarray
```

```
        pop     si
        mov     al,1            ;datos de 16 bits
        call    copmem          ;poner dirección de memoria
        jmp     short   find8
nohaymem6:
        mov     cl,bl
        xor     ch,ch
        push    si
        mov     si,offset op110b ;instrucciones FADDP,FMULP,FCOMPP,FSUBRP,
                                 ;FSUBP,FDIVRP,FDIVP
        call    readarray
        pop     si
        mov     al,4
        cmp     bl,3
        je      find8
        mov     al,' '
        stosb
        call    copreg          ;poner registro del coprocesador
        mov     al,','
        stosb
        mov     ax,'ts'         ;poner st
        stosw
        mov     al,4
        jmp     short   find8

notseis:
        mov     cl,bl
        xor     ch,ch
        push    si
        mov     si,offset op111a  ;instrucciones FILD,FIST,FISTP,FBLD,FILD,
                                  ;FBSTP,FISTP
        call    readarray
        pop     si
        mov     al,1              ;datos de 16 bits
        call    copmem            ;poner dirección de memoria
        cmp     bl,4
        jb      short find8
        mov     al,4
        and     bl,1
        sub     al,bl
find8:
        push    ax
        xor     al,al             ;almacenar 0
        stosb
        pop     ax
        retn
cased8todf      endp
```

```
casee0toe3      proc    near
        push    si
        mov     si,offset operjl
        mov     al,ah
        and     al,3
        xor     cx,cx           ;instrucciónJCXZ
        cmp     al,3
        jne     esloop
        call    readarray
        jmp     short pondir
esloop: inc     cx              ;Instrucción LOOP
        call    readarray
        test    ah,2
        jnz     pondir
        test    ah,1
        jz      esne
        mov     al,'E'          ;instrucción LOOPE
        stosb
        jmp     short pondir
esne:   mov     ax,'EN'         ;Instrucción LOOPNE
        stosw
pondir: pop     si
        xor     al,al
        mov     cx,2
        call    realadr         ;calcular y poner la dirección real del salto
        push    ax
        xor     al,al
        stosb
        pop     ax
        retn
casee0toe3      endp

casee4toe7      proc    near
        mov     al,ah
        and     al,1
        push    si
        mov     si,offset operoio
        test    ah,2
        jz      esin
        xor     cx,cx           ;OUT
        call    readarray
        pop     si
        mov     bx,l1
        add     bx,l2
        add     bx,l3
        mov     dx,ax
        mov     al,[si+bx+1]
```

```
        call    convhex         ;poner número de puerto
        mov     ax,'A,'
        cmp     l2,0            ;es dato de 32 bits?
        je      esa             ;no
        mov     ax,'E,'         ;si
        stosw
        mov     al,'A'          ;dato de 16 bits
        stosb
        jmp     short eseaxx
esa:    stosw
        or      dl,dl
        mov     al,'L'      ;es dato de 8bits
        jz      esal
eseaxx: mov     al,'X'
esal:   stosb
        jmp     short fine4toe7
esin:   mov     cx,1            ;Instrucción IN  ACC,Puerto
        call    readarray
        mov     dx,ax
        mov     al,'A'          ;dato de 16 bits
        cmp     l2,0            ;es dato de 32 bits?
        je      soloa           ;no
        mov     ax,'AE'         ;si
        stosw
        jmp     short coma
soloa:  stosb
coma:   or      dl,dl
        mov     al,'L'          ;dato de 8 bits
        jz      esregal
        mov     al,'X'
esregal:stosb
        mov     al,','          ;poner la coma
        stosb
        pop     si
        mov     bx,l1
        add     bx,l2
        add     bx,l3
        mov     al,[si+bx+1]
        call    convhex         ;poner el número de puerto
fine4toe7:
        xor     al,al
        stosb
        mov     ax,dx
        mov     cx,2
        add     cx,l2
        add     ax,l2
        retn
```

```
casee4toe7    endp

casee8toe9    proc   near
        push   si
        mov    si,offset operjc
        xor    cx,cx          ;instrrucción JMP  cerca
        test   ah,1
        jnz    esjmp
        inc    cx             ;instrucción CALL cercano
esjmp:  call   readarray
        pop    si
        mov    al,1
        mov    cx,3
        call   realadr        ;calcular y poner la dirección real del salto
        push   ax
        xor    al,al
        stosb
        pop    ax
        retn
casee8toe9    endp

caseea  proc   near
        push   si
        mov    si,offset operjc
        xor    cx,cx          ;JMP lejos
        call   readarray
        pop    si
        call   absoluto       ;poner número tipo ####:####
        push   ax
        xor    al,al
        stosb
        pop    ax
        retn
caseea  endp

caseeb  proc   near
        push   si
        mov    si,offset operjc
        xor    cx,cx          ;instrucción JMP corto
        call   readarray
        pop    si
        xor    al,al
        mov    cx,2
        call   realadr        ;calcular y poner dirección real del salto
        push   ax
        xor    al,al
        stosb
```

```
        pop     ax
        retn
caseeb  endp

casecctoef      proc    near
        mov     al,ah
        and     al,1
        mov     si,offset operoio
        test    ah,2
        jz      esin_
        mov     cx,2            ;OUT dx
        call    readarray
        mov     dx,ax
        mov     al,','
        stosb
        mov     al,'E'          ;poner EAX
        cmp     l2,0
        je      unicoa
        stosb
unicoa: mov     al,'A'
        stosb
        mov     al,'X'          ;poner AX
        or      dl,dl
        jnz     esrax
        mov     al,'L'          ;poner AL
esrax:  stosb
        jmp     short finectoef
esin_:  mov     cx,1            ;instrucción IN   ACC,DX
        call    readarray
        mov     dx,ax
        mov     al,'E'
        cmp     l2,0
        je      unicoa_
        stosb
unicoa_: mov    al,'A'
        stosb
        mov     al,'X'
        or      dl,dl
        jnz     esrgax
        mov     al,'L'
esrgax: mov     ah,','
        stosw
        mov     ax,'XD'
        stosw
finectoef:
        mov     cx,1
        xor     al,al
```

```
        stosb
        add     cx,l2
        mov     ax,dx
        add     ax,l2
        retn
caseectoef      endp

casef0  proc    near
        mov     si,offset operlk
        xor     cx,cx           ;LOCK
        call    readarray       ;sin argumentos
        xor     al,al
        stosb
        mov     al,255          ;longitud de datos no definida
        mov     cx,1
        retn
casef0  endp

casef1  proc    near
        call    incognita       ;longitud no reconocida
        retn
casef1  endp

casef2tof3      proc    near
        push    si
        xor     cx,cx           ;instrucción REPE
        test    ah,1
        jz      esrepne
        mov     bx,l1
        mov     al,[si+bx+1]
        and     al,0a6h
        cmp     al,0a6h
        je      finf2tof3
        inc     cx              ;instrucción REP
        jmp     short finf2tof3
esrepne:mov     cx,2            ;instrucción REPNE
finf2tof3:
        mov     si,offset operrrr
        call    readarray
        xor     al,al
        stosb
        pop     si
        mov     al,255
        mov     cx,1
        retn
casef2tof3      endp
```

```
casef4  proc   near
        mov    si,offset operhlt
        xor    cx,cx          ;instrucción HLT, sin argumentos
        call   readarray
        xor    al,al
        stosb
        mov    al,255         ;longitud de datos no específica
        mov    cx,1
        retn
casef4  endp

casef5  proc   near
        mov    si,offset opercmc
        xor    cx,cx          ;instrucción CMC, sin argumentos
        call   readarray
        xor    al,al
        stosb
        mov    al,1
        mov    cx,1
        retn
casef5  endp

casef6tof7     proc   near
        call   cod3llinc
        mov    cl,bl
        xor    ch,ch
        push   si
        mov    si,offset oper7 ;instrucciones
                               ;TEST,NOT,NEG,MUL,IMUL,DIV,IDIV
        call   readarray
        pop    si
        or     bl,bl
        jz     pondat
        mov    al,''
        stosb
        mov    dx,offset atbstr
        mov    bstring,dx
        mov    al,ah
        and    al,1
        call   modrm          ;halla el operando de acuero a MOD y R/M
        call   addbstr        ;poner el operando
        jmp    short finf6tof7
pondat: mov    dl,ah
        and    dl,1
        call   datadic        ;dato inmediato para Test
finf6tof7:
        push   ax
```

```
        xor     al,al
        stosb
        pop     ax
        add     ax,12
        retn
casef6tof7      endp

casef8tofd      proc    near
        mov     si,offset oper8  ;instrucción CLC,STC,CLI,STI,CLD,STD
        mov     cl,ah            ; sin operandos
        and     cx,7
        call    readarray
        xor     al,al
        stosb
        mov     al,1
        mov     cx,1
        retn
casef8tofd      endp

casefetoff      proc    near
        call    cod3l1inc
        push    si
        mov     si,offset oper9  ;instrucciones INC,DEC,CALL,CALL FAR,
                                 ;JMP,JMP FAR,PUSH
        mov     al,ah
        and     al,1
        jnz     normal
        cmp     bl,2
        jb      normal
        mov     cx,7
        jmp     short finfetoff
normal: mov     cl,bl
        xor     ch,ch
        xor     dh,dh
        cmp     cl,3
        je      espec    ;requieren operandos
        cmp     cl,5
        jne     finfetoff
espec:  mov     dh,2
finfetoff:
        call    readarray
        pop     si
        mov     dl,al
        mov     al,' '
        stosb
        mov     ax,offset atbstr
        mov     bstring,ax
```

```
        mov     ax,dx
        call    modrm       ;obtener operando de acuerdo a MOD y R/M
        call    addbstr
        xor     al,al
        stosb
        mov     ax,dx
        or      ah,ah
        jz      same
        mov     al,ah
same:   retn
casefetoff      endp


        public  desensDet
desensDet       proc    near
;-------------------------------------------------------------------------
;Este procedimiento genera un string a partir del código de máquina apuntado
;por DS:SI, desde ES:DI, terminado en 0, consistente en:
;       - Cuatro caracteres que representan al valor inicial de SI en hex.
;       - Cinco carateres del mnemotécnico de la instrucción.
;       - Veintiún caracteres de parámetros del mnemotécnico.
;Ademas, se retorna:
;       AL      tipo de instrucción (0,1,2,3,4 o 255)
;       CX      longitud de instrucción
;-------------------------------------------------------------------------
        push    di
        cld
        mov     ax,si
        push    ax
        mov     al,ah
        call    convhex ;poner el primer byte dedesplazamiento
        pop     ax
        call    convhex ;poner el segundo byte de dezplazamineto
        call    desens  ;obtiene el mnemotécico y los operandos
        push    ax
        push    bx
        push    cx
        push    dx
        push    si
        push    ds
        mov     bx,di           ;DI=comienzo del mnemotécnico
        mov     cx,8            ;buscar espacio en blanco hasta el octavo
        mov     al,' '          ;carácter
        repne   scasb           ;si no se lo encuentra hay parámetros
        je      ConParam
        mov     di,bx
        mov     cx,8            ;busco un 0 hasta el octavo carácter
```

```
        mov     al,0
        repne   scasb
        jne     ConParam        ;si no se lo encuentra, hay parámetros
        add     cx,38           ;poner 38 espacios al final del unico mnemo
        mov     al,' '
        dec     di
        rep     stosb
        mov     al,0            ;poner un 0 al final
        stosb
        jmp     FinDesDet
ConParam:
        mov     di,bx           ;recuperar el comienzo del mnemotécnico
        mov     cx,8            ;buscar un espacio
        mov     al,' '
        repne   scasb
        cmp     cx,1            ;en que posición esta el espacio?
        jb      sexta           ;si CX=0, octava posición; mover parámetros
                                   ;una posición a la izquierda
        je      PonSpcFin       ;si CX=1, séptima posición, no se deben mover
        mov     cx,di           ;los parámetros
        mov     ax,es           ;ci CX>=2, sexta o menos.
        mov     ds,ax           ;en este caso, mover los parámetros a la
        sub     di,bx           ;derecha
        mov     dx,7
        sub     dx,di           ;DX=número de posiciones por mover a la derecha
        mov     di,bx
        mov     bx,cx           ;BX=comienzo de los párametros
        mov     cx,43
        mov     al,0
        repne   scasb           ;buscar el fin de string
        dec     di
        mov     si,di
        add     di,dx
        std                     ;dirección inversa
CicDer: lodsb
        stosb
        cmp     si,bx           ;se llego al comienzo de los parámetros?
        jae     CicDer          ;mover el string
        mov     al,' '
CicSpc: cmp     di,si
        je      PonSpcFin
        stosb
        jmp     short CicSpc

Sexta:  mov     ax,es           ;mover los parámetros una posición a la izq.
        mov     ds,ax
        mov     si,di
```

```
        dec     di
CicIzq: lodsb
        stosb
        cmp     al,0
        jne     CicIzq

PonSpcFin:
        cld
        mov     di,bx
        mov     cx,54
        mov     al,0    ;buscar la posición del cero
        repne   scasb
        dec     di
        mov     al,' '  ;poner espacio en blanco
        rep     stosb
        mov     al,0
        stosb           ;poner un cero

FinDesDet:
        pop     ds
        pop     si
        pop     dx
        pop     cx
        pop     bx
        pop     ax
        pop     di
        retn
DesensDet       endp
        end
```

MODULO ENSAMBLA.ASM

```
;***************PROGRAMA ENSAMBLADOR*****************

        .model  small
        .486


;_____________________________________________
;       AREA PARA DATOS
;_____________________________________________

        .data

transit db      32 dup (?)

mnemo   db      'ADd',0,'PUSh',6,'POp',7,'Or',8
        db      'ADc',10H,'SBb',18H,'ANd',20H,'DAa',27H
        db      'SUb',28H,'DAs',2FH,'XOr',30H,'AAa',37H
        db      'CMp',38H,'NEAr',39H,'FAr',3AH,'PTr',3BH
        db      'BYTe',3CH,'WORd',3DH,'DWORd',3EH,'AAs',3FH
        db      'INc',40H,'DEc',48H,'SHORt',50H,'QWORd',51H,'NOt',52H,'NEg',53H
        db      'MUl',54H,'IMUl',55H,'DIv',56H,'IDIv',57H

        db      'JNAe',72H,'JNb',73H,'Jb',72H,'JAe',73H
        db      'Jz',74H,'JNz',75H,'JNa',76H,'JNBe',77H
        db      'JPe',7AH,'JPo',7BH,'JNGe',7CH,'JNl',7DH
        db      'JNg',7EH,'JNLe',7FH,'Jo',70H,'JNo',71H
        db      'Jc',72H,'JNc',73H,'Je',74H,'JNe',75H
        db      'JBe',76H,'Ja',77H,'Js',78H,'JNs',79H
        db      'Jp',7AH,'JNp',7BH,'Jl',7CH,'JGe',7DH
        db      'JLe',7EH,'Jg',7FH,'Db',80H,'Dw',81H

        db      'TESt',84H,'XCHg',86H,'MOv',88H,'LEa',8DH
        db      'NOp',90H,'CBw',98H,'CWd',99H,'CALl',9AH,'WAIt',9BH
        db      'PUSHf',9CH,'POPf',9DH,'SAHf',9EH,'LAHf',9FH
        db      'MOVSb',0A4H,'MOVSw',0A5H,'CMPSb',0A6H,'CMPSw',0A7H
        db      'STOSb',0AAH,'STOSw',0ABH,'LODSb',0ACH,'LODSw',0ADH
        db      'SCASb',0AEH,'SCASw',0AFH,'REt',0C2H,'RETn',0C2H
        db      'LEs',0C4H,'LDs',0C5H,'RETf',0CAH,'INt',0CCH
        db      'INTo',0CEH,'IREt',0CFH,'ROl',0D0H,'ROr',0D1H

        db      'RCl',0D2H,'RCr',0D3H,'AAm',0D4H,'AAd',0D5H
```

```
        db    'XLAt',0D7H,'ESc',0D8H,'SHl',0DCH,'SHr',0DDH
        db    'SAl',0DCH,'SAr',0DFH,'LOOPNe',0E0H,'LOOPNz',0E0H
        db    'LOOPe',0E1H,'LOOPz',0E1H,'LOOp',0E2H,'JCXz',0E3H
        db    'In',0E4H,'OUt',0E6H,'JMp',0EAH,'LOCk',0F0H
        db    'REPNe',0F2H,'REPNz',0F2H,'REp',0F3H,'REPe',0F3H,'REPz',0F3H
        db    'HLt',0F4H,'CMc',0F5H,'CLc',0F8H,'STc',0F9H
        db    'CLi',0FAH,'STi',0FBH,'CLd',0FCH,'STd',0FDH
        db    0FFH

regs    db    'ALCLDLBLAHCHDHBH'
        db    'AXCXDXBXSPBPSIDI'
        db    'EAXECXEDXEBXESPEBPESIEDI'

sregs   db    'ESCSSSDSFSGS'

loners  db    27H,2FH,37H,3FH,90H,98H,99H,9BH,9CH,9DH,9EH,9FH
        db    0A4H,0A5H,0A6H,0A7H,0AAH,0ABH,0ACH,0ADH,0AEH,0AFH
        db    0CEH,0CFH,0D4H,0D5H,0D7H
        db    0F4H,0F5H,0F8H,0F9H,0FAH,0FBH,0FCH,0FDH

oners   db    6,7,40H,48H,52H,53H,54H,55H,56H,57H
        db    70H,71H,72H,73H,74H,75H,76H,77H
        db    78H,79H,7AH,7BH,7CH,7DH,7EH,7FH
        db    9AH,0C2H,0CAH,0CCH,0E0H,0E1H,0E2H,0E3H,0EAH

;_____________________________________________
;       AREA PARA CODIGO
;_____________________________________________
        .code

NEsp    equ   10
delspc  proc  near
;---------------------------------------------------------------
;Este procedimiento salta los espacios y los TABs en el string por
;ensamblar y deja SI apuntando al primer byte que no sea un espacio.
;
;Entrada:    DS:SI  apunta al punto del string desde el cual
;              se desea proceder.
;Salida:     DS:SI  apunta al primer byte que no es un espacio.
;            CY     es 1 si hay más de NEsp espacios (error), en
;              cuyo caso DS:SI no es alterado
;---------------------------------------------------------------
        push   ax
        push   cx
        push   bp
        mov    cx,NEsp          ;máximo NEsp espacios
        mov    bp,si
```

```
busc:   lodsb
        or      al,al
        jz      errdel          ;si se llega al final del string
        cmp     al,' '          ;hay un espacio?
        je      spac            ;si, saltarlo
        cmp     al,9            ;hay un TAB?
        jne     encont          ;no, hecho
spac:   loop    busc            ;saltar el espacio y ver sig. car.

errdel: mov     si,bp
        stc
        jmp     short findl
encont: dec     si              ;apuntar al carácter válido
        clc
findl:  pop     bp
        pop     cx
        pop     ax
        retn
delspc  endp

encnem  proc    near
;---------------------------------------------------------------------
;Este procedimiento encuentra el pseudocódigo del mnemotécnico que está al
;comienzo del string por ensamblarse.
;
;Entrada:    DS:SI apunta al string, que debe terminar en 0.
;            ES:DI es la dirección en donde se pondrá el pseudocódigo.
;
;Salida:     DS:SI apunta al final del mnemotécnico + 2 en el string si
;              este fue encontrado (pues debe terminar con espacio)
;            ES:DI apunta al final del pseudocódigo + 1
;            CY    es 1 si el mnemotécnico es inválido o si hay error
;              como en DELSPC , en cuyo caso DS:SI y ES:DI no son
;              alterados (excepto si había espacios desde DS:SI)
;            AX    contiene el pseudocódigo (AL=0, AH= cod.)
;---------------------------------------------------------------------
        call    delspc          ;encontrar el mnemotécnico
        jc      finenc2         ;en caso de error, terminar
        push    bx
        push    dx
        push    es
        push    di
        mov     ax,@data
        mov     es,ax           ;hacer que ES:DI apunte a los mnemotécnicos
        mov     di,offset mnemo
        mov     dx,si           ;preservar el comienzo del mnemotécnico
```

```
        pop     es
finenc1:pop     dx
        pop     bx
finenc2:retn
encnem  endp

        public  encreg
encreg  proc    near
;---------------------------------------------------------------------
;Este procedimiento encuentra el código del registro al que apunta DS:SI.
;
;Entrada:       DS:SI apunta al ascii del registro
;
;Salida:        DS:SI es cambiado segun DELSPC
;               AH    es alterado
;               AL    código del registro
;               CY    1 si no es un registro o si hay error de DELSPC
;---------------------------------------------------------------------
        call    delspc
        jc      finencreg1      ;si hay demasiados espacios, fin
        push    cx
        push    bx
        push    es
        push    di
        mov     ax,@data
        mov     es,ax
        mov     di,offset regs  ;apuntar ES:DI a la tabla de registros
        mov     Eax,[si]        ;LO DIGITADO ascii del reg en AX
        and     Eax,0dfdfdfh        ;en mayúsculas
        mov     cx,24
srchreg:mov     Ebx,es:[di]             ;de la tabla
        cmp     cx,8
        jbe     rup0
        cmp     ax,bx
        je      foundreg
        inc     di
        inc     di
        loop    srchreg
        stc                     ;si no se lo encontró, error
        jmp     short errreg
rup0:   and     ebx,0ffffffh
        cmp     eax,ebx
        je      foundreg
        inc     di
        inc     di
        inc     di
        loop    srchreg
```

```
malsreg:stc                ;si no se lo encontró, error
        jmp    short errsreg
foundsreg:
        mov    al,[si+2]     ;tercer carácter
        and    al,0dfh       ;hacerlo mayuscúla
        cmp    al,'A'        ;si es una letra, mal
        jb     oksreg
        cmp    al,'Z'
        jbe    malsreg
oksreg: mov    al,6
        sub    al,cl         ;encontrar el código del registro
        or     al,40h
        clc
errsreg:pop    di
        pop    es
finencsreg:
        pop    cx
finencsreg1:
        retn
encsreg endp

ascahex proc   near
;-------------------------------------------------------------------
;Este procedimiento transforma el string en AX (con AL = MSD) a un byte
;en AL, CY = 1 si no hay tal hex en AX.
;-------------------------------------------------------------------
        push   cx
        cmp    ah,'f'
        ja     errhex
        cmp    ah,'a'
        jb     tstal
        and    ah,0dfh       ;transformar el carácter en AH a mayúscula
tstal:  cmp    al,'f'
        ja     errhex
        cmp    al,'a'
        jb     trns
        and    al,0dfh       ;transformar el carácter en AL a mayúscula
trns:   cmp    ah,'A'        ;ajustar el carácter en AH si es alfabético
        jb     numah
        sub    ah,'A'-'9'-1
numah:  cmp    al,'A'        ;lo mismo con AL
        jb     numal
        sub    al,'A'-'9'-1
numal:  sub    ax,'00'       ;pasar de ascii a numérico
        cmp    al,16
        jae    errhex
        cmp    ah,16
```

```
        jae     errhex
        mov     cl,4
        shl     al,cl
        or      al,ah           ;condensar el número
        clc
        jmp     short finhex
errhex: stc
finhex: pop     cx
        retn
ascahex endp

esletr? proc    near
;---------------------------------------------------------------------
;Este procedimiento determina si el carácter contenido en AL es un dígito
;o una letra, en cuyo caso CY = 0, y de lo contrario, CY = 1.
;AL es alterado.
;---------------------------------------------------------------------
        cmp     al,'0'
        jb      noeslt
        cmp     al,'9'
        jbe     sieslt
        and     al,0dfh         ;mayúscula
        cmp     al,'A'
        jb      noeslt
        cmp     al,'Z'
        jbe     sieslt
noeslt: stc
        retn
sieslt: clc
        retn
esletr? endp

        public  eshex?
eshex?  proc    near
;---------------------------------------------------------------------
;Este procedimiento determina si el string al que apunta DS:SI es un número
;hexadecimal, en cuyo caso encuentra su valor.
;
;Entrada:       DS:SI apunta al string, excluyendo signos +, - o espacios
;
;Salida:    AL  = 16 si el número es de 1 o 2 digitos, 32 en otro caso
;           AH  = longitud del string del número
;           BX  = valor del número
;           CY  = 1 si lo apuntado por DS:SI no es un número o si es un
;                 número de mas de 32 bits
;---------------------------------------------------------------------
```

```
        push    dx
        xor     bx,bx
trycero:cmp     byte ptr[si+bx],'0' ;buscar ceros iniciales
        jne     noceroin
        inc     bx              ;contarlos en BX
        jmp     short trycero
noceroin:
        mov     dh,bl           ;número de ceros en DH
        or      bx,bx
        jz      noproblem       ;si no hay ceros iniciales
        mov     ah,[si+bx]
        mov     al,'0'
        call    ascahex         ;si el string era solo ceros, ya no debe
        jnc     noproblem       ;haber mas caracteres hexadecimales válidos
        mov     ax,16
        xor     bx,bx           ;salir con cero
        jmp     finnum
noproblem:
        mov     al,[si+bx+1]
        call    esletr?
        jc      undig
        mov     al,[si+bx+2]
        call    esletr?
        jc      dosdig
        mov     al,[si+bx+3]
        call    esletr?
        jc      tresdig

        mov     al,[si+bx+4]
        call    esletr?             ;tiene más de 4 dígitos?
        jc      N4Dig               ;no, OK
        stc                         ;si, error
        jmp     short errnum
N4Dig:  mov     ax,[si+bx]          ;el número tiene 4 dígitos
        call    ascahex
        jc      errnum
        mov     dl,al
        mov     ax,[si+bx+2]
        call    ascahex
        jc      errnum
        mov     bl,al
        mov     bh,dl
        mov     ax,420h         ;AL=32, AH=4
        jmp     short finnum
tresdig:mov     al,'0'          ;número de tres dígitos
        mov     ah,[si+bx]
        call    ascahex
```

```
        jc      errnum
        mov     dl,al
        mov     ax,[si+bx+1]
        call    ascahex
        jc      errnum
        mov     bl,al
        mov     bh,dl
        mov     ax,320h         ;AL=32, AH=3
        jmp     short finnum
dosdig: mov     ax,[si+bx]              ;número de dos digitos
        call    ascahex
        jc      errnum
        mov     bl,al
        xor     bh,bh
        mov     ax,210h         ;AL=16, AH=2
        jmp     short finnum
undig:  mov     al,'0'
        mov     ah,[si+bx]
        call    ascahex
        jc      errnum
        mov     bl,al
        xor     bh,bh
        mov     ax,110h         ;AL=16, AH=1
finnum: add     ah,dh           ;incluir los ceros en la longitud
        cmp     ah,2
        jbe     longbyt
        mov     al,20h
longbyt:clc
errnum: pop     dx
        retn
eshex?  endp

esreg?  proc    near
;-----------------------------------------------------------------------
;Este procedimiento es similar al anterior: encuentra si el string al que
;apunta DS:SI es uno de los registros BX, BP, SI o DI.
;
;Entrada:       DS:SI apunta al string excluyendo signos (+ o -) y espacios
;
;Salida:    CY =1 si hay error, caso contrario:
;           AH = 2 (longitud obvia del string)
;           AL = 1 si SI
;                2 si DI
;                4 si BX
;                8 si BP
;-----------------------------------------------------------------------
        mov     ax,[si]
```

```
        shl     al,cl
        mov     dh,al          ;en dh
        call    delspc
        jc      errm1
        lodsb
        cmp     al,':'         ;hay los dos puntos reglamentarios?
        jne     errm1          ;no
        call    delspc
        jnc     corchetes
errm1:  jmp     errmem
nopref: xor     dh,dh          ;codificar inexistencia de sreg
corchetes:
        xor     dl,dl          ;inicializar primera parte del código
        xor     bx,bx          ;inicializar desplazamiento
        mov     cx,bx
        lodsb
        cmp     al,'['         ;el carácter actual debe ser el
        jne     errm1          ;corchete reglamentario
        call    delspc
        jc      errm1
        cmp     byte ptr[si],'+';inicia con un signo +?
        jne     tstmin         ;no
        inc     si             ;si, ignorarlo
        call    delspc
        jc      errm1
analiz: call    eshex?         ;hay un número hex?
        jc      noh1           ;no
        mov     cx,bx          ;si, preservar su valor
        jmp     short rdynum
noh1:   call    esreg?         ;hay un registro reglamentario?
        jnc     rdynum         ;si
errm2:  jmp     errmem         ;no, error
tstmin: cmp     byte ptr[si],'-';inicia con un signo -?
        jne     analiz         ;no
        inc     si             ;si, lo siguiente solo puede ser un número
        call    delspc
        jc      errm2
        call    eshex?
        jc      errm2
        neg     bx             ;encontrar el negativo
        mov     cx,bx          ;preservarlo
rdynum: xor     dl,al          ;poner el bit correspondiente en DL
        test    dl,al          ;estaba ya puesto, implicando duplicación?
        jz      errm2          ;si, error
        mov     al,ah          ;longitud del string
        xor     ah,ah
        add     si,ax          ;ajustar SI
```

```
        jnc     rdynum3
errm4:  jmp     errmem
tstmin3:cmp     al,'-'
        jne     errm4
        call    delspc
        jc      errm4
        call    eshex?          ;despues de un - solo puede haber un número
        jc      errm4
        neg     bx
        mov     cx,bx
rdynum3:xor     dl,al           ;poner el bit apropiado
        test    dl,al           ;duplicación?
        jz      errm4             ;si, error
        mov     al,ah           ;longitud leída hasta ahora
        xor     ah,ah
        add     si,ax           ;ajustar SI
        call    delspc
        jc      errm4

        lodsb                   ;ya se ha leído 3 cosas, debe haber
        cmp     al,']'          ;un corchete finalizador
        jne     errm4
condensar:
        mov     al,dl           ;determinar si el código
        and     al,3            ;en DL es incongruente
        cmp     al,3            ;si DL = 00FEDCBA, entonces
        je      errmem            ;error = ~A~B~C~D~E~F + AB + CD + EF
        mov     al,dl
        and     al,12
        cmp     al,12
        je      errmem
        mov     al,dl
        and     al,30h
        cmp     al,30h
        je      errmem
        mov     al,dl
        and     al,3fh
        jz      errmem

        xor     ah,ah           ;inilicializar código condensado
        test    dl,2            ;bit0 = ~AC + B
        jnz     sb0
        mov     al,dl
        xor     al,1
        and     al,5
        cmp     al,5
        jne     bit1
```

```
sb0:    or      ah,1
bit1:   test    dl,8            ;bit1 = ~A~B + D
        jnz     sb1
        mov     al,dl
        and     al,3
        jnz     bit2
sb1:    or      ah,2
bit2:   mov     al,dl           ;bit2 = ~A~B + ~C~D
        and     al,3
        jz      sb2
        mov     al,dl
        and     al,12
        jnz     bit3
sb2:    or      ah,4
bit3:   mov     al,dl           ;bit3 = ~A~B~CD~F + (A + B + C)E
        xor     al,27h
        and     al,2fh
        cmp     al,2fh
        je      sb3
        test    dl,16
        jz      bit4
        mov     al,dl
        and     al,7
        jz      bit4
sb3:    or      ah,8
bit4:   test    dl,32           ;bit4 = (A + B + C + D)F
        jz      fincond
        and     dl,15
        jz      fincond
        or      ah,16
fincond:mov     dl,ah
        mov     ax,dx
        or      al,60h
        stosw
        mov     ax,cx
        stosw
        clc
        jmp     short finmem
errmem: mov     si,bp
        stc
finmem: pop     bp
        pop     dx
        pop     cx
        pop     bx
        pop     ax
        retn
encmem  endp
```

```
        stosb
        mov     ax,bx
        stosw
        jmp     short noerrhex
unbyte: mov     ah,bl
        stosw
noerrhex:
        clc
finenchex:
        pop     dx
        pop     bx
        pop     ax
        retn
enchex  endp


enspaso1        proc    near
;-------------------------------------------------------------------------
;Este procedimiento ejecuta el primer paso de compilación, es decir, la
;traducción de un string alfanumérico a seudocódigos que serán transformados
;a código verdadero por otro procedimiento.
;
;Entrada:  DS:SI   apunta al string por compilarse, que debe terminar
;                  en 0.
;          ES:DI   apunta al lugar donde se desea el seudocódigo
;
;Salida:   DS:SI   es alterado
;          ES:DI   es alterado
;          CY      set si hay error
;-------------------------------------------------------------------------
        push    ax
        push    bx
        push    cx
        push    dx
        call    encsreg         ;es la instrucción un prefijo como ES: ?
        jc      nosregpref      ;no
        inc     si              ;si, proceder en consecuencia
        inc     si
        call    delspc
        mov     ah,al           ;preservar código
        lodsb
        cmp     al,':'          ;por sintaxis, debe haber :
        jne     errpaso1
        mov     al,ah
        stosb
        jmp     short bienpaso1
nosregpref:
```

```
        call    encnem          ;lo primero debe ser un mnemotécnico
        jc      errpaso1
        cmp     ah,0f0h         ;es un prefijo REP o LOCK?
        jb      nopreflr
        cmp     ah,0f3h
        ja      nopreflr
        call    delspc          ;si, buscar siguiente mnemotécnico
        jc      bienpaso1       ;si no hay más que el prefijo, terminar
        call    encnem
        jc      errpaso1
        cmp     ah,0f0h         ;es un segundo prefijo REP o LOCK?
        jb      nopreflr
        cmp     ah,0f3h
        ja      nopreflr
        call    delspc          ;si, buscar siguiente mnemotécnico
        jc      bienpaso1       ;si no hay mas que el segundo prefijo, terminar
        call    encnem
        jc      errpaso1
        cmp     ah,0f0h         ;no puede haber un tercer prefijo
        jb      nopreflr
        cmp     ah,0f3h
        jbe     errpaso1
nopreflr:
        cmp     ah,39h          ;el mnemotécnico inicial no puede ser NEAR,
        jb      bienmne         ;PTR, BYTE, etc
        cmp     ah,3eh
        jbe     errpaso1
        cmp     ah,50h
        je      errpaso1
        cmp     ah,51h
        je      errpaso1

bienmne:mov     dx,ds           ;preservar DS
        mov     bx,@data
        mov     ds,bx
        mov     bx,offset loners
        mov     cx,35           ;averiguar si el mnemotécnico es uno de los
lone?:  cmp     ah,[bx]         ;que no aceptan argumentos
        je      esloner         ;si
        inc     bx
        loop    lone?
        jmp     short hayarg
esloner:mov     ds,dx
        mov     byte ptr es:[di-2],1    ;marcarlo como loner
                                ;apuntar al punto inmediatamente despues
        call    delspc          ;del mnemotécnico y buscar el fin de string
        jnc     errpaso1        ;si hay mas caracteres válidos, error
```

```
bienpaso1:
        mov     al,0ffh
        stosb
        clc
        jmp     short finpaso1
errpaso1:
        stc
finpaso1:
        pop     dx
        pop     cx
        pop     bx
        pop     ax
        retn

hayarg: mov     bx,offset oners
        mov     cx,35           ;averiguar si el mnemotécnico es uno de los
one?:   cmp     ah,[bx]         ;que aceptan un solo argumento
        je      esoner          ;si
        inc     bx
        loop    one?
        jmp     short dosarg
esoner: mov     byte ptr es:[di-2],2    ;marcarlo como oner
        cmp     ah,9ah          ;CALL?
        je      admitedp        ;si, puede haber dos puntos
        cmp     ah,0eah         ;JMP?
        jne     solouno         ;no
admitedp:
        or      byte ptr es:[di-2],4    ;marcarlo como JMP o CALL
        mov     cl,2            ;marcar que puede haber un argumento doble
        jmp     short args      ;de dirección absoluta
solouno:
        mov     cl,1            ;marcar que debe haber solo un argumento
        jmp     short args
dosarg: xor     cl,cl           ;marcar que debe haber dos argumentos
args:   mov     ds,dx
        call    delspc
        jnc     enarg           ;confirmar presencia de argumentos
        cmp     ah,0c0h         ;su no presencia solo es admisible con
        je      bienpaso1       ;RET, RETN o RETF
        cmp     ah,0c2h
        je      bienpaso1
        cmp     ah,0cab
        je      bienpaso1
        jmp     short errpaso1
enarg:  call    encmem          ;buscar argumentos
        jnc     segarg
        call    encreg
```

```
        jc      noreg1
        stosb
        inc     si
        inc     si
        jmp     short segarg
noreg1: call    encsreg
        jc      nosreg1
        stosb
        inc     si
        inc     si
        jmp     short segarg
nosreg1:call    enchex
        jnc     segarg
        call    encnem          ;buscar por ejemplo BYTE PTR
        jc      errpaso1
        call    encnem          ;el PTR es opcional
        call    encmem
        jnc     segarg
        call    enchex          ;puede también ser un número (para saltos)
        jc      errpaso1
segarg: or      cl,cl           ;dos argumentos?
        jz      haydos          ;si
        call    delspc          ;no, uno solo
        jc      bien1
        cmp     cl,2            ;era JMP o CALL?
        jne     errp1           ;no
        lodsb                   ;solo se admiten dos argumentos aquí si están
        cmp     al,':'          ;separados por :
        jne     errp1
        call    enchex
        jc      errp1
        jmp     bienpaso1

haydos: call    delspc
        jc      errp1           ;si hay un solo argumento, error
        lodsb
        cmp     al,','          ;el separador entre argumentos es la coma
        jne     errp1
        call    encmem          ;buscar segundo argumento
        jnc     bien1
        call    encreg
        jc      noreg2
        stosb
        inc     si
        inc     si
bien1:  jmp     bienpaso1
noreg2: call    encsreg
```

```
        push    cx
        lodsb
        cmp     al,0ffh         ;fin de instrucción?
        je      fininst
        mov     ch,ah
        mov     ah,al
        mov     cl,5
        shr     al,cl           ;obtener el tipo
        mov     dl,al           ;en DL
        jz      esinstr
        dec     al
        jz      esregis
        dec     al
        jz      essregi
        dec     al
        jz      esmemor
        dec     al
        jz      esdatab
        lodsw                   ;es dato de word
        jmp     short rdytipo
esdatab:lodsb                   ;es dato de byte
        xor     ah,ah
        jmp     short rdytipo
esinstr:lodsb                   ;es mnemotécnico
        jmp     short rdytipo
esregis:mov     al,ah           ;es registro
        and     al,0fh
        mov     ah,ch
        jmp     short rdytipo
essregi:mov     al,ah           ;es registro de segmento
        and     al,3
        mov     ah,ch
        jmp     short rdytipo
esmemor:mov     al,ah           ;es memoria
        mov     cl,3
        shl     al,cl
        and     al,0c0h
        and     ah,7
        or      ah,al
        lodsb
        xchg    al,ah
        mov     bx,ax
        lodsw
        mov     cx,4
rdytipo:clc
        jmp     short fintipo
fininst:stc
```

```
fintipo:pop     cx
        retn
tipo    endp

getpref proc    near
;-------------------------------------------------------------------------
;Este procedimiento obtiene el prefijo de redefinición de segmento a partir
;del número de registro de segmento.
;
;Entrada:       BH      = xPSrxxxx con P = presencia de prefijo
;
;Salida:        AH      = 0 si no hay prefijo, o
;                         prefijo
;-------------------------------------------------------------------------
        xor     ah,ah
        test    bh,40h
        jz      fingtp
        mov     ah,bh
        shr     ah,1
        and     ah,18h
        or      ah,26h
fingtp: retn
getpref endp
pref?   proc    near
;-------------------------------------------------------------------------
;Este procedimiento determina si debe incluirse un prefijo de redefinición
;de segmento antes del opcode, del cual solo debe haber 1 byte, y ES:DI
;debe estar apuntando a la dirección siguiente a dicho byte. Hace la
;inclusión y los ajustes del caso de ser necesario.
;
;Entrada:       AH      0 si no debe haber prefijo
;                       opcode del prefijo en caso contrario
;               ES:DI   dirección del byte de opcode + 1
;
;Salida:        AL      es inalterado o contiene el opcode
;               DI      es ajustado
;               CX      incrementado en 1 si hay prefijo
;-------------------------------------------------------------------------
        cmp     ah,0            ;hay prefijo?
        je      noprefijo
        mov     al,ah           ;sí, ajustar
        xchg    al,es:[di-1]
        stosb
        inc     cx
noprefijo:
        retn
pref?           endp
```

```
sizedisp        proc    near
;-------------------------------------------------------------------------
;Determina dados los bits 6 y 7 de AL, que debe tener el byte de modregr/m,
;el tamano en bytes del desplazamiento u offset. Luego incluye este
;desplazamiento en el opcode.
;
;Entrada:       DL o DX desplazamiento
;               AL      byte de modo de direccionamiento
;               ES:DI   apunta al lugar del opcode donde va el desplazamiento
;
;Salida:        AX      es alterado
;               ES:DI   es ajustado
;               CX      incrementado segun el desplazamiento
;-------------------------------------------------------------------------
        test    al,0c0h         ;obtener el mod
        jz      despl?          ;si es 00, averiguar si hay desplazamiento
        and     al,0c0h
        cmp     al,0c0h         ;es 11?
        je      nodesp          ;si, tampoco hay desplazamiento
        inc     cx
        cmp     al,40h          ;es 01?
        jne     dosbyt          ;no
        mov     al,dl           ;si, desplazamiento de un byte
        stosb
        jmp     short nodesp
despl?: and     al,7            ;si mod = 00 y r/m = 110,
        cmp     al,6            ;el modo es directo
        jne     nodesp
dosbyt: mov     ax,dx           ;desplazamiento de dos bytes
        stosw
        inc     cx
nodesp: retn
sizedisp        endp

sizedata        proc    near
;-------------------------------------------------------------------------
;Similar al anterior: determina el tamaño del dato segun el bit 0 de AL, y lo
;coloca en el opcode.
;
;Entrada:       AL      su bit 0 es w
;               BP      dato
;               ES:DI   apunta al lugar del opcode donde debe ir el dato
;
;Salida:        AX      es alterado
;               DI      es ajustado
;               CX      incrementado segun los datos
;-------------------------------------------------------------------------
```

```
        inc     cx
        test    al,1            ;w = 1?
        mov     ax,bp
        jnz     dosdata         ;si
        stosb                   ;no
        jmp     short fsdata
dosdata:inc     cx
        stosw
fsdata: retn
sizedata        endp

modrm   proc    near
;-----------------------------------------------------------------
;Este procedimiento analiza el pseudocódigo de modo de direccionamiento y
;genera datos necesarios para la compilación.
;
;Entrada:       DS:SI   debe apuntar al pseudocódigo de modo de dir.
;
;Salida:        AH      prefijo de redefinición, de haberlo, o 0
;               BL      = Md000r/m
;               BH      = h000000w; h=validez de w
;               DX      desplazamiento, de haberlo; si no, es alterado
;               CL      incrementado en 1 si no hay error
;               CH      = 0
;               CY      set si hay error, en cuyo caso los registros
;                       quedan inalterados
;-----------------------------------------------------------------
        push    bx
        push    cx
        push    dx
        push    si
        push    bp
        push    ax
        lodsb
        dec     si
        test    al,0e0h         ;mnemo?
        jnz     mrmnomne        ;no
        call    tipo
        jc      errmodrm
        cmp     al,3ch          ;BYTE?
        jb      errmodrm
        je      bytptr
        cmp     al,3eh          ;WORD o DWORD?
        ja      errmodrm
        mov     dh,81h          ;marcar word
        jmp     short elptr
bytptr: mov     dh,80h          ;marcar byte
```

```
        pop     bp
        pop     si
        pop     dx
        pop     cx
        pop     bx
        stc
        retn
modrm   endp

modregrm        proc    near
;-----------------------------------------------------------------------
;Este procedimiento genera datos necesarios para el ensamblado de instrucciones
;que tienen dos operadores (como ADD), dados los pseudocódigos generados en el
;primer paso de compilación.
;
;Entrada:       DS:SI   debe apuntar a los pseudocódigos de los operadores.
;
;Salida:    AL      = 000000dw
;           AH      prefijo de redefinición, de haberlo, o 0
;           BL      byte de modo de direccionamiento
;           DX      desplazamiento de haberlo, si no, es alterado
;           CX      incrementado en 1 si no hay error
;           CY      set si hay error, en cuyo caso los registros quedan
;                   inalterados
;-----------------------------------------------------------------------
        push    ax
        push    cx
        push    dx
        push    si
        push    bp
        push    bx
        lodsb
        dec     si
        and     al,0e0h
        cmp     al,20h          ;reg?
        jne     mrmnoreg
        call    tipo
        call    modrm
        jc      errmrr
        call    arregla
        jc      errmrr
        or      al,2            ;d=1
        jmp     short bienmrr
mrmnoreg:
        call    modrm
        jc      errmrr
        mov     bp,dx
```

```
errarr: stc
        retn
modregrm        endp

modrmdata       proc    near
;-----------------------------------------------------------------------
;Similar al anterior, cuando el segundo operador es un dato inmediato. Produce
;resultados diferentes si el primer operador es un acumulador.
;
;Entrada:       AL      = xxCODxxx
;               DS:SI   debe apuntar a los pseudocódigos de los operadores
;
;Salida:        AL      = 000000sw
;               AH      prefijo de redefinición, de haberlo, o 0
;               BL      = md000r/m OR COD
;               DX      desplazamiento, de haberlo, si no, es alterado
;               BP      data
;               CX      incrementado en 1 si no hay error
;                       o:
;               AL      = 0000010w
;               BP      data
;               CX      inalterado
;
;               CY      set si hay error, en cuyo caso los registros quedan
;                       inalterados
;-----------------------------------------------------------------------
        push    cx
        push    dx
        push    si
        push    bp
        push    bx
        push    ax
        mov     ch,al
        lodsb
        dec     si
        mov     ah,al
        and     al,20h          ;reg?
        cmp     al,20h
        jne     noac
        test    ah,7
        jnz     noac
        lodsb
        mov     bl,al
        call    tipo
        cmp     dl,4            ;dato?
        jb      errmrd
        je      mrdok
```

```
        test    bl,8
        jnz     mrdok
        cmp     ax,255
        ja      errmrd
mrdok:  mov     bp,ax
        mov     al,bl
        and     al,8
        shr     al,1
        shr     al,1
        shr     al,1
        or      al,4
        mov     bl,al
        pop     ax
        mov     al,bl
        add     sp,6
        pop     dx
        pop     cx
        retn
noac:   mov     al,ch
        call    modrm
        jc      errmrd
        mov     ch,al
        mov     al,dl
        mov     bp,ax
        call    tipo
        cmp     dl,4            ;dato?
        jb      errmrd
        jne     nosext
        or      bh,2            ;sign extend: s=1
nosext: test    bh,80h          ;definido size?
        jz      mrdnoesp
        test    bh,1
        jnz     mrdmok
        cmp     dl,5
        jne     mrdmok
        cmp     ax,255
        ja      errmrd
        jmp     short mrdmok
mrdnoesp:
        and     bh,2
        or      bh,80h
        cmp     dl,5
        jne     mrdmok
        or      bh,1
mrdmok: xchg    bp,ax
        mov     dl,al
        and     ch,38h
```

```
        or      bl,ch
        mov     al,bh
        and     al,3
        add     sp,2
        mov     ch,bl
        pop     bx
        mov     bl,ch
        xor     ch,ch
        add     sp,8
        clc
        retn
errmrd: pop     ax
        pop     bx
        pop     bp
        pop     si
        pop     dx
        pop     cx
        stc
        retn
modrmdata       endp

delptr  proc    near
;---------------------------------------------------------------------
;El mnemotécnico PTR es innecesario en todos los casos. Si DS:SI apunta
;al pseudocódigo de este mnemotécnico, este procedimiento lo salta y deja
;a SI apuntando al siguiente pseudocódigo; en otro caso, no hace nada.
;---------------------------------------------------------------------
        push    ax
        push    bx
        push    dx
        push    bp
        mov     bp,si
        call    tipo
        jc      noptr
        or      dl,dl           ;mnemo?
        jnz     noptr
        cmp     al,3bh          ;PTR?
        je      siptr
noptr:  mov     si,bp
siptr:  pop     bp
        pop     dx
        pop     bx
        pop     ax
        retn
delptr  endp
```

```
jmpsh?  proc    near
;-------------------------------------------------------------------
;Este procedimiento determina si la distancia entre AX y DI esta entre
;+127 y -128, en cuyo caso pone esta distancia en AX y CY = 0; de lo
;contrario, CY = 1.
;-------------------------------------------------------------------
        sub     ax,di
        cmp     ax,127
        jg      notshrt
        cmp     ax,-128
        jl      notshrt
        clc
        retn
notshrt:stc
        retn
jmpsh?  endp

enspaso2        proc    near
;-------------------------------------------------------------------
;Este procedimiento ejecuta el segundo paso de compilación, es decir, la
;traducción de los pseudocódigos a un código verdadero.
;
;Entrada:       DS:SI   apunta al pseudocódigo
;               ES:DI   apunta al lugar donde se desea el código
;
;Salida:        DS:SI   es alterado
;               ES:DI   es ajustado
;               CX      longitud del código
;               CY      set si hay error, en cuyo caso los registros
;                       quedan inalterados
;-------------------------------------------------------------------
        push    ax
        push    bx
        push    dx
        push    bp
        push    si
        push    di
        push    cx
        xor     cx,cx           ;longitud inicial = 0
startp2:call    tipo            ;determinar tipo de pseudocódigo inicial
        jc      er2
        or      dl,dl           ;es una instrucción?
        jnz     prsrg?          ;no, puede ser sreg
        test    ah,1            ;es un loner?
        jz      noeslon         ;no
        cmp     al,0d4h         ;es AAM?
        je      dosb            ;si
```

```
        cmp     al,0d5h         ;es AAD?
        jne     unb             ;no
dosb:   mov     ah,0ah          ;es AAM o AAD, instrucciones cuyo opcode es
        stosw                   ;de 2 bytes
        add     cx,2            ;incrementar longitud en 2
        jmp     bienpaso2
unb:    stosb
        inc     cx
        jmp     bienpaso2
prsrg?: cmp     dl,2            ;sreg?
        jne     er2             ;no, error
        shl     al,1
        shl     al,1
        shl     al,1
        or      al,26h          ;hallar el prefijo
        stosb
        inc     cx
        jmp     bienpaso2
er2:    jmp     errp2

noeslon:cmp     al,70h          ;es un salto condicional?
        jb      nojcond
        cmp     al,7fh
        ja      nojcond
saltorel:                       ;si
        stosb
        call    tipo
        jc      er2
        cmp     dl,4            ;lo siguiente es data?
        jb      er2             ;no, error
        dec     ax
        call    jmpsh?          ;calcular distancia relativa
        jc      er2             ;y determinar si el salto es posible
        stosb
        add     cx,2
        jmp     bienpaso2

nojcond:cmp     al,0e0h         ;LOOP o JCXZ?
        jb      nolpjcx
        cmp     al,0e3h
        jbe     saltorel
nolpjcx:cmp     al,8dh          ;LEA?
        jne     diflea
regmemdisp:                     ;si
        stosb
        call    modregrm        ;hallar el código de los argumentos
        jc      er2
```

```
        cmp     al,3            ;d y w deben ser 1
        jne     er2
        call    pref?
        mov     al,bl           ;byte de modo de direccionamiento
        stosb
        call    sizedisp
        inc     cx
bp2:    jmp     bienpaso2
diflea: cmp     al,0c4h         ;LDS o LES?
        jb      noldsles
        cmp     al,0c5h
        jbe     regmemdisp
noldsles:
        cmp     al,0f0h         ;prefijo REP o LOCK?
        jb      alterables
        cmp     al,0f3h
        ja      alterables
        stosb                   ;si, incluir el prefijo
        inc     cx
        lodsb
        cmp     al,0ffh
        je      bp2
        dec     si
        jmp     startp2         ;empezar otra vez con lo que sigue al prefijo

alterables:
        test    ah,2            ;oner?
        jnz     sioner
        jmp     nooner
sioner: test    ah,4            ;JMP o CALL?
        jz      nojc
        jmp     jmpocall
nojc:   cmp     al,0c2h         ;es oner.
        jb      nori            ;si no es RET o INT
        jmp     retoint
nori:   cmp     al,7            ;PUSH o POP?
        ja      nopushpop       ;no
        mov     dh,al
        lodsb
        dec     si
        mov     bl,al
        and     bl,0e0h
        cmp     bl,20h          ;reg?
        je      ppreg           ;si
        mov     al,dh
        call    modrm           ;no, mem?
        jc      ppnomem         ;no
```

```
        cmp     bh,80h          ;si, w=1?
        je      er2a            ;no, error
        cmp     al,7            ;POP?
        mov     al,8fh
        je      espop1          ;si, dejar su código
        mov     al,0ffh         ;no, código de PUSH
espop1: stosb
        inc     cx
        call    pref?
        cmp     al,8fh          ;POP?
        je      espop2
        or      bl,30h          ;no, compensar byte de direcc.
espop2: mov     al,bl
        stosb
        call    sizedisp
        jmp     bienpaso2
er2a:   jmp     errp2

ppnomem:call    tipo
        cmp     dl,0
        je      er2a
        cmp     dl,4
        jae     er2a
        shl     al,1            ;es sreg
        shl     al,1
        shl     al,1
        or      al,7
        cmp     dh,7
        je      espop4
        and     al,0feh
espop4: stosb
        inc     cx
        jmp     bienpaso2

ppreg:  test    al,8            ;reg de 16 bits?
        jz      er2a            ;no, error
        and     al,7            ;dejar solo los tres LSBits
        or      al,58h          ;poner header del byte
        cmp     dh,7
        je      espop3
        and     al,0f7h
espop3: stosb
        inc     cx
        jmp     bienpaso2

nopushpop:
        cmp     al,48h          ;INC o DEC?
```

```
        ja      noincdec
        mov     ah,al
        lodsb
        dec     si
        and     al,0e8h
        cmp     al,28h          ;registro de 16 bits?
        je      incdecreg
        mov     al,ah
        call    modrm
        jc      er2a
        test    bh,80h          ;debe especificarse size
        jz      er2a
        or      bh,0feh
        xchg    bh,al
        stosb                   ;guardar código
        call    pref?
        cmp     bh,40h          ;INC?
        je      esinc1
        or      bl,8
esinc1: mov     al,bl
        stosb
        call    sizedisp
        inc     cx
        jmp     bienpaso2

incdecreg:
        lodsb
        and     al,7            ;aislar registro
        or      al,ah
        stosb
        inc     cx
        jmp     bienpaso2

noincdec:
        call    modrm           ;es NOT o NEG o etc.
        jc      er2a
        test    bh,80h          ;debe especificarse el size
        jz      er2b
        or      bh,0f6h
        xchg    al,bh
        stosb                   ;guardar código
        inc     cx
        mov     al,bh
        and     al,7
        shl     al,1
        shl     al,1
        shl     al,1
```

```
        or      al,bl           ;obtener byte de dir. completo
        stosb
        call    sizedisp
        jmp     bienpaso2
er2b:   jmp     errp2
retoint:cmp     al,0cch         ;INT?
        jne     esret           ;no
        call    tipo            ;si
        cmp     dl,4            ;lo siguiente es un dato?
        jb      er2b
        cmp     ax,255
        ja      er2b
        inc     cx
        cmp     al,3            ;INT 3?
        je      esint3
        mov     ah,al           ;no, dos bytes
        mov     al,0cdh
        stosw
        inc     cx
        jmp     bienpaso2
esint3: mov     al,0cch         ;si, un bute
        stosb
        jmp     bienpaso2

esret:  inc     cx
        mov     bl,al           ;preservar código
        call    tipo
        jnc     conoff          ;si no hay un fin, con offset
        mov     al,bl
        inc     al              ;opcode adecuado
        stosb
bp2a:   jmp     bienpaso2
conoff: cmp     dl,4
        jb      er2b            ;error si lo siguiente no es un dato
        jne     retwo           ;si el dato es word
        mov     ah,al
        mov     al,bl
        stosw
        inc     cx
        cmp     al,0c2h         ;RETN?
        je      bp2a
        xor     al,al           ;no, un byte mas
        stosb
        inc     cx
        jmp     bienpaso2
retwo:  cmp     bl,0c2h         ;RETN?
        je      peque?
```

```
        jmp     short latalista
esfar:  mov     dh,3
latalista:
        call    deIptr
        call    tipo
        jc      ej
        cmp     dl,2            ;sreg?
        je      ej
        cmp     dl,1            ;reg?
        jne     cualarg
        test    al,8            ;de 16 bits?
        jz      ej              ;no, error

cualarg:cmp     dl,4            ;datos?
        jae     jdat
        jmp     jnodat
jdat:   push    dx              ;si
        push    ax
        call    tipo
        jc      jdat8o16
        cmp     dl,4            ;dos datos?
        jae     jdat32
        pop     ax
        pop     dx
        jmp     errj
jdat32: mov     bx,ax           ;dato de 32 bits
        pop     ax
        pop     dx
        cmp     dh,1            ;short?
        je      errj
        cmp     dh,2            ;near?
        je      errj
        xchg    al,ch
        stosb                   ;solo en este caso, el código es inalterado
        xchg    ch,al
        xchg    ax,bx
        stosw
        mov     ax,bx
        stosw
        add     cl,5
        jmp     short jbien
jdat8o16:
        pop     ax
        pop     dx
        cmp     dh,3            ;far?
        je      errj
        cmp     dh,2            ;near?
```

```
        je      jnear
        cmp     dh,1
        je      jsh?
        cmp     ch,0eah         ;jmp?
        jne     jnear
        sub     ax,2
        call    jmpsh?          ;calcular distancia
        jnc     jshort
        dec     ax
        jmp     short yaresta
jnear:  sub     ax,3
        sub     ax,di
yaresta:mov     bl,0e8h
        cmp     ch,9ah          ;call?
        je      escall1
        inc     bl
escall1:xchg    bl,al
        stosb
        add     cl,3
        xchg    al,bl
        stosw
        jmp     short jbien
jsh?:   cmp     ch,0eah         ;jmp?
        jne     errj            ;solo se admite SHORT en JMP
        sub     ax,2
        sub     ax,di
        or      ah,ah
        jz      jshort
        cmp     ah,0ffh
        jne     errj
jshort: mov     ah,al
        mov     al,0ebh
        stosw
        inc     cl
        inc     cl

jbien:  xor     ch,ch
        jmp     bienpaso2
errj:   xor     ch,ch
        jmp     errp2

jnodat: cmp     dl,1            ;reg?
        jne     noreg16
        or      dh,dh           ;no debe haber restricciones de BYTE PTR, etc
        jne     errj
        mov     bl,al
        mov     al,0ffh
```

```
        stosb               ;opcode = FF
        inc     cl
        mov     al,bl
        and     al,7        ;aislar número del registro
        or      al,0c0h     ;mod = 11
        cmp     ch,0eah     ;jmp?
        je      esjmpreg
        or      al,10h
        jmp     short fjd
esjmpreg:
        or      al,20h
fjd:    stosb
        inc     cl
        jmp     short jbien
noreg16:cmp     dh,1        ;es mem, short es inadmisible
        je      errj
        sub     si,4        ;restaurar pseudocódigo
        mov     al,dh
        push    bp
        push    cx
        call    modrm
        mov     bp,sp
        mov     ch,[bp+1]
        inc     sp
        inc     sp
        pop     bp
        jc      errj
        test    bh,80h      ;especificado size?
        jz      noesps      ;no, word por default
        test    bh,1        ;word?
        jz      errj
noesps: cmp     al,3        ;far?
        jne     nojfar
        or      bl,8        ;marcar FAR
nojfar: mov     al,0ffh     ;opcode
        stosb
        inc     cl
        call    pref?
        cmp     ch,0eah     ;jmp?
        je      esjmpmem
        or      bl,10h
        jmp     short fjm
esjmpmem:
        or      bl,20h
fjm:    mov     al,bl
        stosb
        call    sizedisp
```

```
        jmp     short jbien

nooner: cmp     al,40h
        jae     mayque40h
        mov     bh,al
        call    modregrm
        jc      coninmed
        or      al,bh
        stosb
        call    pref?
        mov     al,bl
        stosb
        call    sizedisp
        inc     cx
        jmp     bienpaso2
er2h:   jmp     errp2
coninmed:
        call    modrmdata
        jc      er2h
        test    al,4            ;operación con acumulador?
        jz      sinacc          ;no
        or      al,bb
        stosb
        call    sizedata
        inc     cx
        jmp     bienpaso2
sinacc: or      al,80h
        stosb
        call    pref?
        xchg    bl,al
        stosb
        call    sizedisp
        mov     al,bl
        shr     al,1
        xor     al,1
        call    sizedata
        inc     cx
        jmp     bienpaso2
mayque40h:
        cmp     al,84h          ;TEST?
        jne     noestest
        call    modregrm
        jc      tdat?
        and     al,1
        or      al,84h          ;generar código
        stosb
        inc     cx
```

```
        call    pref?
        mov     al,bl
        stosb
        call    sizedisp
        jmp     bienpaso2
tdat?:  xor     al,al
        call    modrmdata
        jc      er2f
        test    al,4
        jnz     taccd
        or      al,0f6h
        stosb
        inc     cx
        call    pref?
        mov     bh,al
        mov     al,bl
        stosb
        call    sizedisp
        mov     al,bh
        call    sizedata
        jmp     bienpaso2
taccd:  and     al,1
        or      al,0a8h
        stosb
        inc     cx
        call    sizedata
        jmp     bienpaso2

noestest:
        cmp     al,86h          ;XCHG?
        jne     noesxchg
        call    tipo
        jc      er2f
        or      dl,dl           ;mnemo?
        jz      er2f            ;si, error
        cmp     dl,2            ;sreg?
        je      er2f            ;si, error
        cmp     dl,4            ;datos?
        jae     er2f            ;si, error
        cmp     dl,1            ;reg?
        je      xreg
        sub     si,4            ;no
        jmp     short xmerg
xreg:   test    al,8            ;reg de 16 bits?
        jnz     xr16
        dec     si              ;no
        jmp     short xmerg
```

```
xr16:  mov    ah,al
       call   tipo
       jc     er2f
       or     dl,dl          ;el segundo operando tampoco
       jz     er2f           ;puede ser mnemo, sreg ni datos
       cmp    dl,2
       je     er2f
       cmp    dl,4
       je     er2f
       cmp    dl,1           ;reg?
       je     x2reg
       sub    si,5           ;no
       jmp    short xmerg
er2f:  jmp    errp2
x2reg: test   al,8           ;segundo reg. de 16 bits?
       jz     er2f           ;no, error por incompatibilidad
       cmp    ah,8           ;primer reg. acumulador?
       je     rgenal
       cmp    al,8           ;no, y el segundo?
       je     rgenah
       sub    si,2           ;no, ninguno es acumulador
       jmp    short xmerg
rgenah: mov   al,ah
rgenal: and   al,7
       or     al,90h
       stosb
       inc    cx
       jmp    bienpaso2
xmerg: call   modregrm
       jc     er2f
       or     al,86h
       stosb
       inc    cx
       call   pref?
       mov    al,bl
       stosb
       call   sizedisp
       jmp    bienpaso2
noesxchg:
       cmp    al,88h         ;MOV?
       je     siesmov
       jmp    noesmov

;-----Lo siguiente es el procesamiento de MOV, que es bastante complejo----
siesmov:push  si
       call   tipo
       jc     er2f
```

```
        or      dl,dl           ;mnemo?
        jnz     mnomne
        pop     si
        jmp     mesmem
mnomne: cmp     dl,1            ;reg?
        je      msireg
        jmp     mnoreg
msireg: test    al,7            ;si, acumulador?
        jnz     mnoac
        call    modrm           ;seg. oper. [disp]?
        cmp     bl,6
        jne     mnoac
        test    bh,80h          ;size especificado?
        jz      macany
        shl     bh,1
        shl     bh,1
        shl     bh,1
        mov     dl,al
        and     al,8
        xor     al,bh           ;sizes contradictorios?
        jnz     er2e            ;si, error
        mov     al,dl
macany: shr     al,1
        shr     al,1
        shr     al,1
        or      al,0a0h
        stosb
        inc     cx
        call    sizedisp
        pop     ax
        jmp     bienpaso2
mnoac:  pop     si
        call    modregrm
        jc      noc1
        jmp     mcaso1
noc1:   lodsb
        and     al,15           ;volver a obtener el reg
        mov     bl,al
        call    tipo
        jc      er2e
        cmp     dl,2            ;seg. oper. sreg?
        je      m2sreg
        cmp     dl,4            ;seg. oper. dato?
        jb      er2e
        cmp     dl,5            ;de word?
        jne     m2byte
        test    bl,8            ;si, compatible?
```

```
        jz      er2e
m2byte: mov     bp,ax       ;preservar dato
        mov     al,bl       ;caso 3
        or      al,0b0h
        stosb
        inc     cx
        mov     al,bl
        shr     al,1
        shr     al,1
        shr     al,1
        call    sizedata
        jmp     bienpaso2
m2sreg: test    bl,8        ;primer operador compatible?
        jz      er2e
        mov     bh,al
        mov     al,8ch
        stosb
        inc     cx
        mov     al,bh
        shl     al,1
        shl     al,1
        shl     al,1
        and     bl,7
        or      al,bl
        or      al,0c0h     ;generar byte de direccionamiento
        stosb
        inc     cx
        jmp     bienpaso2
er2e:   jmp     err2
mnoreg: cmp     dl,2        ;primer oper. sreg?
        jne     mnosreg
        pop     dx          ;restaurar stack
        mov     dh,al
        call    tipo
        jc      er2e
        cmp     dl,1        ;segundo oper. reg?
        je      m2r
        cmp     dl,3        ;seg. oper. mem?
        je      m2m
        or      dl,dl       ;seg. oper mnemo?
        jnz     er2e
        cmp     dl,3dh      ;WORD o DWORD?
        jb      er2e
        cmp     dl,3eh
        ja      er2e
        call    delptr
        call    tipo
```

```
        cmp     dl,3            ;despues mem?
        jne     er2e
m2m:    mov     bp,ax
        mov     al,8eh
        stosb
        inc     cx
        call    getpref
        call    pref?
        mov     al,dh
        shl     al,1
        shl     al,1
        shl     al,1
        or      al,bl
        stosb
        mov     dx,bp
        call    sizedisp
        jmp     bienpaso2
m2r:    test    al,8
        jz      er2e
        mov     dl,al
        mov     al,8eh
        stosh
        inc     cx
        mov     al,dh
        shl     al,1
        shl     al,1
        shl     al,1
        and     dl,7
        or      al,dl
        or      al,0c0h         ;generar byte de direccionamiento
        stosb
        inc     cx
        jmp     bienpaso2
mnosreg:pop     si
        cmp     dl,3            ;mem?
        jne     er2d            ;no, no mas opciones
mesmem: push    si
        call    modrm
        jc      errm
        mov     bp,dx
        call    tipo
        cmp     dl,2            ;seg. oper. sreg?
        je      m2sr
        cmp     dl,1            ;seg. oper. reg?
        jne     mtry12
        test    al,7            ;si, es acumulador con [disp]?
        jnz     mtry12
```

```
        inc     cx
        call    pref?
        mov     al,bl
        stosb
        call    sizedisp
        jmp     bienpaso2
mtry2:  xor     al,al
        call    modrmdata
        jc      er2d
        test    al,4
        jnz     er2d            ;solo se admite una de las formas
        or      al,0c6h
        stosb
        inc     cx
        mov     bh,al
        call    pref?
        mov     al,bl
        stosb
        call    sizedisp
        mov     al,bh
        call    sizedata
        jmp     bienpaso2
;----------Fin del procesamiento de MOV----------

noesmov:cmp     al,0d8h         ;ESC?
        jne     noesesc
        call    tipo
        cmp     dl,4            ;el primer operador debe ser un dato
        jb      er2d
        cmp     ax,3fh          ;de 6 bits
        ja      er2d
        mov     ah,al
        shr     al,1
        shr     al,1
        shr     al,1
        or      al,0d8h
        stosb
        inc     cx
        mov     al,ah
        call    modrm
        jc      er2c
        call    pref?
        and     al,7
        shl     al,1
        shl     al,1
        shl     al,1
        or      al,bl
```

```
errs:  xor     ch,ch
er2c:  jmp     errp2

esinout:cmp    al,0e6h        ;OUT?
        je     esout
        call   tipo           ;no, IN
        cmp    dl,1
        jne    er2c
        test   al,7
        jnz    errp2
        shr    al,1
        shr    al,1
        shr    al,1
        or     al,0e4h
        mov    bl,al
        call   tipo
        cmp    dl,1
        je     iconr
        cmp    dl,4
        jb     errp2
        cmp    ax,255
        ja     errp2
        mov    ah,al
        mov    al,bl
        stosw
        inc    cx
        inc    cx
        jmp    short bienpaso2
iconr:  cmp    al,10          ;DX?
        jne    errp2
        mov    al,bl
        or     al,8
        stosb
        inc    cx
        jmp    short bienpaso2
esout:  call   tipo
        cmp    dl,1
        je     oconr
        cmp    dl,4
        jb     errp2
        cmp    ax,255
        ja     errp2
        mov    ah,al
        call   tipo
        cmp    dl,1
        jne    errp2
        test   al,7
```

```
        jnz     errp2
        shr     al,1
        shr     al,1
        shr     al,1
        or      al,0e6h
        stosw
        inc     cx
        inc     cx
        jmp     short bienpaso2
oconr:  cmp     al,10           ;DX?
        jne     errp2
        call    tipo
        cmp     dl,1
        jne     errp2
        test    al,7
        jne     errp2
        shr     al,1
        shr     al,1
        shr     al,1
        or      al,0eeh
        stosb
        inc     cx
        jmp     short bienpaso2


errp2:  pop     cx
        pop     di
        pop     si
        stc
        jmp     short finp2
bienpaso2:
        add     sp,6
        clc
finp2:  pop     bp
        pop     dx
        pop     bx
        pop     ax
        retn
enspaso2        endp

        public  assem

assem   proc    near
;---------------------------------------------------------------
;Este es el ensamblador, que llama a los pasos 1 y 2 de compilación.
;
;Entrada:       DS:SI   apunta al string por compilarse
```

```
;           ES:DI   apunta al lugar donde se desea el código
;
;Salida:    DS:SI   es alterado
;           ES:DI   es alterado
;           CX      longitud del código
;           CY      set si hay error
;-----------------------------------------------------------------------
        push    es
        push    di
        push    ax
        cld
        mov     ax,ds
        mov     es,ax
        mov     di,offset transit
        call    enspaso1
        jc      error
        mov     si,offset transit
        pop     ax
        pop     di
        pop     es
        call    enspaso2
        retn
assem   endp
error:  pop     ax
        pop     di
        pop     es
        retn

        end
```

MODULO MEMORIA.ASM

```
        .model small
        .486

;_____________________________________________
;       AREA PARA CODIGO
;_____________________________________________

.code

      public  llenar
llenar   proc   near
;-------------------------------------------------------------
;Este procedimiento llena un bloque de memoria con un valor de byte
;determinado.
;Entrada:     ES:DI   apunta al inicio del bloque
;            CX      contiene el tamaño del bloque
;            AL      contiene el valor por ponerse en el bloque
;-------------------------------------------------------------
        cld
        push    cx
        push    di
        rep     stosb
        pop     di
        pop     cx
        retn
llenar   endp

      public    buscar
buscar  proc    near
;-------------------------------------------------------------------------
;Este procedimiento busca un string en un bloque de memoria.
;Entrada:  ES:DI   apunta al inicio del bloque
;            CX      máximo número de bytes que deben ser explorados
;            DS:SI   apunta al string por buscarse
;            BX      longitud del string
;
;Salida:     ES:DI   apunta al string que se encontro, o al final del bloque
;                    si no se lo encontro
;            CX      >0 si se encontro, 0 si no se encontro
;-------------------------------------------------------------------------
```

```
        cld
BuscaStr:
        push    cx
        push    si
        push    di
        mov     cx,bx
        repe    cmpsb
        pop     di
        pop     si
        pop     cx
        je      FinSrch
        inc     di
        dec     cx
        jnz     BuscaStr
FinSrch:retn
buscar  endp

        public  copiar
copiar  proc    near
;---------------------------------------------------------------
;Este procedimiento copia un bloque de un lugar a otro en la memoria.
;Entrada:  DS:SI  apunta al inicio del bloque por copiarse
;          ES:DI  apunta al inicio del destino
;          CX     numero de bytes por copiar
;---------------------------------------------------------------
        push    si
        push    di
        push    cx
        cmp     di,si
        je      fincopiar
        ja      invcopiar
        cld
        rep     movsb
        jmp     short fincopiar
invcopiar:add   si,cx
        add     di,cx
        dec     si
        dec     di
        std
        rep     movsb
        cld
fincopiar:pop   cx
        pop     di
        pop     si
        retn
copiar          endp
        end
```

PROCEDIMIENTO CHILD486.ASM

```
        dosseg
        .model large            ;small

;_____________________________________________
;       AREA PARA CODIGO
;_____________________________________________


        .code

;Este es un overlay que la permite a un programa bajar de nivel haciendo EXEC
;de este programa, tras haber instalado un handler de 63H que traslada el
;control al programa con la funcion 80h.

start:  mov     ax,8080h
        int     63h
          mov   ah,4ch          ;nunca llega aqui, esto es solo por si se
          int   21h             ;ejecuta este programa sin el handler de la
          end   start           ;interrupcion 21h instalado.
          end
```

MACROS INIMAC.ASM

```
;_____________________________________________________________
;          Macros usados en el archivo INICIAL.ASM
;_____________________________________________________________
;
;---------------------------------------------------------------
;Macro        Parametros      Funcion
;---------------------------------------------------------------
;actuar                      salta a una de las dir. en una tabla
;ayuda        dir,nlin,color  presenta ventana con instrucciones
;bufcla       pnclave,pbuf    encuentra la posicion de un buffer
;curs         x               pone un "cursor" en x,24
;explica      pnclave         muestra la explicacion(PNCLAVE)
;getkey                       recibe dato del teclado (BIOS)
;getnmac                      da el # del macro con nombre en NOMBUF
;getxpr                       obtiene ultimo nibble <>0 en BUFFER
;guarpantalla                 guarda una pantalla
;iluminaop                    resalta la opcion actual
;inbuff       dir             recibe string desde teclado (BIOS)
;indmac                       da una lista de los macros en el indice
;lmppnt                       limpia la pantalla
;menaba                       cursor abajo en ventana de subopc.
;menarr                       cursor arriba
;mender                       cursor a la derecha
;menizq                       cursor a la izquierda
;menulinea    pnclave         muestra menu(PNCLAVE)
;menuvent                     muestra menu en forma de ventana
;mostrar      pdir,plin,pcol  muestra pregunta y espera respuesta
;movbuf       pbuf1,pbuf2     PBUF1 <-- PBUF2
;nocurs       x               pone atributo normal en x,24
;nunivel      pbuf,pniv,pnum  pone PNUM en PBUF(PNIV)
;opcion?                      espera que se elija una opcion
;pchar        car             escribe CAR (DOS)
;pmenu        pnlin,pcolor    muestra menu string apuntado por AX
;posdir       pvarx,pvary,dir convierte pos. de pantalla en direccion
;restpantalla                 restaura ultima pantalla salvada
;rutina                       pasa control al usuario y lo recibe
;salvareg                     push a todo
;statusf                      muestra fase del programa
;statusv                      muestra datos importante del usuario
;traereg                      pop a todo
;valida?                      ve si una opcion es del menu del usuario
```

```
;subrut                 subrutinas de todos los macros
;-----------------------------------------------------------------

;****** ACTUAR ***************
;Busca el AL-esimo elemento de la tabla de direcciones
;que ira despues de MTABLA para saltar a esa direccion.
actuar  macro
        local   mtabla
        push    bx
        mov     bx,offset mtabla
        call    mact
        pop     bx
        jmp     dword ptr diract
mtabla:
        endm

;****** AYUDA  ***************
;Escribe NLIN lineas separadas por ceros desde DIR en COLOR.
;Ocupa columnas 1 a 78 y las filas del centro de la pantalla.
ayuda   macro   dir,nlin,color
        push    ax
        push    cx
        mov     cl,nlin
        mov     ch,color
        mov     ax,offset dir
        call    may
        pop     cx
        pop     ax
        endm

;****** BUFCLA ***************
;Busca PBUF en TABBUF y pone su ubicacion en PNCLAVE
bufcla  macro   pnclave,pbuf
        push    si
        push    ax
        mov     si,offset pbuf
        call    mbc
        mov     al,mbcbyte
        mov     pnclave,al
        pop     ax
        pop     si
        endm

;****** CURS  ***************
;pone el atributo de (x,24) en inverso.
curs    macro   x
        push    ax
```

```
        endm

;****** INBUFF ***************
;Recibe desde el teclado el buffer DIR cuya longitud se indica en
;el primer byte del buffer. En el segundo byte se obtendra el
;numero de bytes recibidos. Solo se aceptan letras o numeros.
;Para edicion, solo Backspace. Pita si se ha llegado al limite.
inbuff  macro   dir
        push    si
        mov     si,offset dir
        call    mib
        pop     si
        endm

;****** INDMAC ***************
;Llena la lista LISMAC en base al indice de macros
;en AREAMAC. Reemplaza los ceros por espacios (20H).
indmac  macro
        call    mim
        endm

;****** LMPPNT ***************
;Borra la pantalla
lmppnt  macro
        push    di
        push    cx
        push    ax
        mov     cx,2000
        mov     di,0
        mov     ax,0720h
        rep     stosw
        pop     ax
        pop     cx
        pop     di
        endm

;****** MENArr ***************
;Mueve cursor abajo en la ventana de subopciones.
menarr  macro
        call    mmab
        endm

;****** MENARR ***************
;Similar a MENABA.
menaba  macro
        call    mma
        endm
```

```
;****** MENDER ***************
;Mueve cursor de opcion actual a la derecha.
mender  macro
        call    mmd
        endm

;****** MENIZQ ***************
;Similar a MENDER.
menizq  macro
        call    mmi
        endm

;****** MENULINEA *************
;Muestra menu (PNCLAVE)
menulinea       macro   pnclave
        push    cx
        mov     cl,pnclave
        call    mml
        pop     cx
        endm

;****** MENUVENT **************
;Muestra menu(NCLAVE1) en forma de ventana.
menuvent        macro
        call    mmv
        endm

;****** MOSTRAR ***************
;Muestra pregunta apuntada por PDIR y espera respuesta. La pregunta
;debe tener esta forma: Mensaje,0,# de respuestas, lista de
;respuestas. En AL sale el numero de la respuesta elegida.
mostrar macro   pdir,plin,pcol
        push    ax
        mov     ah,plin
        mov     al,pcol
        mov     moslin,ah
        mov     moscol,al
        mov     ax,offset pdir
        call    mmos
        pop     ax
        mov     al,mnmos
        endm

;****** MOVBUF ***************
;Mueve PBUF1 a PBUF2.
movbuf  macro   pbuf2,pbuf1
        push    si
```

```
        push    di
        mov     si,offset pbuf1
        mov     di,offset pbuf2
        call    mmb
        pop     di
        pop     si
        endm

;****** NOCURS  ***************
;pone el atributo de (x,24) en normal.
nocurs  macro   x
        push    ax
        mov     al,x
        mov     ah,bnnorm
        call    mcu
        pop     ax
        endm

;****** NUNIVEL ***************
;Pone PNUM en el buffer PBUF en el nivel PNIV.
nunivel macro   pbuf,pniv,pnum
        push    cx
        push    dx
        mov     cl,pniv
        mov     ch,pnum
        mov     dx,offset pbuf
        call    mnn
        pop     dx
        pop     cx
        endm

;****** OPCION? ***************
;Espera que se digite una opcion. Saca en MOPACT el numero de
;opcion. Las opciones son: Flechas,ENTER,^O,^K,^L,^P y otros.
;Ademas es aqui donde se almacenan los datos al grabar macros
;y se leen del macro en vez del teclado al correr o probarlos.
opcion? macro
        call    mop
        endm

;****** PCHAR   ***************
;Usa funcion del DOS para escribir CAR en pantalla.
pchar   macro   car
        push    dx
        mov     dl,car
        call    mpch
        pop     dx
```

```
        endm

;****** PMENU  ***************
;Muestra menu apuntado por AX. Al salir MPMPRX
;da la direccion del primer byte luego del 0.
pmenu  macro  pnlin,pcolor
        push    cx
        push    dx
        mov     cl,pnlin
        mov     dh,pcolor
        call    mpm
        pop     dx
        pop     cx
        endm

;****** POSDIR  ***************
;Da la direccion PDIR correspondiente a (PVARX,PVARY)
;para acceso directo a la memoria de video.
posdir  macro  pvarx,pvary,pdir
        push    ax
        push    bx
        push    cx
        mov     cl,pvary
        mov     bl,pvarx
        call    mpd
        mov     bx,mpddir
        mov     pdir,bx
        pop     cx
        pop     bx
        pop     ax
        endm

;****** RESTPANTALLA **********
;Trae de PANBUF la ventana que fue salvada en GUARPANTALLA.
restpantalla   macro
        call    mrp
        call    funciones
        endm

;****** RUTINA  ***************
;Pone la direccion de la rutina(NCLAVE) de usuario en DIRSAL y la de
;regreso en MRAUX2 y MRAUX3. Pasa control al usuario, que debe regresar
;poniendo RETORNO al fin de su rutina para regresar a MRUT1.
rutina  macro
        mov     cl,nclave
        mov     al,ninfbuf
        mul     cl
```

```
        add   ax,offset tabinf      ;busco dir. de rutina(nclave)
        add   ax,4
        mov   si,ax
        mov   bx,[si]
        mov   al,[si+2]             ;y tipo de rutina.
        mov   tiprut,al             ;Preparo el salto.
        mov   mraux1,al             ;guardo tipo de rutina.
        mov   bp,offset dirsal
        mov   ds:[bp],bx
        mov   ds:[bp+2],cs
        mov   bx,offset mrut1       ;Preparo dir. de regreso.
        mov   mraux2,cs
        mov   mraux3,bx
        jmp   dword ptr dirsal      ;Control al usuario.
mrut1:  mov   al,tiprut             ;Regreso a FLUJO.
        cmp   al,mraux1             ;Ha cambiado TIPRUT?
        je    mrut2
        mov   contmcr,0ffh          ;si, cod. de error de macros,
        mov   x,1                   ;ademas X,Y previos pueden
        mov   y,0                   ;estar errados.
mrut2:
        endm

;****** SALVAREG **************
;Push a todo.
salvareg    macro
        push  eax
        push  ebx
        push  ecx
        push  edx
        push  esp
        push  ebp
        push  esi
        push  edi
        push  ds
        push  es
        push  ss
        endm

;****** STATUSF ***************
;Presenta 6 bytes apuntados por FASEPTR en (73,23)
;Se usa para indicar la fase actual del programa.
statusf macro
        call  msf
        endm

;****** STATUSV ***************
```

```
;Presenta 6 bytes apuntados por VALPTR en (73,24)
;Se usa para mostrar importantes datos del usuario.
statusv macro
        call    msv
        endm

;****** TRAEREG ***************
;Pop a todo.
traereg macro
        pop     ss
        pop     es
        pop     ds
        pop     edi
        pop     esi
        pop     ebp
        pop     esp
        pop     edx
        pop     ecx
        pop     ebx
        pop     eax
        endm

;****** VALIDA? ***************
;Si en OPCION? se digito una opcion distinta de las que maneja
;normalmente, ACTUAR manda el flujo aca para ver si es una opcion
;puesta por el usuario en el menu. Si es asi, AL=numero de opcion.
;Si el primer byte despues de la ultima opcion del menu es FF se
;supone que la ultima es la "opcion de escape" para todas las
;posibilidades que no consten en el menu.
valida? macro
        call    mv?
        mov     al,mv?aux
        endm

;****** SUBRUT *********
;Este es un macro que contiene todas las subrutinas llamadas por los
;otros macros de este archivo. Se hizo de esta manera para evitar el
;uso de demasiadas banderas, y que su ausencia no produzca "Phase
;errors" al momento del ensamblado.
subrut  macro

mact:
        salvareg
        dec     al              ;pongo el AL-esimo elemento de la
        mov     cl,2            ;tabla en DIRACT y CS en DIRACT+2
        mul     cl              ;para hacer el JMP al regresar.
        add     ax,bx
```

```
        mov     bp,offset diract
        push    si
        mov     si,ax
        mov     bx,cs:[si]
        pop     si
        mov     ds:[bp],bx
        mov     ds:[bp+2],cs
        traereg
        retn

may:
        push    dx
        push    bx
        mov     bx,ax
        mov     dl,23           ;centro el texto en pantalla:
        sub     dl,cl           ;linea inicial= (23-#lineas)/2 -1
        shr     dl,1
        dec     dl
        mov     maylin,dl
        dec     maylin
        mov     maycol,ch
        mov     ax,offset blanco
may1:   pmenu   maylin,maycol   ;primero una linea en blanco.
        inc     maylin
        mov     mpmprx,bx
may2:   mov     ax,mpmprx       ;Ahora el resto de lineas. MPMPRX
        pmenu   maylin,maycol   ;da la direccion del primer byte
        inc     maylin          ;despues del 0.
        dec     cl
        jnz     may2
        mov     ax,offset blanco
        pmenu   maylin,maycol   ;Linea en blanco al final.
        pop     bx
        pop     dx
        retn

mbc:
        salvareg
        mov     al,nmaxbuf      ;Busca un buffer en la tabla TABBUF
        cmp     al,0            ;y entrega el numero clave
        jne     mbc0            ;que servira para obtener toda la
        jmp     mbc4            ;informacion necesaria acerca de la
mbc0:   mov     mbcmenu,0       ;opcion representada por el buffer.
        mov     mbcbuf,si       ;Si el buffer no existe, clave=FF.
        mov     mbcpos,offset tabbuf
mbc1:   mov     mbcbyte,0
        mov     bx,mbcbuf
```

```
mbc3:  mov    ah,[bx]
       inc    bx
       mov    bp,mbcpos
       xor    ch,ch
       mov    cl,mbcbyte
       inc    mbcbyte
       mov    di,cx
       cmp    ah,ds:[bp+di]
       jnz    mbc2
       cmp    mbcbyte,8
       jnz    mbc3
       mov    al,mbcmenu
       jmp    mbc5
mbc2:  inc    mbcmenu
       mov    ah,numaxbuf
       cmp    ah,mbcmenu
       je     mbc4
       add    mbcpos,8
       jmp    mbc1
mbc4:  mov    al,0ffh
mbc5:  mov    mbcbyte,al
       traereg
       retn

mcu:
       salvareg            ;Pone el atributo dado por AH en
       posdir 0,24,mcudir  ;la posicion (AL,24)
       mov    si,mcudir
       mov    bh,ah
       mov    cl,2
       mul    cl
       add    si,ax
       inc    si
       mov    es:[si],bh
       traereg
       retn

mex:
       salvareg            ;Busca en TABINF la direccion de la
       mov    al,ninfbuf   ;explicacion y la pone en linea 23.
       mul    cl
       add    ax,offset tabinf
       mov    bx,ax
       mov    ax,[bx]
       pmenu 24,colexp
       traereg
       retn
```

```
mgk:
        salvareg            ;Uso funcion del BIOS que espera
repetir:
        mov     ah,0bh
        int     21h
        cmp     al,0
        je      verraton
siguiente:
        mov     ah,0        ;que se pulse una tecla.
        int     16h
        jmp     short finmgk
verraton:
        call    pantalla
        cmp     bx,0ffh
        je      repetir
finmgk:
        cmp     ax,3b00h
        jne     nof1
        mov     vent,1
nof1:
        mov     getword,ax
        traereg
        ret

mgn:
        salvareg            ;Busca si el nombre en NOMBUF esta
        push    es          ;en la lista de macros y da su numero
        push    ds          ;+80h en NUMMAC si asi ocurre. Si es
        pop     es          ;nuevo, da el # del primero libre.
        mov     dh,0ffh     ;Si ya no hay espacio, da FF.
mgn1:   inc     dh
        cmp     dh,8
        je      mgn2
        mov     si,offset nombuf
        add     si,2        ;bytes 0 y 1 son de uso de otro macro.
        mov     di,offset areamac
        mov     al,8
        mul     dh
        add     di,ax
        inc     di          ;byte 0 es la longitud del macro.
        mov     cx,7
        repe    cmpsb       ;comparo.
        jne     mgn1        ;si no es igual, comparo otro.
        add     dh,80h      ;macro viejo.
        mov     nummac,dh
        jmp     mgn4
mgn2:   mov     dh,0ffh     ;Busco primero libre.
```

```
mgn21: inc    dh
       cmp    dh,8
       je     mgn3
       mov    di,offset areamac
       mov    al,8
       mul    dh
       add    di,ax
       mov    al,0
       mov    cx,8
       repe   scasb
       jne    mgn21
       mov    nummac,dh       ;macro nuevo.
       jmp    mgn4
mgn3:  mov    nummac,0ffh     ;macro nuevo sin sitio.
mgn4:  pop    es
       traereg
       retn
mgx:
       salvareg               ;se divide NIVEL/2, se suma a direccion
       mov    cl,nivel        ;inicial de BUFFER, y se obtiene el msn
       shr    cl,1            ;distinto de 0 en ese byte.
       xor    ch,ch           ;Ese es el ultimo nibble<>0 en BUFFER
       mov    si,offset buffer
       add    si,cx
       mov    al,[si]
       mov    ah,al
       shr    ah,1
       shr    ah,1
       shr    ah,1
       shr    ah,1
       cmp    ah,0
       jz     mgx1
       mov    al,ah
mgx1:  mov    xprev,al
       traereg
       retn

mgp:
       salvareg               ;paso pantalla a PANBUF.
       xor    si,si
       mov    di,offset panbuf
       mov    bx,ds
       mov    es,bx
       mov    ds,videoseg
       mov    cx,3680
       rep    movsb
       traereg
```

```
        retn

mio:
        salvareg            ;Resalta la opcion actual, dada por X.
        mov   es,videoseg
        mov   dl,collact        ;collact=video inverso
        mov   al,x
mio13:  posdir 1,1,miodir   ;trabajo directo en pantalla.
        mov   si,miodir     ;busco la X-esima palabra en la
        mov   di,si         ;linea 23 y cambio su atributo.
        push  es
        pop   ds
        mov   cl,al
mio131: lodsw
        cmp   al,20h        ;salto espacios iniciales.
        je    mio131
        sub   si,2
        dec   cl
        jz    mio16
mio14:  lodsw
        cmp   al,20h        ;salto palabras.
        jne   mio14
mio15:  lodsw
        cmp   al,20h
        je    mio15
        dec   si
        dec   si
        dec   cl
        jnz   mio14
mio16:  mov   di,si         ;resalto palabra.
        lodsw
        mov   ah,dl
        cmp   al,20h
        je    mio17
        cmp   al,0
        je    mio17
        stosw
        jmp   mio16
mio17:  traereg
        retn

mib:
        salvareg            ;obtengo string desde teclado.
        mov   cl,[si]       ;maximo # de caracteres
        mov   ch,0          ;contador
        mov   dx,si         ;puntero auxiliar
        add   si,2          ;bytes 0 y 1 de uso del macro
```

```
        posdir  1,24,mibdir    ;direccion de primer caracter
        mov     bx,mibdir
mib1:   mov     mibx,ch
        inc     mibx
        curs    mibx
mib2:   getkey
        cmp     al,08h         ;Backspace?
        jne     mib21
        jmp     mib4
mib21:  cmp     al,0dh         ;Enter?
        jne     mib22
        jmp     mib5
mib22:  cmp     al,30h         ;<"0"?
        jb      mib2
        cmp     al,7bh         ;>"z"?
        jnb     mib2
        cmp     al,3ah         ;es un numero?
        jb      mib3
        cmp     al,41h         ;"9"<al<"A"?
        jb      mib2
        cmp     al,5bh         ;es una mayuscula?
        jb      mib3
        cmp     al,61h         ;"Z"<al<"a"?
        jb      mib2
        sub     al,20h         ;es minuscula, hacer mayuscula
mib3:   nocurs  mibx
        cmp     ch,cl          ;contador=maximo?
        jne     mib31
        pchar   07h            ;si, beep.
        jmp     mib1
mib31:  mov     [si],al        ;AL al buffer
        inc     si             ;apunto prox. direccion
        mov     es:[bx],al     ;AL a pantalla
        add     bx,2           ;apunto prox. direccion
        inc     ch             ;contador+1
        jmp     mib1
mib4:   nocurs  mibx
        cmp     ch,0           ;buffer vacio?
        jne     mib41
        pchar   07h            ;si, beep.
        jmp     mib1
mib41:  dec     si             ;borro ultimo caracter
        mov     byte ptr[si],0 ;en el buffer
        sub     bx,2           ;y en pantalla
        mov     byte ptr es:[bx],20h
        dec     ch             ;contador-1
        jmp     mib1
```

```
mib5:  nocurs  mibx
       mov     si,dx          ;fin de rutina: byte 1
       inc     si             ;indica # de bytes recibidos.
       mov     [si],ch
       traereg
       retn

mim:
       salvareg               ;pone la lista de macros actual
       push    es             ;en LISMAC. Reemplaza los 00 con
       push    ds             ;2E (puntos).
       pop     es
       mov     si,offset areamac
       mov     di,offset lismac
       mov     dl,8
mim1:  inc     si
       mov     al,20h
       stosb
       mov     cl,7
mim2:  lodsb
       cmp     al,0
       jne     mim3
       mov     al,2eh
mim3:  stosb
       dec     cl
       jnz     mim2
       dec     dl
       jnz     mim1
       pop     es
       traereg
       retn

mmab:
       salvareg
       mov     cl,nclave1     ;Busca el # de subopciones que tiene
       mov     al,ninfbuf     ;el menu de la opcion actual dada por
       mul     cl             ;NCLAVE1. Si Y supera ese #, Y=0.
       add     ax,offset tabinf
       inc     ax
       inc     ax
       push    bx
       mov     bx,ax
       mov     si,[bx]
       pop     bx
mmab1: lodsb
       cmp     al,0
       jne     mmab1
```

```
        lodsb
        inc     al
        inc     y
        cmp     al,y
        jne     mmab2
        mov     y,0
mmab2:  traereg
        retn
mma:
        salvareg
        mov     cl,nclave1      ;Busca # de subopc. de menu(NCLAVE1)
        mov     al,ninfbuf      ;Si Y=FF, Y igual a ese #.
        mul     cl
        add     ax,offset tabinf
        inc     ax
        inc     ax
        push    bx
        mov     bx,ax
        mov     si,[bx]
        pop     bx
mma1:   lodsb
        cmp     al,0
        jne     mma1
        lodsb
        dec     y
        cmp     y,0ffh
        jne     mma2
        mov     y,al
mma2:   traereg
        retn
mmd:
        salvareg
        mov     cl,nclave       ;Busca # de opciones de menu actual.
        mov     al,ninfbuf      ;Si X supera ese valor, X=1.
        mul     cl              ;Siempre Y=0.
        add     ax,offset tabinf
        inc     ax
        inc     ax
        push    bx
        mov     bx,ax
        mov     si,[bx]
        pop     bx
mmd1:   lodsb
        cmp     al,0
        jne     mmd1
        lodsb
        inc     al
```

```
        inc     x               ;a la derecha.
        cmp     al,x
        jne     mmd2
        mov     x,1
mmd2:   mov     y,0
        traereg
        retn
mmi:
        salvareg
        mov     cl,nclave       ;Busca # de opciones de menu actual.
        mov     al,ninfbuf      ;Si X=0, X igual a ese #. Siempre Y=0.
        mul     cl
        add     ax,offset tabinf
        inc     ax
        inc     ax
        push    bx
        mov     bx,ax
        mov     si,[bx]
        pop     bx
mmi1:   lodsb
        cmp     al,0
        jne     mmi1
        lodsb
        dec     x
        cmp     x,0
        jne     mmi2
        mov     x,al
mmi2:   mov     y,0
        traereg
        retn
mml:
        salvareg
        mov     al,ninfbuf      ;Muestra el menu(CL) y resalta la
        mul     cl              ;opcion actual.
        add     ax,offset tabinf
        inc     ax
        inc     ax
        mov     bx,ax
        mov     ax,[bx]
      push      si
      push      ax
      mov       si,ax
otraletra:
      lodsb
      cmp       al,0
      jne       otraletra
      lodsb
```

```
        mov     numeroop,al
        pop     ax
        pop     si
        pmenu   1,collin
        iluminaop
          traereg
          retn

mmv:
        salvareg                ;Muestra ventana se subopc.
        mov     cl,nclave1        ;primero veo si la opcion
        mov     al,ninfbuf        ;tiene algun submenu.
        mul     cl              ;Si es asi, la rutina
        add     ax,offset tabinf    ;debe ser de tipo 3.
        add     ax,6
        mov     bx,ax
        cmp     byte ptr[bx],3
        jz      mmv3
        jmp     mmv7
mmv3:   mov     cl,nclave         ;Ahora busco la direccion
        mov     al,ninfbuf        ;del menu al que pertenece
        mul     cl              ;esta opcion en TABINF.
        add     ax,offset tabinf
        inc     ax
        inc     ax
        push    bx
        mov     bx,ax
        mov     si,[bx]           ;SI apunta al inicio del menu.
        pop     bx              ;Voy a contar los bytes hasta
        mov     cl,0            ;la opcion actual (x) para saber
        mov     dl,x            ;columna inicial de ventana.
mmv30:  lodsb                   ;Salto espacios iniciales
        inc     cl              ;pero los cuento.
        cmp     al,20h
        je      mmv30
        cmp     x,1             ;si x=1 ya tengo la columna
        jnz     mmv31
        dec     cl
        mov     mmvx1,cl
        jmp     mmv4
mmv31:  dec     dl              ;DL: cuantas opc mas debo saltar.
mmv32:  lodsb                   ;Cuento bytes hasta encontrar
        inc     cl              ;un espacio (CL contador).
        cmp     al,20h
        jne     mmv32
mmv33:  lodsb                   ;sumo espacios entre opciones
        inc     cl
```

```
        cmp   al,20h
        je    mmv33
        dec   dl              ;una opc menos que saltar
        jnz   mmv32
        dec   cl              ;Termino. Ventana empieza en
        mov   mmvx1,cl            ;columna anterior.
mmv4:   mov   mmvcar,0
        mov   cl,nclave1          ;Ahora busco direccion de menu
        mov   al,ninfbuf          ;asociado a la opcion actual
        mul   cl              ;en TABINF
        add   ax,offset tabinf
        inc   ax
        inc   ax
        push  bx
        mov   bx,ax               ;SI apunta a primer byte
        mov   si,[bx]             ;de ese menu
        pop   bx
        push  si
mmv40:  lodsb                     ;no tomo en cuenta los
        cmp   al,20h              ;espacios iniciales.
        je    mmv40
        dec   si
        mov   cx,1                ;Veo cuantas filas y columnas
mmv41:  lodsb                     ;ocupa la ventana
        cmp   al,20h              ;Los espacios no cuentan
        jz    mmv42
        cmp   al,0                ;termina si AL=0
        jne   mmv410
        mov   mmvlin,cl           ;# de subopciones del menu
        jmp   mmv5
mmv410: inc   ch                  ;CH: # de car de subopcion
        cmp   ch,mmvcar           ;MMVCAR: maximo CH logrado
        jb    mmv41
        mov   mmvcar,ch
        jmp   mmv41
mmv42:  lodsb                     ;saltar espacios
        cmp   al,20h
        je    mmv42
        cmp   al,0                ;termina si AL=0
        jne   mmv43
        mov   mmvlin,cl           ;# de subopciones del menu
        jmp   mmv5
mmv43:  inc   cl                  ;otra subopcion
        mov   ch,0                ;# de car = 0
        dec   si
        jmp   mmv41
mmv5:   pop   si
```

```
        mov   al,mmvlin          ;Busco la linea inicial
        inc   al                 ;de la ventana incluidas
        inc   al                 ;dos lineas de recuadro.
      mov     ah,1               ;
      add     ah,al              ;
        mov   mmvlin,ah
        mov   mmvlin1,ah         ;servira despues
        inc   mmvcar
        posdir mmvx1,mmvlin,mmvdir   ;dir. de esquina superior
        mov   di,mmvdir          ;izquierda en pantalla
        mov   es,videoseg
      mov     al,esqii           ;dibujar esa esquina
        mov   ah,colven          ;en color de ventana
        stosw
        mov   al,linsup          ;dibuja linea superior
        mov   cl,mmvcar
        xor   ch,ch
        rep   stosw
      mov     al,esqid           ;dibuja esq. sup. der.
        stosw
mmv50: DEC    mmvlin             ;proxima linea
        mov   bl,mmvlin
      cmp     bl,2               ;linea inferior si BL=22
        je    mmv6
        posdir mmvx1,mmvlin,mmvdir   ;escribir las subopciones
        mov   di,mmvdir
        mov   al,linlat          ;pero antes el recuadro.
        mov   bl,mmvcar          ;long de la linea
        stosw
mmv501: lodsb                    ;no tomo en cuenta los
        cmp   al,20h             ;espacios iniciales
        je    mmv501
        dec   si
mmv51:  lodsb                    ;leo byte del menu
        stosw                    ;lo escribo en pantalla
        dec   bl
        cmp   al,20h             ;He terminado la opcion?
        je    mmv52
        cmp   al,0               ;He terminado el menu?
        jne   mmv51
        dec   di                 ;en ese caso borro el 00H
        dec   di                 ;que escribi
        mov   al,20h             ; y pongo un " "
        stosw
mmv52:  cmp   bl,0               ;lleno el resto de la linea
        je    mmv53              ;con espacios
        mov   al,20h
```

```
mmv521: stosw
        dec     bl
        jnz     mmv521
mmv53:  mov     al,linlat           ;y termino con recuadro.
        stosw
mmv54:  lodsb                       ;salto espacios
        cmp     al,20h
        jz      mmv54
        dec     si
        jmp     mmv50
mmv6:   posdir  mmvx1,mmvlin,mmvdir ;dibuja linea inferior
        mov     di,mmvdir
      mov       al,esqsi
        mov     ah,colven
        stosw
        mov     al,linsup
        mov     cl,mmvcar
        xor     ch,ch
        rep     stosw
      mov       al,esqsd
        stosw                       ;fin de dibujo de ventana.
        cmp     y,0                 ;existe subopcion seleccionada?
        jz      mmv7                ;no, termina trabajo.
      mov       al,y                ;si, resaltarla cambiando
mmv63:  mov     bl,mmvlin1          ;el atributo de la linea.
      sub       bl,al
      mov       mmvlin,bL
      posdir    mmvx1,mmvlin,mmvdir
        mov     di,mmvdir
        mov     ax,3
        add     di,ax
        mov     bl,mmvcar
        mov     al,colact
mmv64:  stosb
        inc     di
        dec     bl
        jnz     mmv64
mmv7:   traereg
        retn
mmos:
        salvareg
        pmenu   moslin,moscol   ;Presenta pregunta y espera respuesta.
        mov     si,ax
mmos0:  lodsb
        cmp     al,0
        jne     mmos0
        mov     mdirop,si
```

```
mmos1:  lodsb               ;Obtengo el # de opciones.
        mov     cl,al
        mov     dl,al
        getkey              ;obtengo respuesta.
        cmp     al,'a'      ;minusculas --> mayusculas.
        jc      mmos11
        cmp     al,'z'+1
        jnc     mmos11
        sub     al,20h
mmos11: mov     bx,ax
mmos2:  mov     al,[si]     ;comparo y si no es ninguna de las
        inc     si          ;posibles, vuelve a pedir respuesta.
        sub     cl,1
        jc      mmos4
        cmp     al,0
        je      mmos3
mmos3:  cmp     al,bl
        jne     mmos2
        jmp     mmos5
mmos4:  mov     si,mdirop
        jmp     mmos1
mmos5:  sub     dl,cl
        mov     mnmos,dl    ;# de respuesta en MNMOS.
        traereg
        retn
mmb:
        salvareg            ;mueve 8 bytes.
        push    ds
        pop     es
        mov     cx,8
        rep     movsb
        traereg
        retn

mnn:
        salvareg
        test    cl,1        ;pongo CH en el CL-esimo nibble
        jz      mnn1        ;del buffer apuntado por DX.
        mov     al,16
        mul     ch
        cmp     al,0
        jnz     mnn4
        mov     ch,0fh
        mov     al,0f1h
        jmp     mnn1
mnn4:   mov     ch,al
mnn1:   shr     cl,1
```

```
        mov   bl,cl
        xor   bh,bh
        add   bx,dx
        cmp   ch,0
        jz    mnn5
        cmp   al,0ffh
        jnz   mnn2
mnn5:   and   [bx],ch
        jmp   mnn3
mnn2:   or    [bx],ch
mnn3:   traereg
        retn

mop:
        salvareg                ;maneja opciones.
        mov   cl,nclave1        ;Primero veo de que tipo es la
        mov   al,ninfbuf        ;opcion actual. (Esto es usado
        mul   cl                ;al salir de este macro).
        add   ax,offset tabinf
        add   ax,6
        mov   si,ax
        mov   al,[si]
        mov   tiprut1,al
        cmp   bmac,1            ;Corriendo/probando macro?
        je    mop1
        jmp   mop2
mop1:   mov   al,longmac        ;si.
        inc   al
        cmp   al,contmac        ;termino?
        jne   mop10
        mov   bmac,0            ;si,
        mov   faseptr,offset fnorm  ;fase normal
        statusf
        jmp   mop2
mop10:  mov   si,offset buffmac ;estoy en macro,
        push  ax                ;leo dato de BUFFMAC
        mov   al,contmac
        xor   ah,ah
        add   ax,ax
        add   si,ax
        pop   ax
        mov   bx,[si]
        inc   contmac
        cmp   contmcr,0ffh      ;fin de ejecucion por
        je    mop12             ;flujo ambiguo?
        cmp   bver,0            ;probando macro?
        je    mop13
```

```
mop11:  getkey                  ;si, espero ya sea
        cmp     al,cr           ;ENTER para seguir
        je      mop13
        cmp     al,ctrlk        ;o ^K para terminar.
        jne     mop11
mop12:  mov     bmac,0          ;termino prueba,
        mov     bver,0
        mov     bx,0ff00h       ;no ejecuto ultima opcion
        mov     faseptr,offset fnorm    ;fase normal.
        statusf
mop13:  mov     ax,bx           ;tomo el valor del buffer
        jmp     mop3            ;en vez del teclado.
mop2:   getkey                  ;no es macro,dato de teclado.
        cmp     bgmac,0         ;grabando macro?
        jne     mop21
        jmp     mop3
mop21:  mov     bx,0ff00h
        cmp     contmcr,0ffh    ;si, fin macro por razones de
        je      mop22           ;rutina de macros de usuario?
        cmp     al,ctrlk        ;si,termino de grabar?
        je      mop22
        mov     bl,contmac      ;grabo dato en BUFFMAC
        xor     bh,bh
        add     bx,bx
        add     bx,offset buffmac
        mov     ds:[bx],ax
        inc     contmac         ;y cuento # de datos. No eje-
        cmp     contmac,maxmac  ;cuto la primera en exceso.
        je      mop23
        jmp     mop3
mop23:  mov     bx,ax
        dec     contmac         ;excedi el limite.
        mov     ax,offset blanco
        pmenu   24,colexp
        mov     ax,offset escmcr
        pmenu   23,collact
mop231: getkey
        mov     ax,getword
        cmp     al,1bh          ;ESC?
        jne     mop231
mop22:  mov     al,nummac       ;paso datos al area de macros.
        mov     cl,8
        mul     cl
        add     ax,offset areamac
        push    es
        push    ds
        pop     es
```

```
        mov     di,ax
        mov     al,contmac          ;long. del macro,
        mov     [di],al
        inc     di
        mov     si,offset nombmac   ;y nombre
        mov     cx,7
        rep     movsb
        mov     al,nummac           ;ahora paso BUFFMAC
        mov     cl,2*maxmac         ;al area que empieza
        mul     cl                  ;con MAC1. Cada macro
        add     ax,offset mac1      ;ocupa 2*MAXMAC bytes
        mov     di,ax               ;o MAXMAC words
        mov     si,offset buffmac
        mov     cx,maxmac
        rep     movsw
        pop     es
        mov     bgmac,0             ;termina grabacion
        mov     faseptr,offset fnorm    ;fase normal.
        statusf
        mov     ax,bx               ;ejecuto la ultima opcion
mop3:   cmp     al,0                ;es un car.especial?
        je      mop32
mop31:  cmp     al,cr               ;no, es Enter?
        jne     mop311
        mov     al,5
        jmp     mop4
mop311: cmp     al,ctrlo            ;es ^O?
        jne     mop312
        mov     al,7
        jmp     mop4
mop312: cmp     al,ctrlp            ;es ^P?
        jne     mop313
        mov     al,8
        jmp     mop4
mop313: cmp     al,ctrlk            ;es ^K?
        jne     mop314
        mov     al,9
        jmp     mop4
mop314: cmp     al,1bh              ;es ESC?
        jne     mop315
        mov     al,10
        jmp     mop4
mop315: mov     mopaux,al           ;es otro?
        mov     al,6
        jmp     mop4
mop32:  cmp     ah,fleizq           ;es <--?
        jne     mop321
```

```
        mov   al,1
        jmp   mop4
mop321: cmp   ah,fleder          ;es -->?
        jne   mop322
        mov   al,2
        jmp   mop4
mop322: cmp   ah,flearr          ;es flecha arriba?
        jne   mop323
        mov   al,3
        jmp   mop4
mop323: cmp   ah,fleaba          ;es flecha abajo?
        jne   mop324
        mov   al,4
        jmp   mop4
mop324: mov   mopaux1,ah         ;es otro.
        mov   mopaux,0
        mov   al,6
mop4:   mov   mopact,al
        cmp   bopc,1             ;tengo ventana de opciones?
        jne   mop5
        restpantalla             ;si,recupero pantalla.
mop5:   traereg
        retn

mpch:
        salvareg
        mov   ah,2       ;Uso rutina del DOS para escribir DL.
        int   21h
        traereg
        retn

mpm:
        salvareg
        push  dx         ;pone un string terminado en 0
        xor   ch,ch      ;en linea CL con atributo DH.
        mov   si,ax      ;EL string es apuntado por AX.
        mov   al,longlin
        mul   cl
        mov   es,videoseg
        mov   di,ax
        pop   dx
        mov   ah,dh
        push  di
        mov   al,20h
        mov   cx,80      ;borro la linea.
        rep   stosw
        pop   di
```

```
        inc     di
        inc     di
mpm0:   lodsb                   ;pongo el string.
        cmp     al,0
        je      mpm1
        stosw
        jmp     mpm0
mpm1:   mov     mpmprx,si       ;MPMPRX apunta a proximo string.
        traereg
        retn
mpd:
        salvareg
        mov     al,160          ;Direccion = CL*160 + BL.
        mul     cl
        xor     bh,bh
        add     ax,bx
        add     ax,bx
        mov     mpddir,ax
        traereg
        retn
mrp:
        salvareg                ;Paso PANBUF a pantalla.
        xor     di,di
        mov     es,videoseg
        mov     si,offset panbuf
        mov     cx,3680
        rep     movsb
        traereg
        retn

msf:
        salvareg
        mov     si,faseptr      ;FASEPTR tiene la direccion de un
        posdir  73,24,msfdir    ;string de 6 bytes que debe ser
        mov     di,msfdir       ;mostrado en la posicion (73,24).
        mov     cx,6
        mov     ah,collact
msf1:   lodsb
        stosw
        dec     cx
        jnz     msf1
        traereg
        retn

msv:
        salvareg
      mov     si,valptr         ;VALPTR tiene la direccion de un string
```

```
        posdir  67,24,msfdir   ;de 6 bytes en el que el usuario puede
        mov     di,msfdir      ;poner algo para aparecer en (67,24).
        mov     cx,6
        mov     ah,collact
msv1:   lodsb
        stosw
        dec     cx
        jnz     msv1
        traereg
        retn

mv?:
        salvareg                ;Busca si la eleccion esta
        mov     cl,nclave         ;en el menu del usuario.
        mov     al,ninfbuf
        mul     cl
        add     ax,offset tabinf
        inc     ax
        inc     ax
        push    bx
        mov     bx,ax
        mov     si,[bx]           ;SI apunta al menu
        pop     bx
mv?1:   lodsb
        cmp     al,0
        jne     mv?1
        lodsb                   ;SI apunta a primera opcion
        mov     cl,al             ;AL tiene el # de opciones
        mov     dl,al
        push    si              ;si el primer byte despues
        push    ax              ;de las opciones validas
        mov     mv?aux1,0
        xor     ah,ah             ;es FF, entonces evacuar
        add     si,ax             ;las teclas no validas
        cmp     byte ptr[si],0ffh     ;por la ultima opcion.
        jne     mv?11
        mov     mv?aux1,al            ;dejo MV?AUX1 como bandera
mv?11:  pop     ax
        pop     si
        mov     al,mopaux             ;A las minusculas
        cmp     al,'a'
        jb      mv?2
        cmp     al,'z'
        ja      mv?2
        sub     al,20h            ;las hago mayusculas.
mv?2:   mov     bl,al
mv?21:  lodsb
```

```
        cmp     al,bl           ;veo si opcion es valida
        je      mv?3
        dec     cl
        jnz     mv?21
        cmp     mv?aux1,0           ;no fue valida, veo si hay
        je      mv?22               ;rutina de escape
        mov     al,mv?aux1          ;si hay esa rutina
        mov     mv?aux,al           ;salgo con respuesta
        xor     ah,ah               ;y ZF=1
        jmp     mv?4
mv?22:  or      al,1            ;no hay esa rutina,ZF=0
        jmp     mv?4
mv?3:   dec     cl              ;opcion valida,
        sub     dl,cl           ;guardo respuesta
        mov     mv?aux,dl
        xor     ah,ah               ;y salgo con ZF=1
mv?4:   traereg
        retn

        endm
```

MACRO MPRO.ASM

```
;-------------------------------------------------------------------
;Macro  Parametros          Funcion
;-------------------------------------------------------------------
;actuar                    salta a la AL-esima dir. de una tabla
;finmacro                  termina macro y da un mensaje
;getchar                   lee un ASCII
;getdata x,y,buff,tipo     lee un bin, hex o dec
;getdata2 x,y,buff,tipo    lee un dato en hex (para mamoria relativa)
;getline buff              lee un string del teclado
;limpln num,atrib          borra una linea
;lmppnt                    borra pantalla
;print  x,y,dir,atrib      pone un texto en pantalla
;rest23                    restaura linea 23
;salva23                   salva linea 23
;usrpro                    intercambia pantallas de usuario y PRO
;sbrpro                    subrutinas de los otros macro

;****************************************
;Macro-instrucciones para macro-comandos
;****************************************
;-------------------------------------------------------------------
;Macro  Parametros          Funcion
;-------------------------------------------------------------------
;cerrmcr                   cierra archivo de macros
;creamcr dirnom            crea archivo de macros
;grabmcr                   graba y cierra archivo de macros
;getkey                    lee teclado o buffer de macros
;inimcr dirnom             abre y lee (o crea) archivo de macros
;lmpmcr                    pone 0 en las areas de macros
;rtodo                     pop todo
;stodo                     push todo
;sbrmcr                    subrutinas de los otros macros
;-------------------------------------------------------------------

;****** ACTUAR ***************
;Lee el AL-esimo elemento de la tabla TABACT y salta a la direccion
;dada por ese elemento. AL=0 para el primer elemento de la tabla.
actuar  macro
        local   tabact
        push    si
        mov     ah,0
```

```
        add     ax,ax
        mov     si,offset tabact
        add     si,ax
        mov     ax,cs:[si]
        pop     si
        jmp     ax
tabact:
        endm

;****** FINMACRO **************
;Da un mensaje de fin de macro y espera un ESC para seguir
finmacro macro
        call    rfinm
        endm

;****** GETCHAR ***************
;Lee en GCASCII un ASCII del teclado o buffer de macros. En GCCOD
;sale el siguiente codigo: 0, hay dato valido en GCASCII. 1, ENTER. 2,
;TAB. 3, SHIFT TAB. 4, ESC. 5, alguna funcion o tecla especial, cuyo
;ASCII extendido sale en GCASCII. Para que ENTER, TAB, SHIFT TAB y
;ESC sean aceptados como datos normales, se debe presionar antes F10.
getchar macro
        call    rgch
        endm

;****** GETDATA ***************
;Lee datos del teclado o buffer de macros en BUFF. En BUFF se tiene:
;# de bytes pedidos, # de datos leidos, datos, # convertido en hex
;(4 bytes). Hace eco en pantalla en (X,Y). De acuerdo al TIPO
;recibe datos en bin ('b'), hex ('h'), dec ('d'), La longitud de
;BUFF debe ser adecuada: bin, cualquiera; hex, 2 0 4 bytes; dec, 3
;o 5 bytes. Si se presiona DEL o BS se borra el ultimo dato ingresado.
;ENTER es esperado para confirmar un dato y convertirlo en hex. En
;GCCOD sale un codigo si ocurre lo siguiente: 0, hay un dato correcto al
;final de BUFF. 1, flecha arriba. 2, flecha abajo. 3, flecha derecha.
;4, flecha izquierda. 5, TAB. 6, SHIFT TAB. 7, ESC. 8, PGUP. 9 PGDN.
;Cualquier otro dato es procesado para ver si es valido incluirlo en el
;buffer. Se produce un BEEP si un dato es invalido o al llegar a los
;limites. Si al ingreso a la subrutina GDRESET es 1, buffer y variables
;son inicializadas. Si es 0, no lo son, y se puede seguir recibiendo
;el mismo dato. Los numeros binarios no son convertidos en hex.
getdata macro   x,y,buff,tipo
        push    bx
        push    cx
        push    dx
        mov     bx,offset buff
        mov     cl,x
```

```
        mov    ch,y
        mov    dl,tipo
        call   rgd
        pop    dx
        pop    cx
        pop    bx
        endm
;****** GETDATA2 ***************
;Lee datos del teclado. En BUFF se tiene:
;# de bytes pedidos, # de datos leidos, datos, # convertido en hex
;(4 bytes). Hace eco en pantalla en (X,Y). De acuerdo al TIPO
;recibe datos en bin ('b'), hex ('h'), dec ('d'), La longitud de
;BUFF debe ser adecuada: bin, cualquiera; hex, 2 0 4 bytes; dec, 3
;o 5 bytes. Si se presiona DEL o BS se borra el ultimo dato ingresado.
;ENTER es esperado para confirmar un dato y convertirlo en hex. En
;GCCOD sale un codigo si ocurre lo siguiente: 0, hay un dato correcto al
;final de BUFF. 1, flecha arriba. 2, flecha abajo. 3, flecha derecha.
;4, flecha izquierda. 5, TAB. 6, SHIFT TAB. 7, ESC. 8, PGUP. 9 PGDN.
;Cualquier otro dato es procesado para ver si es valido incluirlo en el
;buffer. Se produce un BEEP si un dato es invalido o al llegar a los
;limites. Si al ingreso a la subrutina GDRESET es 1, buffer y variables
;son inicializadas. Si es 0, no lo son, y se puede seguir recibiendo
;el mismo dato. Los numeros binarios no son convertidos en hex.
getdata2 macro  x,y,buff,tipo
        push   bx
        push   cx
        push   dx
        mov    bx,offset buff
        mov    cl,x
        mov    ch,y
        mov    dl,tipo
        jmp    rgdJ          ;sucede algo extranio si se lo hace con call
virus:  pop    dx            ;en el retorno no regresa aca (se cuelga)
        pop    cx
        pop    bx
        endm

;****** GETLINE ***************
;Lee del teclado un buffer BUFF (ASCII), que tiene este formato: Numero
;de bytes pedidos, numero de bytes realmente ingresados, texto. Para
;edicion se acepta Backspace o Delete. ENTER para terminar el texto.
;En GCCOD sale un codigo: 1, se digito ENTER (buffer valido). 2, TAB.
;3, SHIFT TAB. 4, ESC y 5, especial. Este macro usa GETCHAR para
;recibir datos. En  X, Y recibe la posicion en que debe hacer eco.
getline macro  x,y,buff
        push   si
        push   cx
```

```
        mov     cl,x
        mov     ch,y
        mov     si,offset buff
        call    rgl
        pop     cx
        pop     si
        endm

;****** LIMPLN ***************
;Borra la linea NUM y pone en ella el atributo ATRIB
limpln  macro   num,atrib
        push    ax
        mov     al,num
        mov     ah,atrib
        call    rll
        pop     ax
        endm

;****** LMPPNT ***************
;Borra la pantalla
lmppnt  macro
        push    cx
        push    di
        push    ax
        mov     cx,2000
        mov     di,0
        mov     ax,0720h
        rep     stosw
        pop     ax
        pop     di
        pop     cx
        endm

;****** PRINT  ***************
;Escribe el texto apuntado por DIR en la pantalla a partir de
;(X,Y) con atributo ATRIB. El texto debe terminar con un 0.
print   macro   x,y,dir,atrib
        posdir  x,y
        push    si
        push    ax
        mov     si,offset dir
        mov     ah,atrib
        call    rprt
        pop     ax
        pop     si
        endm
```

```
;****** SALVA23 ***************
salva23 macro
        call   rsal23
        endm

;****** REST23  ***************
rest23  macro
        call   rres23
        endm

;****** USRPRO  ***************
;Intercambia las pantallas de usuario y del programa
usrpro  macro
      mmuestra 0,visible,ratonpresente
        call   rusrp
      mmuestra 1,visible,ratonpresente

        endm
;-------------------------------------------------------------
;Inicio de macros para macros de Usuario
;****** CERRMCR ***************
;Cierra el archivo cuyo handle es MCRHAND.
cerrmcr macro
        call   rcmcr
        endm

;****** CREAMCR ***************
;Crea el archivo cuyo nombre esta dado por el ASCIIZ
;que empieza en DIRNOM. Pone el handle en MCRHAND.
creamcr macro  dirnom
        push   dx
        mov    dx,offset dirnom
        call   rcmc
        pop    dx
        endm

;****** GRABMCR ***************
;Graba en el archivo cuyo handle es MCRHAND los 1088 bytes que
;empiezan en AREAMAC y contienen el indice y datos del macro manejados
;por FLUJO; y los 2048 bytes que empiezan en MCRDAT y que componen
;el area de datos de los macros que el usuario ha dado en respuesta
;a rutinas del programa usuario de FLUJO.
;Codigos de error en AX: 0, todo bien; 1, no hay archivo abierto.
grabmcr macro
        call   rgm
        endm
```

```
;****** GETKEY ***************
;Es la funcion de teclado elemental intervenida para poder
;leer un dato del teclado o del buffer de macros, y poder
;grabar un dato en ese buffer si es necesario. Su salida
;es GETWORD que tiene el codigo ASCII extendido de una tecla.
;Si se estaba grabando macro y se termino el espacio para el
;macro, se pone el byte de interface en FF y MCRERR en 1.
getkey  macro
        call    rgk
        endm

;****** INIMCR ***************
;Recibe a partir de DIRNOM un ASCIIZ con el nombre del archivo
;de macros. Trata de abrir el archivo. Si existe, lo lee en
;AREAMAC (1088 bytes) y en MCRDAT (siguientes 2048 bytes).
;Guarda el handle en MCRHAND.Codigos de error extendidos en AX.
;Si detecta que un archivo no es de macros, sale AX = FF.
;Pone valor AREAMAC+1088 en DIRNMAC. En [DIRNMAC] esta el # del
;macro actual y el siguiente byte es el de interface, que debe
;hacerse FF cuando deba terminarse la grabacion de un macro.
inimcr  macro   dirnom
        push    dx
        mov     dx,offset dirnom
        call    rim
        pop     dx
        endm

;****** LMPMCR ***************
;Limpia las areas de macros
lmpmcr  macro
        call    rlmcr
        endm

;****** RTODO  ***************
rtodo   macro
        pop     eax
        pop     ebx
        pop     ecx
        pop     edx
        pop     esi
        pop     edi
        pop     esp
        pop     ebp
        pop     ds
        pop     es
        endm
```

```
;****** STODO  ***************
stodo  macro
       push   es
       push   ds
       push   ebp
       push   esp
       push   edi
       push   esi
       push   edx
       push   ecx
       push   ebx
       push   eax
       endm


;****** SBRPRO  ***************
sbrpro  macro
;_____________________________________________
;       AREA PARA DATOS MACRO SBRPRO
;_____________________________________________

       .data

gdtabj  db      '0123456789ABCDEFabcdefGgIiPpXxSs:+-' ;para memoria relativa
badinst db      'instruccion incorrecta, intente otra vez',0
dhexseg dd      0  ;almaceno segmento para memoria relativa
dhexoff dd      0  ;almaceno offset para memoria relativa
sign1   db      0  ;ley signos
sign2   db      0  ;ley signos
prevsi  dw      0  ;puntero buffer dhexMr
readerr db      0  ;error al leer catos de dhexMr
posini  dw      0  ;posicion inicial de datos en dhexMr
posfin  dw      0  ;posicion final de dhexMr para fin de lazo
posmedium   db     0  ;0 o 1 ,alerta que se cambio de seg. a offset
signopos    db     0  ;signo para sumar a seg. u offset
signobef    db     0  ;signo para sumar a seg. u offset
                          ;(+ = 0)   (- = 1)
xxxx    dd     0    ;registro para almacenar datos o registros

gccod   db     0          ;codigo de tecla especial pulsada
gdreset db     1          ;GETDATA inicializa variables?
gcascii db     0          ;resultado de GETCHAR
gdtab   db     '0123456789ABCDEFabcdef'
pant    dw     2000 dup(0720h) ;buffers para guardar pantallas (USRPRO)
pantr   dw     2000 dup(0720h)
```

```
pant?  db    1          ;PANT tiene pantalla si PANT?=1
rglx   db    0          ;x,y para getline
rgly   db    0
finbufmcr db  'Fin del buffer y fin del macro. Presiona ESC.',0
buf23  dw    80 dup(0)  ;para guardar la linea 23
soluc  db    1500 dup(0)

;=============================================
;       AREA PARA CODIGO MACRO SBRPRO
;=============================================


        .code

rfinm:  stodo
        salva23          ;salvo linea 23
        limpln  23,revers
        print   1,23,finbufmcr,revers
rfinm1: getkey           ;recibo ESC para salir
        mov     ax,getword
        cmp     al,1bh
        jne     rfinm1
        rest23           ;restauro linea 23
        rtodo
        retn

rgch:   push    ax
        push    dx
rgch1:  mov     dl,1
        getkey           ;leo dato de teclado o buffer
        cmp     mcrerr,1 ;error en macro en GETKEY?
        jne     rgch11
        finmacro
        jmp     rgch1    ;si.
rgch11: mov     ax,getword
        cmp     al,0     ;es tecla especial?
        je      rgch3    ;si.
rgch2:  cmp     al,0dh   ;ENTER?
        je      rgchf
        inc     dl
        cmp     al,9     ;TAB?
        je      rgchf
        inc     dl
        inc     dl
        cmp     al,1bh   ;ESC?
        je      rgchf
```

```
        mov   dl,0          ;es un ASCII valido.
        mov   dh,al
        jmp   rgchf
rgch3:  mov   dl,5
        mov   dh,ah
        cmp   ah,68         ;F10?
        je    rgch31        ;si.
        cmp   ah,15         ;SHIFT TAB?
        jne   rgchf
        mov   dl,3
        jmp   rgchf
rgch31: mov   dl,0
        getkey
        cmp   mcrerr,1      ;error?
        jne   rgch32
        finmacro
        jmp   rgch31
rgch32: mov   ax,getword
        mov   dh,al
        cmp   al,1bh        ;ESC?
        je    rgchf
        cmp   al,0dh        ;ENTER?
        je    rgchf
        cmp   al,9          ;TAB?
        je    rgchf
        jmp   rgch1
rgchf:  mov   gccod,dl
        mov   gcascii,dh
        pop   dx
        pop   ax
        retn

rgd1:   stodo
        cmp   gdreset,1     ;inicializo variables?
        jne   rgd2J         ;no.
        mov   dh,[bx]       ;# de datos pedidos.
        mov   al,0
        mov   [bx+1],al
        mov   si,2
        mov   al,'0'
rgd1J:  mov   [si+bx],al    ;inicializo buffer
        inc   si
        dec   dh
        jnz   rgd1J
      mov     dhexseg,0
        mov   dhexoff,0
```

```
        pusha
        mov    bx,offset dhex3
        mov    dh,[bx]
        mov    al,0
        mov    [bx+1],al
        mov    si,2
        mov    al,'0'
rgd20j: mov    [si+bx],al
        inc    si
        dec    dh
        jnz    rgd20j
        popa

rgd2J:  getkey
        mov    gccod,0

rgd21J: mov    ax,getword
        cmp    al,0          ;tecla especial?
        jne    rgd4J
rgd3J:  mov    gccod,1
        cmp    ah,72         ;flecha arriba?
        jne    rgd38j
        push   si
        mov    si,offset autoseg
        sub    [si],1
        push   eax
        xor    eax,eax
        mov    ax,[si]
        mov    xxxxseg,eax
        pop    eax
        pop    si
        jmp    rgdfj
rgd38j: inc    gccod
        cmp    ah,80         ;flecha abajo?
        jne    rgd37j
        push   si
        mov    si,offset autoseg
        add    [si],1
        push   eax
        xor    eax,eax
        mov    ax,[si]
        mov    xxxxseg,eax
        pop    eax
        pop    si
        jmp    rgdfj

rgd37j: inc    gccod
```

```
        cmp   ah,77        ;->?
        je    rgd2J        ;no hacer nada
        inc   gccod
        cmp   ah,75        ;<-?
        je    rgd2J        ;no hacer nada
        add   gccod,2
        cmp   ah,15        ;SHIFT TAB?
        je    rgdfJ
        add   gccod,2
        cmp   ah,73        ;PGUP?
        jne   rgd35j
        push  si
        mov   si,offset autooff
        sub   [si],68h
        push  eax
        xor   eax,eax
        mov   ax,[si]
        mov   xxxxoff,eax
        pop   eax
        pop   si
        jmp   rgdfj

rgd35j: inc   gccod
        cmp   ah,81        ;PGDN?
        jne   rgd36j
        push  si
        mov   si,offset autooff
        add   [si],68h
        push  eax
        xor   eax,eax
        mov   ax,[si]
        mov   xxxxoff,eax
        pop   eax
        pop   si
        jmp   rgdfj

rgd36j: cmp   ah,83        ;DEL?
        jne   rgd2J
        jmp   rgddJ
rgd4J:  cmp   al,8         ;BS?
        jne   rgd411J
        jmp   rgddJ
rgd411J: cmp  al,0dh       ;ENTER?
        jne   rgd412J
        jmp   rgdeJ
rgd412J: cmp  al,9         ;TAB?
        jne   rgd413J
```

```
        mov   gccod,5
        jmp   rgdfJ
rgd413J: cmp   al,1bh       ;ESC?
        jne   rgd44J
        mov   gccod,7
        jmp   rgdfJ
rgd44J:                 ;hex!
        mov   dh,0          ;veo si dato es correcto.
        mov   si,offset gdtabJ
rgd441J: cmp   al,[si]      ;consta en la tabla?
        je    rgd46J
        inc   si
        inc   dh
        cmp   dh,35         ;acabo la tabla?
        jne   rgd441J
        jmp   rgdbJ         ;no fue valido.
rgd46J:  mov   ax,getword
        mov   dh,al
        call  rgd0J         ;DH=#, BX-->inicio buffer
        cmp   al,1          ;exceso de datos?
        jne   rgd5J
rgdbJ:  push  dx            ;BEEP
        mov   ah,2
        mov   dl,7
        int   21h
        pop   dx
        jmp   rgd2J
rgd5J:  call  rgd01
        jmp   rgd2J
rgddJ:  cmp   byte ptr[bx+1],0      ;DELETE. Buffer vacio?
        je    rgdbJ
        call  rgd02
        call  rgd01  ;Presento dato en ventana (en ascii, no transformacion)
        jmp   rgd2J       ;A esperar por otra tecla

rgdeJ:  push  cx            ;rutina que analiza los terminos ingresados
        push  dx            ;en dhexMR (para acceso a memoria relativa)
        mov   ch,0
        mov   al,[bx+1]
        mov   cl,[bx]
        sub   cl,al
        mov   si,bx
        push  bx
        add   bx,29
        mov   posfin,bx
        pop   bx
        add   si,2
```

```
        inc     si
        lodsb
        cmp     al,'X'
        je      rgd70           ;ax,bx,cx,dx
        cmp     al,'x'
        jne     rgd71
rgd70:  jmp     regpgx
rgd71:  cmp     al,'P'
        je      rgd72           ;bp,ip,sp
        cmp     al,'p'
        jne     rgd73
rgd72:  jmp     regpgp
rgd73:  cmp     al,'I'
        je      rgd74           ;si,di
        cmp     al,'i'
        jne     rgd75
rgd74:  jmp     regpgi
rgd75:  cmp     al,'S'
        je      rgd76           ;cs,ds,es,fs,gs,ss
        cmp     al,'s'
        je      rgd76
        mov     si,prevsi
        lodsb
        dec     si
        jmp     number?
rgd76:  jmp     regseg
number?: cmp    al,'-'          ;entonces se trata de un dato (hex)
        ja      rgd78
        call    rgd0003         ;sumo registros o datos a dhexseg y dhexoff
erase:  pusha                   ;dependiendo de posmedium,borro buffer y salto
        mov     bx,offset dhex3 ;a inicio
        mov     dh,[bx]
        mov     al,0
        mov     [bx+1],al
        mov     si,2
        mov     al,'0'
rgd103: mov     [si+bx],al      ;borro buffer dhex3
        inc     si
        dec     dh
        jnz     rgd103
        popa
        jmp     inicio
rgd78:  cmp     al,':'
        je      dosp            ;se trata entonces de ":"
dhex03:                 ;aqui almaceno maximo 4 #s en hex hasta hallar
        mov     bx,offset dhex3 ;un "+" o "-" sino BEEP y buelve a cond. inic.
rgd4603: mov    dh,al
```

```
        add   si,cx
        mov   posini,si
        mov   cl,[bx+1]
        cmp   cl,0
        jne   inicio1
        mov   readerr,1

fin0;   pop   dx
        pop   cx
        rtodo
        jmp   virus

inicio1: mov  prevsi,si
        dec   prevsi
inicio: mov   si,prevsi      ;inicio de lazo
        inc   si
        cmp   si,posfin
        jne   next0

        mov   bx,offset dhex3
        cmp   byte ptr [bx+1],0
        je    rgd106
        call  rgd0003        ;entonces buffer contiene datos
        pusha                ;borro buffer y salto a inicio
        mov   bx,offset dhex3 ;a ingresar nuevos datos
        mov   dh,[bx]
        mov   al,0
        mov   [bx+1],al
        mov   si,2
        mov   al,'0'
rgd105; mov   [si+bx],al     ;borro buffer dhex3
        inc   si
        dec   dh
        jnz   rgd105
        popa

rgd106;  mov   posini,0
        mov   readerr,0
        mov   posmedium,0
        mov   signopos,0
        mov   signobef,0
        jmp   fin0
next0;  lodsb
        dec   si
        mov   prevsi,si
        cmp   al,':'
        jbe   number?        ;se permite maximo 4 digitos en hex
```

```
        call    rgd003          ;almacena y rota n-1 veces donde n=# datos
        cmp     al,1            ;pedidos (n=4)
        jne     rgd503
fin1:   push    dx              ;exceso de datos
        mov     ah,2
        mov     dl,7
        int     21h
        pop     dx
        print   0,24,badinst,normal
        call    error0
        mov     dhexseg,0
        mov     dhexoff,0
        mov     signobef,0
        mov     signopos,0
        jmp     fin0            ;fin de lazo
rgd503:
        jmp     inicio          ; a leer nuevo dato de dhexMr

RGD003: push    bx              ;rutina que almacena y rota n-1 veces dato
        push    cx              ;de dhexMr si dato en hex
        mov     al,[bx]
        cmp     al,[bx+1]
        mov     al,1
        je      rgd0f03
        inc     byte ptr [bx+1] ;incremento contador
        mov     si,bx
        mov     bx,1
rgd0103: mov    al,[bx+si+2]    ;roto n-1 veces
        mov     [bx+si+1],al
        inc     bl
        cmp     bl,[si]
        jb      rgd0103
        mov     [bx+si+1],dh    ;incluyo nuevo dato
        mov     al,0
rgd0f03: pop    cx
        pop     bx
        retn
DOSP:   cmp     si,posini       ;si al inicio de buffer ":" incorrecto
        je      fin1
        mov     bx,offset dhex3
        cmp     byte ptr [bx+1],0
        je      dosp1           ;entonces fue  registro
        call    rgd0003         ;entonces buffer contiene datos
        pusha                   ;borro buffer y salto a inicio
        mov     bx,offset dhex3
        mov     dh,[bx]
        mov     al,0
```

```
        mov   [bx+1],al
        mov   si,2
        mov   al,'0'
rgd104: mov   [si+bx],al     ;borro buffer dhex3
        inc   si
        dec   dh
        jnz   rgd104
        popa

dosp1:  mov   posmedium,1    ;paso a analizar el offset de mem. rel.
        mov   dhexoff,0
        mov   signopos,0
        mov   signobef,0
        jmp   inicio         ;en buffer dhexMr
RGD0003: pusha               ;sumo registros o datos a dhexseg y dhexoff
        call  signlow        ;dependiendo de posmedium.Borro buffer y salto a inicio
        push  bx
        mov   bx,offset dhex3
        cmp   byte ptr [bx+1],0
        pop   bx
        jne   next9
        popa
        retn                 ;entonces no hacer conversion

next9:  mov   bx,offset dhex3 ;a inicio
        mov   si,bx
        mov   dl,[bx]
        mov   dh,0
        add   si,2
        add   si,dx          ;SI apunta a posicion de datos a convertirse
rgd6103: mov  al,[bx+2]      ; en hex
        call  rgd0303        ;convierto MSB
        mov   cl,4
        shl   dh,cl
        mov   [si+1],dh
        mov   al,[bx+3]
        call  rgd0303
        add   [si+1],dh
        add   bx,2
rgd6203: mov  al,[bx+2]      ;convierto LSB
        call  rgd0303
        mov   cl,4
        shl   dh,cl
        mov   [si],dh
        mov   al,[bx+3]
        call  rgd0303
        add   [si],dh
```

```
        push   eax
        mov    ax,[si]        ; copio 4 bytes
        and    eax,0ffffh
        mov    xxxx,eax
        pop    eax
        cmp    posmedium,0
        je     esleft
        cmp    signobef,0
        je     esmas2
        push   eax
        mov    eax,dhexoff
        sub    eax,xxxx   ;derecha (-)
        mov    dhexoff,eax
        mov    autooff,eax
        pop    eax
        popa
        retn
esmas2: push   eax
        mov    eax,dhexoff
        add    eax,xxxx   ;derecha (+)
        mov    dhexoff,eax
        mov    autooff,eax
        pop    eax
        popa
        retn
esleft: cmp    signobef,0
        je     esmas3
        push   eax
        mov    eax,dhexseg
        sub    eax,xxxx   ;left (-)
        mov    dhexseg,eax
        mov    autoseg,eax
        pop    eax
        popa
        retn
esmas3: push   eax
        mov    eax,dhexseg
        add    eax,xxxx   ;left (+)
        mov    dhexseg,eax
        mov    autoseg,eax
        pop    eax
        popa
        retn

signlow:
        cmp    al,'+'
        je     signos
```

```
        cmp   al,'-'
        je    signos
        push  ax
        mov   al,signopos
        mov   signobef,al
        pop   ax
        retn

signos:
        push  ax          ;ley de signos
        mov   al,signopos
        mov   signobef,al
        pop   ax
        mov   sign1,al
        inc   si
        mov   al,[si]
        cmp   al,'-'

        je    leysignos
        cmp   al,'+'
        je    leysignos
        mov   al,sign1
        cmp   al,'+'
        je    esmas
        mov   signopos,1    ;signo "-"
nosignos: retn
esmas:  mov   signopos,0    ;SIGNO "+"
        retn
leysignos: mov  prevsi,si
        mov   sign2,al
        and   al,sign1
        cmp   al,29h
        je    esmenos
        mov   signopos,0
        push  ax
        mov   al,signopos
        mov   sign1,al
        pop   ax
        jmp   finbuff?
esmenos: mov   signopos,1
        push  ax
        mov   al,signopos
        mov   sign1,al
        pop   ax
finbuff?: push  bx
        add   bx,28
        cmp   bx,si
```

```
        pop     bx
        jne     lazosign
        retn
lazosign: inc           si
        mov     al,[si]
        cmp     al,'+'
        je      leysignos
        cmp     al,'-'
        je      leysignos
        jmp     nosignos
RGD0303: mov    di,offset gdtab      ;busco AL en gdtab y traigo ubicacion
        mov     dh,0
rgd03103: cmp   al,[di]
        je      rgd03203
        inc     di
        inc     dh
        cmp     dh,22
        jne     rgd03103
rgd03203: cmp   dh,16
        jb      rgd03f03
        sub     dh,6
rgd03f03:       retn
REGPGX: inc     prevsi          ;longitud de registro
        mov     bx,offset dhex3     ;se trata de ax,bx,cx,dx
        cmp     byte ptr [bx+1],0   ;error?
        jne     fin1
        push    ax
        mov     al,signopos
        mov     signobef,al
        pop     ax

        dec     si
        dec     si
        lodsb
        cmp     al,'A'
        je      rgd80
        cmp     al,'a'
        jne     regbx?
rgd80:  push    regax
        and     regax,0ffffh
        push    eax
        mov     eax,regax
        mov     xxxx,eax
        pop     eax
        pop     regax
        jmp     rgd0jb
regbx?: cmp     al,'B'
```

```
        je      rgd81
        cmp     al,'b'
        jne     regcx?
rgd81:  push    regbx
        and     regbx,0ffffh
        push    eax
        mov     eax,regbx
        mov     xxxx,eax
        pop     eax
        pop     regbx
        jmp     rgd0jb
regcx?: cmp     al,'C'
        je      rgd82
        cmp     al,'c'
        jne     regdx?
rgd82:  push    regcx
        and     regcx,0ffffh
        push    eax
        mov     eax,regcx
        mov     xxxx,eax
        pop     eax
        pop     regcx
        jmp     rgd0jb
regdx?: push    regdx
        and     regdx,0ffffh
        push    eax
        mov     eax,regdx
        mov     xxxx,eax
        pop     eax
        pop     regdx
        jmp     rgd0jb

REGPGP: inc     prevsi              ;longitud de registro
        mov     bx,offset dhex3     ;registros bp,ip,sp
        cmp     byte ptr [bx+1],0   ;error?
        jne     fin1
        push    ax
        mov     al,signopos
        mov     signobef,al
        pop     ax
        dec     si
        dec     si
        lodsb
        cmp     al,'B'
        je      rgd83
        cmp     al,'b'
        jne     regip?
```

```
rgd83:  push    regbp
        and     regbp,0ffffh
        push    eax
        mov     eax,regbp
        mov     xxxx,eax
        pop     eax
        pop     regbp
        jmp     rgd0jb
regip?: cmp     al,'I'
        je      rgd84
        cmp     al,'i'
        jne     regsp?
rgd84:  push    regip
        and     regip,0ffffh
        push    eax
        mov     eax,regip
        mov     xxxx,eax
        pop     eax
        pop     regip
        jmp     rgd0jb
regsp?: push    regsp
        and     regsp,0ffffh
        push    eax
        mov     eax,regsp
        mov     xxxx,eax
        pop     eax
        pop     regsp
        jmp     rgd0jb

REGPGI: inc     prevsi          ;longitud de registro
        mov     bx,offset dhex3   ; registros di,si
        cmp     byte ptr [bx+1],0   ;error?
        jne     fin1
        push    ax
        mov     al,signopos
        mov     signobef,al
        pop     ax
        dec     si
        dec     si
        lodsb
        cmp     al,'D'
        je      rgd85
        cmp     al,'d'
        jne     regsi?
rgd85:  push    regdi
        and     regdi,0ffffh
        push    eax
```

```
        mov    eax,regdi
        mov    xxxx,eax
        pop    eax
        pop    regdi
        jmp    rgd0jb
regsi?: push   regsi
        and    regsi,0ffffh
        push   eax
        mov    eax,regsi
        mov    xxxx,eax
        pop    eax
        pop    regsi
        jmp    rgd0jb

REGSEG: inc    prevsi              ;longitud de registro
        mov    bx,offset dhex3     ;registros ds,es,fs,gs,ss,cs
        cmp    byte ptr [bx+1],0   ;error?
        jne    fin1
        push   ax
        mov    al,signopos
        mov    signobef,al
        pop    ax
        dec    si
        dec    si
        lodsb
        cmp    al,'D'
        je     rgd86
        cmp    al,'d'
        jne    reges?
rgd86:  push   eax
        xor    eax,eax
        mov    ax,regds
        mov    xxxx,eax
        pop    eax
        jmp    rgd0jb
reges?: cmp    al,'E'
        je     rgd87
        cmp    al,'e'
        jne    regfs?
rgd87:  push   eax
        xor    eax,eax
        mov    ax,reges
        mov    xxxx,eax
        pop    eax
        jmp    rgd0jb
regfs?: cmp    al,'F'
        je     rgd88
```

```
        cmp    al,'f'
        jne    reggs?
rgd88:  push   eax
        xor    eax,eax
        mov    ax,regfs
        mov    xxxx,eax
        pop    eax
        jmp    rgd0jb
reggs?: cmp    al,'G'
        je     rgd89
        cmp    al,'g'
        jne    regss?
rgd89:  push   eax
        xor    eax,eax
        mov    ax,reggs
        mov    xxxx,eax
        pop    eax
        jmp    rgd0jb
regss?: cmp    al,'S'
        je     rgd8a
        cmp    al,'s'
        jne    regcs?
rgd8a:  push   eax
        xor    eax,eax
        mov    ax,regss
        mov    xxxx,eax
        pop    eax
        jmp    rgd0jb
regcs?: push   eax
        xor    eax,eax
        mov    ax,regcs
        mov    xxxx,eax
        pop    eax

RGD0JB: cmp    posmedium,0
        je     esleftj
        cmp    signobef,0
        je     esmas2j
        push   eax
        mov    eax,dhexoff
        sub    eax,xxxx        ;derecha (-)
        mov    dhexoff,eax
        mov    autooff,eax
        pop    eax
        jmp    inicio
esmas2j:push   eax
        mov    eax,dhexoff
```

```
        add     eax,xxxx        ;derecha (+)
        mov     dhexoff,eax
        mov     autooff,eax
        pop     eax
        jmp     inicio
esleftj: cmp    signobef,0
        je      esmas3j
        push    eax
        mov     eax,dhexseg
        sub     eax,xxxx        ;left (-)
        mov     dhexseg,eax
        mov     autoseg,eax
        pop     eax
        jmp     inicio
esmas3j:push    eax
        mov     eax,dhexseg
        add     eax,xxxx        ;left (+)
        mov     dhexseg,eax
        mov     autoseg,eax
        pop     eax
        jmp     inicio
rgdfJ:  rtodo
        jmp     virus

rgd:    stodo
        cmp     gdreset,1       ;inicializo variables?
        jne     rgd2            ;no.
        mov     dh,[bx]         ;# de datos pedidos.
        mov     al,0
        mov     [bx+1],al
        mov     si,2
        mov     al,'0'
rgd1:   mov     [si+bx],al      ;inicializo buffer
        inc     si
        dec     dh
        jnz     rgd1
rgd2:   getkey
        mov     gccod,0
        cmp     mcrerr,1        ;error en macros?
        jne     rgd21
        finmacro
        jmp     rgd2
rgd21:  mov     ax,getword
        cmp     al,0            ;tecla especial?
        jne     rgd4
rgd3:   mov     gccod,1
        cmp     ah,72           ;flecha arriba?
```

```
        je      rgd3f
        inc     gccod
        cmp     ah,80       ;flecha abajo?
        je      rgd3f
        inc     gccod
        cmp     ah,77       ;->?
        je      rgd3f
        inc     gccod
        cmp     ah,75       ;<-?
        je      rgd3f
        add     gccod,2
        cmp     ah,15       ;SHIFT TAB?
        je      rgd3f
        add     gccod,2
        cmp     ah,73       ;PGUP?
        je      rgd3f
        inc     gccod
        cmp     ah,81       ;PGDN?
        je      rgd3f
        cmp     ah,83       ;DEL?
        jne     rgd2
        jmp     rgdd
rgd3f:  jmp     rgdf
rgd4:   cmp     al,8        ;BS?
        jne     rgd411
        jmp     rgdd
rgd411: cmp     al,0dh      ;ENTER?
        jne     rgd412
        jmp     rgde
rgd412: cmp     al,9        ;TAB?
        jne     rgd413
        mov     gccod,5
        jmp     rgdf
rgd413: cmp     al,1bh      ;ESC?
        jne     rgd42
        mov     gccod,7
        jmp     rgdf
rgd42:  cmp     dl,'b'      ;binario?
        jne     rgd44
rgd43:  cmp     al,'0'      ;es '0' o '1'?
        je      rgd431
        cmp     al,'1'
        je      rgd431
        jmp     rgdb        ;dato no valido, beep.
rgd431: jmp     rgd46
rgd44:  cmp     dl,'h'      ;hex?
        jne     rgd45
```

```
        mov   dh,0          ;veo si dato es correcto.
        mov   si,offset gdtab
rgd441: cmp   al,[si]       ;consta en la tabla?
        je    rgd442
        inc   si
        inc   dh
        cmp   dh,22         ;acabo la tabla?
        jne   rgd441
        jmp   rgdb          ;no fue valido.
rgd442: jmp   rgd46         ;'abcdef' = 'ABCDEF'
rgd45:  mov   dh,0          ;decimal. Veo si dato es correcto.
        mov   si,offset gdtab
rgd451: cmp   al,[si]       ;consta en la tabla?
        je    rgd452
        inc   si
        inc   dh
        cmp   dh,10         ;acabo la tabla?
        jne   rgd451
        jmp   rgdb          ;no fue valido.
rgd452: mov   al,[bx]
        cmp   al,[bx+1]     ;buffer lleno?
        jne   rgd4521
        jmp   rgdb
rgd4521:cmp   byte ptr[bx],10
        jne   rgd4530
        push  edx
        push  ecx
        push  edx
        mov   si,bx
        add   si,3
        mov   eax,1000000000
        xor   ecx,ecx
        mov   cl,[si]
        sub   cl,30h
        mul   ecx
      cmp     edx,0         ;<FFFFFFFF?
        je    rgd45a
        jmp   rgd45o
rgd45a: mov   edi,eax
        inc   si
        mov   eax,100000000
        mov   cl,[si]
        sub   cl,30h
        mul   ecx
        add   edi,eax
        jnc   rgd45b
        jmp   rgd45o
```

```
rgd45b: inc   si
        mov   eax,10000000
        mov   cl,[si]
        sub   cl,30h
        mul   ecx
        add   edi,eax
        jnc   rgd45c
        jmp   rgd45o
rgd45c: inc   si
        mov   eax,1000000
        mov   cl,[si]
        sub   cl,30h
        mul   ecx
        add   edi,eax
        jnc   rgd45d
        jmp   rgd45o
rgd45d: inc   si
        mov   eax,100000
        mov   cl,[si]
        sub   cl,30h
        mul   ecx
        add   edi,eax
        jnc   rgd45e
        jmp   rgd45o
rgd45e: inc   si
        mov   eax,10000
        mov   cl,[si]
        sub   cl,30h
        mul   ecx
        add   edi,eax
        jnc   rgd45j
        jmp   rgd45o
rgd45j: inc   si
        mov   eax,1000
        mov   cl,[si]
        sub   cl,30h
        mul   ecx
        add   edi,eax
        jnc   rgd45g
        jmp   rgd45o
rgd45g: inc   si
        mov   eax,100
        mov   cl,[si]
        sub   cl,30h
        mul   ecx
        add   edi,eax
        jnc   rgd45h
```

```
        jmp    rgd45o
rgd45h: inc    si
        mov    eax,10
        mov    cl,[si]
        sub    cl,30h
        mul    ecx
        add    edi,eax
        jnc    rgd45i
        jmp    rgd45o
rgd45i: pop    edx
        mov    cl,dh
        push   edx
        add    edi,ecx
        jc     rgd45o
        inc    si
        mov    [si],edi
        pop    edx
        pop    ecx
        pop    edx
        jmp    rgd46

rgd4530:push   dx
        push   cx
        push   dx
        mov    si,bx
        add    si,3
        cmp    byte ptr[bx],3  ;3 bytes (0-255)?
        je     rgd455
rgd453: mov    ax,10000
        mov    cl,[si]
        mov    ch,0
        sub    cl,30h
        mul    cx
        cmp    dx,0            ;<FFFF?
        je     rgd454
        jmp    rgd45o          ;overflow, beep.
rgd454: mov    di,ax
        inc    si
        mov    ax,1000
        mov    cl,[si]
        sub    cl,30h
        mul    cx
        add    di,ax
        jnc    rgd455
        jmp    rgd45o          ;overflow
rgd455: inc    si
        mov    ax,100
```

```
        mov    cl,[si]
        sub    cl,30h
        mul    cx
        add    di,ax
        jnc    rgd4551
        jmp    rgd45o          ;overflow
rgd4551: cmp   byte ptr[bx],3
        jne    rgd456
        cmp    ah,0
        je     rgd456
        jmp    rgd45o          ;overflow
rgd456: inc    si
        mov    ax,10
        mov    cl,[si]
        sub    cl,30h
        mul    cx
        add    di,ax
        jnc    rgd4561
        jmp    rgd45o          ;overflow
rgd4561: cmp   byte ptr[bx],3
        jne    rgd457
        cmp    ah,0
        je     rgd457
        jmp    rgd45o          ;overflow
rgd457: pop    dx
        mov    cl,dh
        push   dx
        add    di,cx
        jnc    rgd4571
        jmp    rgd45o          ;overflow
rgd4571: cmp   byte ptr[bx],3
        jne    rgd45f
        cmp    ah,0
        je     rgd45f
rgd45o: pop    edx             ;overflow
      pop      ecx
      pop      edx
        jmp    rgdb
rgd45f: inc    si
        mov    [si],di
        pop    dx
        pop    cx
        pop    dx
rgd46:  mov    ax,getword
        mov    dh,al
        call   rgd0            ;DH=#, BX-->inicio buffer
        cmp    al,1            ;exceso de datos?
```

```
        jne     rgd5
rgdb:   push    dx          ;BEEP
        mov     ah,2
        mov     dl,7
        int     21h
        pop     dx
        jmp     rgd2
rgd5:   call    rgd01
        jmp     rgd2
rgdd:   cmp     byte ptr[bx+1],0      ;DELETE. Buffer vacio?
        je      rgdb
        call    rgd02
        call    rgd01
        jmp     rgd2
rgde:   cmp     dl,'h'        ;hex?
        je      rgd6          ;a los binarios no los transformo y
        jmp     rgdf          ;los dec. llegan ya transformados.
rgd6:   mov     si,bx         ;convierto en hex.
        mov     dl,[bx]
        mov     dh,0
        add     si,2
        add     si,dx
        cmp     dl,2
        je      rgd62

rgd61:  cmp     byte ptr [bx],8
        je      rgdE61
        mov     al,[bx+2]     ;convierto MSB
        call    rgd03
        mov     cl,4
        shl     dh,cl
        mov     [si+1],dh
        mov     al,[bx+3]
        call    rgd03
        add     [si+1],dh     ;MSB transformado
        add     bx,2
rgd62:  mov     al,[bx+2]     ;convierto LSB
        call    rgd03
        mov     cl,4
        shl     dh,cl
        mov     [si],dh
        mov     al,[bx+3]
        call    rgd03
        add     [si],dh
rgdf:   rtodo
        retn
```

```
rgdE61:
        mov   al,[bx+2]     ;convierto MSB
        call  rgd03
        mov   cl,4
        shl   dh,cl
        mov   [si+3],dh
        mov   al,[bx+3]
        call  rgd03
        add   [si+3],dh     ;MSB transformado
        mov   al,[bx+4]     ;convierto LSB
        call  rgd03
        mov   cl,4
        shl   dh,cl
        mov   [si+2],dh
        mov   al,[bx+5]
        call  rgd03
        add   [si+2],dh     ;LSB transformado
        add   bx,4

        mov   al,[bx+2]     ;convierto MSB
        call  rgd03
        mov   cl,4
        shl   dh,cl
        mov   [si+1],dh
        mov   al,[bx+3]
        call  rgd03
        add   [si+1],dh
        add   bx,2

        mov   al,[bx+2]     ;convierto LSB
        call  rgd03
        mov   cl,4
        shl   dh,cl
        mov   [si],dh
        mov   al,[bx+3]
        call  rgd03
        add   [si],dh
        rtodo
        retn

rgd0j:  push  bx
      push  cx          ;esto para que reg. EIP solo acepte
        mov   al,[bx]       ;pone DH en el buffer apunt. por BX
        cmp   al,[bx+1]     ;ya no hay espacio?
        mov   al,1
        je    rgd0ff
        inc   byte ptr[bx+1]  ;incremento contador
```

```
        mov   si,bx
        mov   bx,1
rgd0lJ: mov   al,[bx+si+2]   ;rota a la izquierda n-1 veces
        mov   [bx+si+1],al   ;donde n=# de datos pedidos.
        inc   bl
        cmp   bl,[si]
        jb    rgd0lj
rgd0nj: mov   [bx+si+1],dh   ;incluyo nuevo dato.
        mov   al,0
rgd0fj: pop   cx
        pop   bx
        retn

rgd0:   push  bx
        push  cx           ;esto para que reg. EIP solo acepte
        cmp   cual2,0        ;16 bits    (dl=# de reg.)
        jne   rgd010
        mov   al,4           ;esto ya no pues bx contiene el # de
        cmp   al,[bx+1]      ;datos a ser leidos
        mov   al,1
        je    rgd0f
rgd010:
        mov   al,[bx]        ;pone DH en el buffer apunt. por BX
        cmp   al,[bx+1]      ;ya no hay espacio?
        mov   al,1
        je    rgd0f
        inc   byte ptr[bx+1] ;incremento contador
        mov   si,bx
        mov   bx,1
        cmp   byte ptr[si],1
        je    rgd0n
rgd0l:  mov   al,[bx+si+2]   ;rota a la izquierda n-1 veces
        mov   [bx+si+1],al   ;donde n=# de datos pedidos.
        inc   bl
        cmp   bl,[si]
        jb    rgd0l
rgd0n:  mov   [bx+si+1],dh   ;incluyo nuevo dato.
        mov   al,0
rgd0f:  pop   cx
        pop   bx
        retn
rgd01:  push  cx           ;presento buffer en pantalla
        push  dx
        posdir cl,ch
        mov   di,pddir
        mov   bp,di
        mov   cl,[bx]
```

```
        mov    ch,0
        mov    al,'_'          ;primero borro
rgd011: stosb
        inc    di
        loop   rgd011
        mov    al,[bx+1]
        mov    cl,[bx]
        sub    cl,al
        mov    si,bx
        add    si,2
        add    si,cx
        mov    di,bp
        add    cx,cx
        add    di,cx
        mov    cl,[bx+1]
        cmp    cl,0
        je     rgd013
rgd012: lodsb                  ;muestro el numero
        stosb
        inc    di
        loop   rgd012
rgd013: pop    dx
        pop    cx
        retn
rgd02:  push   bx              ;rota a la derecha
        dec    byte ptr[bx+1]  ;disminuyo contador
        mov    si,bx
        mov    bl,[si]
        mov    bh,0
        cmp    bl,1
        je     rgd022
rgd021: mov    al,[bx+si]
        mov    [bx+si+1],al
        dec    bx
        cmp    bl,1
        jne    rgd021
rgd022: mov    byte ptr[bx+si+1],'0'
        pop    bx
        retn
rgd03:  mov    di,offset gdtab ;Busco AL en GDTAB
        mov    dh,0            ;y traigo ubicacion en DH.
rgd031: cmp    al,[di]
        je     rgd032
        inc    di
        inc    dh
        cmp    dh,22
        jne    rgd031
```

```
rgd032: cmp   dh,16
        jb    rgd03f
        sub   dh,6
rgd03f: retn

rgl:  push    di          ;recibo un string desde teclado
      push    cx          ;SI apunta al inicio del buffer
      push    bx
      mov     rglx,cl
      mov     rgly,ch
      cmp     gdreset,1     ;inicializo el buffer?
      jne     rgla
      mov     di,si       ;limpia el buffer
      inc     di
      mov     cl,[si]
      mov     ch,0         ;CX=# car.
      inc     cx
      mov     ax,ds
      mov     es,ax
      mov     al,0
      rep     stosb
      mov     es,videoseg
rgla: mov     bh,0
rgl1: mov     cl,[si+1]
      call    curs24        ;pongo cursor
      getchar             ;recibo un dato del teclado
      mov     al,atrprev    ;quito cursor
      mov     di,dirprev
      stosb
      mov     cl,[si]       ;num. maximo de caracteres
      mov     bl,[si+1]     ;num. actual de caracteres
      cmp     gccod,5       ;car. especial?
      je      rgl2
      cmp     gccod,1       ;ENTER?
      je      rgl3
      cmp     gccod,0       ;ascii normal?
      je      rgl11
      jmp     rgl3          ;TAB, SHIFT TAB y ESC: salir.
rgl11: mov    al,gcascii
      cmp     al,8          ;Backspace?
      je      rgld
      cmp     bl,cl         ;fin del buffer?
      je      rglb
      mov     [bx+si+2],al   ;no, guardo dato
      mov     cl,bl         ;eco a pantalla
      add     cl,rglx
      mov     ch,rgly
```

```
        posdir cl,ch
        mov    di,pddir
        mov    al,gcascii
        stosb
        inc     byte ptr[si+1]
        jmp    rgl1
rglb:   mov    dl,7          ;BEEP
        mov    ah,2
        int    21h
        jmp    rgl1
rgl2:   cmp    gcascii,83     ;DELETE?
        je     rgld
        jmp    rgl3
rgld:   cmp    bl,0          ;puedo seguir borrando?
        je     rglb          ;no,beep
        dec    byte ptr[si+1]
        mov    byte ptr[bx+si+1],0
        mov    cl,bl         ;borro en pantalla
        add    cl,rglx
        dec    cl
        mov    ch,rgly
        posdir cl,ch
        mov    di,pddir
        mov    al,' '
        stosb
        jmp    rgl1
rgl3:   pop    bx
        pop    cx
        pop    di
        retn

curs24: push   cx
        push   di
        add    cl,rglx
        mov    ch,rgly        ;encuentro posicion del cursor
        posdir cl,ch          ;altero AX,DI
        mov    di,pddir
        inc    di
        mov    ah,es:[di]
        mov    atrprev,ah
        mov    dirprev,di
        mov    al,revers
        stosb
        pop    di
        pop    cx
        retn
```

```
rll:    push  cx
        push  di
        posdir 0,al
        mov   di,pddir      ;dir de inicio de linea
        mov   al,' '        ;AL tiene el ' ' y AH el atributo
        mov   cx,80
        rep   stosw         ;borra linea
        pop   di
        pop   cx
        retn

rlmp:   push  di
        push  ax
        push  cx
        mov   di,0

        mov   cx,2000
        mov   ax,0720h      ;car: espacio, atributo: normal.
        rep   stosw         ;borra pantalla.
        pop   cx
        pop   ax
        pop   di
        retn

rprt:   push  di
        mov   di,pddir      ;direccion de inicio
rprt1:  lodsb
        cmp   al,0          ;termino?
        je    rprt2
        stosw               ;AL tiene el ASCII, AH el atributo
        jmp   rprt1
rprt2:  pop   di
        retn

rsal23: push  es
        push  ds
        pop   es
        mov   ds,videoseg
        mov   si,23*160
        mov   di,offset buf23
        mov   cx,80
        rep   movsw
        mov   ax,@data
        mov   ds,ax
        mov   es,videoseg
        pop   es
        retn
```

```
rres23: mov   si,offset buf23
        mov   di,23*160
        mov   cx,80
        rep   movsw
        retn

rusrp:
        push  si          ;Si PANT?=1, la pantalla a mostrarse esta en
        push  di          ;PANT, sino en PANTR. Uno de estos buffers
        push  cx          ;es cambiado con la memoria de video.
        mov   si,0
        cmp   pant?,1     ;datos actuales en PANT ?
        jne   rusrp1
        mov   di,offset pantr ;si
        jmp   rusrp2
rusrp1: mov   di,offset pant  ;no, datos actuales en PANTR
rusrp2: mov   ax,@data    ;guardo la pantalla en
        mov   es,ax       ;el buffer que no esta en uso.
        mov   ds,videoseg
        mov   cx,2000
        rep   movsw
        mov   di,0        ;PANT o PANTR a pantalla
        mov   ax,@data
        mov   ds,ax
        mov   es,videoseg
        cmp   pant?,1     ;datos actuales en PANT ?
        jne   rusrp3
        mov   si,offset pant  ;si
        jmp   rusrp4
rusrp3: mov   si,offset pantr ;no, en PANTR
rusrp4: mov   cx,2000
        rep   movsw

rusrp6: xor   pant?,1     ;toggle bandera PANT?
        pop   cx
        pop   di
        pop   si
        retn

        endm

;_____________________________________________________________
;Subrutinas para macros de manejo de macros de Usuarios
;****** SBRMCR ***************
sbrmcr  macro
        .data
mcrdat  db    2048 dup(0)   ;datos manejados por programa usuario
```

```
control     db    0,0       ;Para ver si archivo es de macros (FO)
dirnmac     dw    0         ;direccion # de macro actual
mcrhand     dw    0         ;handle de archivo de macros
getword     dw    0         ;salida de GETKEY
mcrerr      db    0         ;para sacar codigo de error.

        .code
rcmcr:  stodo
        mov   bx,mcrhand    ;obtengo el handle y
        mov   ah,3eh        ;cierro el archivo
        int   21h
        rtodo
        retn

rcmc:   stodo
        mov   ah,3ch        ;creo archivo de macros.
        mov   cx,0
        int   21h
        mov   mcrhand,ax
        rtodo
        retn

rgm:    stodo
        cmp   mcrhand,0     ;habia archivo de macros abierto?
        jne   rgm1          ;si, continuo.
        mov   mcrerr,1      ;no salgo con codigo de error
        jmp   rgmf
rgm1:   mov   control,'F'
        mov   control+1,'O'
        mov   ah,42h        ;fijo puntero de R/W
        mov   cx,0          ;al inicio del archivo.
        mov   dx,0
        mov   bx,mcrhand
        mov   al,0
        int   21h
        mov   cx,1088       ;escribo primera parte del archivo
        mov   dx,offset areamac
        mov   ah,40h
        int   21h
        mov   cx,2050       ;escribo segunda parte
        mov   dx,offset mcrdat
        mov   ah,40h
        int   21h
        mov   mcrerr,0
rgmf:   rtodo
        mov   al,mcrerr
        mov   ah,0
```

```
        retn

rgk:    stodo
        mov     mcrerr,0
        mov     bx,dirnmac      ;veo si puedo entrar a macros
        cmp     byte ptr[bx+1],0ffh
        je      rgkn
        cmp     bgmac,0         ;grabando macros?
        jne     rgk2            ;si.
        cmp     bmac,0          ;no, corriendo o probando macros?
        jne     rgk1            ;si.
rgkn:   call    rgk0            ;no, rutina normal de teclado.
        jmp     rgkf
rgk1:   mov     si,offset mcrdat        ;leo dato de buffer
        mov     bx,dirnmac
        mov     dh,[bx]
        mov     dl,0            ;DX=# de macro * 256
        add     si,dx           ;SI apunta a macro actual
        mov     dl,[bx+1]
        mov     bl,dl
        mov     bh,0
        add     bx,bx
        mov     ax,[si+bx]
        mov     getword,ax      ;paso dato como leido de teclado.
        mov     bx,dirnmac
        inc     byte ptr[bx+1]
        cmp     byte ptr[bx+1],80h      ;fin?
        je      rgk3
        cmp     bver,0          ;probando?
        je      rgkf            ;no, salgo.
        push    getword         ;si,espero ENTER para confirmar.
rgk11:  call    rgk0
        mov     ax,getword      ;es ENTER?
        cmp     al,0dh
        jne     rgk11           ;no, espero un ENTER.
        pop     getword         ;si, salgo.
        jmp     rgkf
rgk2:   mov     si,offset mcrdat        ;leo dato de teclado
        mov     bx,dirnmac
        mov     dh,[bx]
        mov     dl,0            ;DX=# de macro * 256
        add     si,dx           ;SI apunta a macro actual
        mov     dl,[bx+1]
        mov     bl,dl
        mov     bh,0
        add     bx,bx
        call    rgk0            ;guardo dato en el buffer
```

```
        mov     ax,getword
        mov     [si+bx],ax
        mov     bx,dirnmac
        inc     byte ptr[bx+1]
        cmp     byte ptr[bx+1],80h      ;termino area de macros?
        jb      rgkf                    ;no.
rgk3:   mov     byte ptr[bx+1],0ffh     ;si, mando estos valores como
        mov     mcrerr,1                ;indicacion de fin de macro
rgkf:   rtodo
        retn

rgk0:
    stodo
    cmp     vent,0
    jc repetirrgk0
    mov     al,ventnueva
    cmp     al,ventactual
    je      repetirrgk0
    mov     ax,0f09h
    jmp     finmgk0
repetirrgk0:
    mov     ah,0bh
    int     21h
    cmp     al,0
    je      verratonmgk0

    mov     ah,0        ; espera que se pulse una tecla.
    int     16h
    cmp     ax,0f09h
    jne     notab
    inc     Ventnueva
    cmp     Ventnueva,6
    jne     finmgk0
    mov     ventnueva,0
    jmp     finmgk0
notab:
    cmp     ax,0f00h
    jne     finmgk0
    dec     VentNueva
    cmp     VentNueva,0ffh
    jne     finmgk0
    mov     VentNueva,5
    jmp     short finmgk0
verratonmgk0:
    call    pantalla
    cmp     bx,0ffh
    je      repetirrgk0
```

```
finmgk0:
      mov    getword,ax
      rtodo
        ret

rim:   stodo              ;intento abrir el archivo
        mov    bx,offset areamac
        add    bx,1088        ;inicializo DIRNMAC
        mov    dirnmac,bx
        mov    ah,3dh
        mov    al,02h
        int    21h
        jnc    rim1           ;todo bien
        mov    ah,59h         ;error, veo de que tipo es.
        mov    bx,0
        int    21h
        mov    mcrerr,al      ;salgo con codigo de error extendido
        jmp    rimf
rim1:  mov    mcrhand,ax
        mov    bx,ax
        mov    ah,42h         ;veo si es de macros
        mov    cx,0
        mov    dx,3136
        mov    al,0
        int    21h
        mov    cx,2
        mov    bx,mcrhand
        mov    dx,offset control
        mov    ah,3fh         ;READ
        int    21h
        cmp    control,'F'
        jne    rim11
        cmp    control+1,'O'
        je     rim12
rim11: mov    ah,3eh         ;no paso chequeo
        mov    bx,mcrhand     ;cierro archivo
        int    21h
        mov    mcrerr,0ffh
        jmp    rimf
rim12: mov    bx,mcrhand     ;es de macros
        mov    ah,42h         ;LSEEK
        mov    cx,0
        mov    dx,0
        mov    al,0
        int    21h
        mov    dx,offset areamac
        mov    cx,1088
```

```
        mov   ah,3fh        ;READ
        int   21h
        mov   dx,offset mcrdat
        mov   cx,2050       ;leo segunda parte del archivo
        mov   ah,3fh
        int   21h
        mov   mcrerr,0
rimf:   rtodo
        mov   al,mcrerr
        mov   ah,0
        retn

rlmcr:  push  di
        push  cx
        push  ax
        mov   di,offset areamac
        mov   cx,1088
        mov   al,0
        push  es
        push  ds
        pop   es
        rep   stosb
        mov   di,offset mcrdat
        mov   cx,2050
        rep   stosb
        pop   es
        pop   ax
        pop   cx
        pop   di
        retn

        endm
```

MACRO PROVNT.ASM

```
;
;_____________________________________
;  Rutinas para manejo de ventanas
;
;_____________________________________
;-------------------------------------------------------------
;Macro  Parametros          Funcion
;-------------------------------------------------------------
;hexasc                     hex en AL ==> ascii en BX
;idvnt  dirasc,dirres,longres  identifica archivo y area de ventanas
;popall                     pop a todo
;posdir x,y                  calcula direccion en memoria de video
;pushall                    push a todo
;vnt   nombre,tipo           muestra o actualiza una ventana
;sbrvnt                     subrutinas de los otros macros
;-------------------------------------------------------------

;****** HEXASC ***************
;El byte en AL se convierte en dos bytes ASCII en BX.
hexasc  macro
        call   rha
        endm

;****** IDVNT  ***************
;Este macro recibe el nombre del archivo de ventanas en un ASCIIZ que
;inicia en DIRASC. Abre el archivo, trae todos los grupos principales
;al area que empieza en DIRRES y guarda DIRRES y LONGRES en el indice.
;En AX salen codigos de error: 00 todo bien, 01 error al abrir archivo,
;02 si se encuentra que no es un archivo de ventanas, y
;03 error por falta de espacio (LONGRES= 6 grupos para indice + # de
;grupos principales + 1 grupo para una tabla de residencia actual + al
;menos 4 grupos para secundarios, a menos que todas las ventanas sean
;principales). Nota: Un grupo=256 bytes.
idvnt   macro  dirasc,dirres,longres
        push    dx
        push    di
        push    bx
        mov     dx,offset dirasc
        mov     di,offset dirres
        mov     bl,longres
        call    riv
        pop     bx
        pop     di
        pop     dx
```

```
        endm

;****** POPALL ***************
popall  macro
        pop     eax
        pop     ebx
        pop     ecx
        pop     edx
        pop     esi
        pop     edi
        pop     ebp
        pop     esp
        pop     ds
        pop     es
        endm

;****** POSDIR ***************
;Convierte la posicion (X,Y) en una direccion en
;PDDIR para acceso directo a la memoria de video.
posdir  macro   x,y
        push    cx
        mov     cl,x
        mov     ch,y
        call    rpd
        pop     cx
        endm

;****** PUSHALL ***************
pushall macro
        push    es
        push    ds
        push    esp
        push    ebp
        push    edi
        push    esi
        push    edx
        push    ecx
        push    ebx
        push    eax
        endm

;****** VNT    ***************
;BX apunta al nombre de la ventana. Si existe y es principal, sus
;grupos en el area de residentes son los mismos que tenia en el
;archivo. Si es secundaria ve si es residente, caso en que sus grupos
;estan en la entrada en el indice. Si no es residente, se la trae al
;area de residentes y se la marca como tal, borrando la residencia
```

```
;de las ventanas sobre las que esta ventana se coloque. Muestra la
;ventana y donde se encuentren variables se las leen del area apuntada
;por AX:SI. Existen dos TIPOs de llenado: 'h', hexadecimal, c/byte
;se muestra en dos variables de la ventana y 'a', ASCII, c/byte se
;muestra como tal en pantalla.
;En AX sale un codigo de error: 0, todo bien; 1, nombre no existe.
vnt    macro  tipo
       mov    tipvnt,tipo
       call   rvnt
       endm

;****** SBRVNT ***************
sbrvnt macro
tabha  db     '0','1','2','3','4','5','6','7','8','9'
       db     'A','B','C','D','E','F'
rha:   push   ax
       push   cx
       push   dx
       mov    bx,offset tabha
       mov    dl,al
       and    al,0f0h       ;nibble mas significativo
       mov    ah,0
       mov    cl,16         ;lo paso a la parte menos signif.
       div    cl
       push   ds           ;hago DS=CS para que funcione el XLAT
       mov    cx,cs
       mov    ds,cx
       xlat                ;traduzco a ASCII
       mov    ch,al
       mov    al,dl
       and    al,0fh        ;nibble menos significativo
       xlat                ;Traduzco a ASCII
       pop    ds
       mov    cl,al
       mov    bx,cx
       pop    dx
       pop    cx
       pop    ax
       retn

dirind dw     0             ;direccion del indice
riv:   push   si
       push   cx
       mov    si,bx
       mov    ah,3dh        ;abro archivo
       mov    al,0
       int    21h
```

```
        jnc     riv1            ;error.
        mov     ax,1
        jmp     rivf
riv1:   mov     bx,ax           ;handle
        mov     ah,3fh          ;leo entradas 0, 1 y 2.
        mov     dx,di
        mov     cx,72
        int     21h
        mov     [di+13],bx      ;guardo handle
        mov     bx,si
        mov     [di+12],bl      ;guardo LONGRES
        mov     [di+10],di      ;guardo DIRRES
        mov     dirind,di
        mov     bx,24           ;chequeo paridad del archivo
        mov     cl,8
        mov     al,0
riv11:  add     al,[di+bx]
        inc     bx
        dec     cl
        jnz     riv11
        cmp     al,[di+bx]
        je      riv12
        mov     ax,2            ;paridad de ventanas no se cumple
        jmp     rivf
riv12:  mov     bx,36
        mov     cl,32
        mov     al,0
riv13:  add     al,[di+bx]
        inc     bx
        dec     cl
        jnz     riv13
        cmp     al,[di+bx]
        je      riv2
        mov     ax,2            ;paridad de grupos no se cumple
        jmp     rivf
riv2:   mov     al,[di+9]       ;# de grupos principales
        cmp     al,[di+1]       ;= total de grupos?
        je      riv21
        add     al,4            ;+4 grupos minimo de secundarias
riv21:  add     al,6            ;+6 de indice
        inc     al              ;+1 para tabla de residencia
        cmp     al,[di+12]
        jbe     riv3            ;01/05/89
        mov     ax,3            ;muy poco espacio
        jmp     rivf
riv3:   mov     dx,di           ;todo bien, leo resto del indice
        add     dx,72
```

```
        mov   cx,1536-72
        mov   bx,[di+13]    ;handle
        mov   ah,3fh
        int   21h
riv4:   mov   dx,di         ;leo todos los principales
        add   dx,1536
        mov   ch,[di+9]     ;CX=# de grupos principales*256
        mov   cl,0          ;= # de bytes por leer.
        mov   ah,3fh
        int   21h
        mov   ch,[di+9]     ;preparo numero de proximo grupo
        add   ch,6          ;a usarse de area de residentes
        mov   [di+15],ch
        mov   ch,[di+8]     ;copio los # de grupos en el archivo
        add   di,48         ;a los grupos en memoria para
        mov   bp,di         ;las CH ventanas principales
riv5:   add   bp,24         ;y las marco como residentes
        mov   di,bp
        mov   byte ptr[di+8],3
        mov   bx,[di+16]
        mov   [di+20],bx
        mov   bx,[di+18]
        mov   [di+22],bx
        dec   ch
        jnz   riv5
        mov   ax,0
rivf:   pop   cx
        pop   si
        retn
pddir   dw    0
rpd:    push  ax
        push  cx
        mov   al,160        ;80*2 bytes por linea
        mul   ch
        add   cl,cl         ;+ 2 veces el # de columna
        mov   ch,0
        add   ax,cx
        mov   pddir,ax
        pop   cx
        pop   ax
        retn
segvar  dw    0             ;segmento donde estan las variables
offvar  dw    0             ;offset de la variable actual
tipvnt  db    0             ;tipo: 'h', 'a' o 't'
dirent  dw    0             ;direccion de entrada de vent. actual
numgrp  db    0             ;# de grupos que faltan por traer
columna db    0             ;columna escrita actualmente
```

```
linea  db     0          ;linea escrita actualmente
numbyte dw     0          ;# de byte mostrado del grupo actual

rvnt:  pushall
       mov    segvar,ax
       mov    offvar,si
       mov    di,dirind     ;direccion del indice
       mov    dl,[di+2]     ;ultima ventana valida
       add    di,48         ;voy a buscar el nombre en las entradas
       push   es            ;del indice.
       push   ds
       pop    es
       mov    dh,2
       mov    bp,di
rvnt1: add    bp,24
       mov    di,bp         ;ubico entrada indicada por
       inc    dh            ;DH
       mov    si,bx
       mov    cx,9          ;nombre de 8 bytes
rvnt10: mov   ah,es:[di]    ;leo 1 byte del nombre de una ventana
       inc    di
       lodsb                ;leo 1 byte del nombre buscado
       dec    cx
       jz     rvnt11        ;nombres iguales?
       cmp    al,ah
       jne    rvn10         ;nombres diferentes
       cmp    al,0          ;iguales?
       je     rvnt11
       jmp    rvnt10
rvn10: cmp    dh,dl         ;busco en todas las ventanas?
       jbe    rvnt1         ;no, sigo.01/05/89
       pop    es            ;si, nombre no existe,
       mov    numbyte,1     ;error = 1
       jmp    rvntf
rvnt11: pop   es
       mov    di,dirind
       mov    [di+6],dh     ;guardo # ventana actual
       mov    al,24
       mul    dh            ;ubico entrada actual y
       add    di,ax         ;DS:DI apunta a ventana actual
       mov    dirent,di
       mov    dh,[di+8]     ;veo si es P o S.
       cmp    dh,3
       jne    rvnt2         ;S.
       jmp    rvnt4         ;P.
rvnt2: cmp    dh,2          ;secundaria. Reside?
       jne    rvnt20
```

```
        jmp   rvnt4          ;si.
rvnt20: mov   dl,[di+9]      ;no, ubicar la ventana.
        mov   byte ptr[di+8],2       ;doy residencia a la ventana
        mov   si,dirind      ;DS:SI apunta al indice
        mov   ah,[si+12]     ;ubico tabla de residencia
        dec   ah             ;AX tiene el offset de la tabla
        mov   al,0           ;respecto del inicio del indice
        mov   bp,si
        add   bp,ax          ;DS:BP apunta a la tabla
        mov   cl,[si+15]     ;CL = # grupo que va a ser dado
        mov   bx,16          ;a la ventana actual
rvnt21: cmp   byte ptr[di+bx],0      ;termino ventana?
        je    rvnt3          ;si.
        mov   [di+bx+4],cl
        push  bp             ;no, leer valor en tabla
        add   bp,cx
        mov   al,ds:[bp]
        cmp   al,[si+6]      ;01/05/89: No quitarse la residencia
        je    rvnt22         ;a si misma.
        cmp   al,0           ;grupo estaba libre?
        je    rvnt22         ;si.
        push  si             ;no, quito residencia a la ventana
        mov   ah,24
        mul   ah
        add   si,ax
        mov   byte ptr[si+8],0
        pop   si
rvnt22: mov   al,[si+6]      ;ventana actual
        mov   ds:[bp],al     ;a la tabla de residencia
        pop   bp
        inc   cl             ;preparo proximo grupo a usarse
        mov   ah,[si+12]
        sub   ah,2
        cmp   cl,ah
        jbe   rvnt23         ;01/05/89
        mov   ah,[si+9]      ;termino area de residentes
        add   ah,6           ;==> voy a inicio de area de
        mov   cl,ah          ;secundarias
rvnt23: mov   [si+15],cl
        inc   bl             ;ya reviso cuatro grupos?
        cmp   bl,20
        jne   rvnt21         ;no, volver.
rvnt3:  mov   bx,16
rvnt31: mov   dh,[di+bx]     ;leo # de grupo
        cmp   dh,0           ;termino?
        je    rvnt4          ;si.
        mov   dl,0           ;no. CX:DX=offset desde
```

```
        mov   cx,0          ;inicio de archivo
        mov   al,0
        mov   ah,42h        ;LSEEK
        push  bx
        mov   bx,[si+13]
        int   21h
        pop   bx
        mov   dh,[di+bx+4]  ;DS:DX apunta a buffer de lectura
        mov   dl,0          ;Leo un grupo del disco
        add   dx,si
        mov   cx,256
        mov   ah,3fh        ;READ
        push  bx
        mov   bx,[si+13]
        int   21h
        pop   bx
        inc   bl
        cmp   bl,20         ;ya leyo 4 grupos?
        jne   rvnt31        ;no, sigo leyendo.
rvnt4:  mov   ax,[di+12]    ;esq. sup. izq.
        mov   linea,ah
        mov   columna,al
        mov   al,[di+9]     ;# de grupos
        mov   numgrp,al
        mov   numbyte,0     ;# de byte actual
        mov   si,dirind
        mov   ah,[di+20]    ;AX posicion primer grupo
        mov   bp,di
        mov   al,0
        add   si,ax         ;SI apunta al grupo
        posdir columna,linea
        mov   di,pddir
rvnt41: lodsw
        cmp   al,1          ;variable?
        je    rvnt42        ;si.
        stosw               ;no, mostrar caracter.
        jmp   rvnt44
rvnt42: push  ds
        mov   bx,offvar     ;leo una variable
        mov   ds,segvar
        mov   al,[bx]
        pop   ds
        inc   offvar
        cmp   tipvnt,'h'    ;ventana hexadecimal?
        je    rvnt43        ;si.
        stosw               ;no, ventana ascii.
        jmp   rvnt44
```

```
rvnt43: hexasc              ;AL se convierte en ascii en BX
        mov   al,bh
        stosw               ;muestro mitad mas significativa.
        lodsw               ;Leo atributo siguiente.Supongo
        mov   al,bl          ;que AL=1 (otra variable) y
        stosw               ;muestro la otra mitad.
        inc   columna
        mov   dx,si          ;Con DX vere si hubo cambio de grupo.
        call  rvnt0          ;ve si grupo actual ya termino.
        cmp   dx,si          ;Hubo cambio de grupo?
        je    rvn431          ;no, SI apunta correctamente.
        add   si,2           ;si, hago que SI apunte bien.
rvn431: cmp   al,1           ;termino la ventana?
        je    rvnt5          ;si, salir.
rvnt44: call  rvnt0          ;no, seguir.
        cmp   al,1           ;termino ventana?
        je    rvnt5          ;si.
        inc   columna
        mov   al,ds:[bp+14]  ;termino linea?
        cmp   al,columna
        jae   rvnt41         ;no, continuar.01/05/89
        inc   linea          ;si, nueva linea.
        mov   al,ds:[bp+12]  ;primera columna.
        mov   columna,al
        posdir  columna,linea
        mov   di,pddir
        mov   al,ds:[bp+15]  ;termino ventana?
        cmp   al,linea
        jb    rvnt5
        jmp   rvnt41         ;no, continuar.
rvnt5:  mov   numbyte,0      ;si, salir.
rvntf:  popall
        mov   ax,numbyte
        retn
rvnt0:  push  bp             ;varia contadores y ve si es necesario
        add   numbyte,2      ;empezar otro grupo. Altera AX.
        cmp   numbyte,100h   ;termino un grupo?
        jb    rvnt02         ;no.
        dec   numgrp         ;si, un grupo menos por mostrar
        cmp   numgrp,0       ;termino?
jne   rvnt01          ;no.
mov   al,1            ;si, salgo con AL=1.
jmp   rvnt03
rvnt01: mov   ah,numgrp      ;leo un # de grupo
mov   al,ds:[bp+9]
sub   al,ah
mov   ah,0
```

```
add     bp,ax
mov     ah,ds:[bp+20]   ;AX=posicion de nuevo grupo
mov     al,0
mov     numbyte,0
mov     si,dirind
add     si,ax           ;SI apunta a nuevo grupo
rvnt02: mov     al,0            ;AL=0 ==> no termina aun
rvnt03: pop     bp
        retn

        endm
```

MACROS MOUSE.ASM

```
;
/*************************************************************************
******
; * Archivo:    MOUSE.ASM                                          *
; * Descripción: Macros para interfase de ratón (modo texto )      *
; * Lenguaje Ensamblador                                           *
; * Traducido del código fuente escrito en el libro:               *
; * EZZEL, Ben, "Turbo C++ Programming: An Object-Oriented Approach",  *
; *          Addison-Wesley, USA, 1990.    ISBN: 0-201-55023-7   *
; * Historia:                                                      *
; *    Autor.      Versión     Modificaciones                      *
; *    ======      =======     ==============================
*
; *    a.t.        1.0         Conversión a lenguaje Ensamblador *
;
*************************************************************************
*****/
;Para usar éste macro es necesario que el ratón se encuentre previamente
;instalado con un controlador (mouse.com o mouse.sys).
;Existen funciones para el ratón en modo gráfico que son similares a las que
;se presentan aquí a escepción de la función para la forma del cursor.
;También de acuerdo al fabricante del ratón existen otras funciones adiciona-
;les. Aquí solo se incluyen las que generalmente está presentes en todo con-
;trolador.
;El ratón se utiliza con la Interrupción 51 (33h) del Bios, en caso de no
;estar instalado el mouse la dirección de salto de ésta interrupción estará
;con un IRET o no contendrá nada, por lo que en ese caso no se recomienda que
;se ejecuten las funciones del raton aquí listadas. Para controlar la presen-
;cia o no del ratón en el sistema se usa la variable PRESENTEX (x valor de 1
;a 10) en cada uno
;de los macros de tal forma que si PresenteX es cero no se ejecuta el macro
;PRESENTE1 se pone en Uno en el macro Mreseteo que es el que detecta la pre-
;sencia del ratón.

;;*****************************************

;Forma de llamar a cada macro
;       mreseteo    Nombre1,arg2,arg3   (arg2 debe ser variable tipo word)
;

;       Mmuestra    arg1,arg2           (arg1 puede ser variable o valor
```

```
;                               inmediato, arg2 debe ser variable
;                               ambas tipo word)
;       mposicion       Nombre2,arg3
;       mmovera         Nombre1,arg3
;         mpresionado     arg1,Nombre3,arg3 (arg1 variable o valor inmediato tipo
word)
;       mliberado       arg1,Nombre3,arg3 (arg1 variable o valor inmediato tipo word)
;       Mxlimit         Nombre1,arg3
;       Mylimit         Nombre1,arg3
;         formacursor     arg1,Nombre1,arg3 (arg1 word valor inmediato, registro,
variable)
;       mmovimiento     Nombre1,arg3
;       mbrillante      arg1,arg3       (arg1 variable o valor inmediato tipo word)
;       MoveRatio       Nombre1,arg3
;       mocultar        Nombre3,arg3
;       mvelocidad      arg1,arg3       (arg1 variable o valor inmediato tipo word)
;
;En todos los casos "arg3" es una variable tipo byte que se define en el pro-
;cedimiento que va a usar estos macros, se usa para indicar la presencia o
;no del ratón, para condicionar la ejecución o no de los macros.
;
;************************************************
; Para emplear los macros defina variables (en el procedimiento desde el que
; va ha usar los macros) del siguiente tipo:
;
;* Para los macros: Mreseteo,Mmovera,MXlimit,MYlimit,Mmovimiento,Moveratio
;* ????   STRUCT   ;???? indica cualquier nombre
;*       var11 dw 0  ;
;*       var12 dw 0  ;
;* ????   ENDS
;* nombre1 ???? <0,0>  ; esta línea realmente crea la variable "nombre1" tipo
;*                 ; estructura ???? con valores iniciales 0
;*
;* Para los Macros: Mposicion, FormaCursor
;* ???? STRUCUT      ;???? indica cualquier nombre
;*       var21 dw 0
;*       var22 dw 0
;*       var23 dw 0
;* ????   ENDS
;* nombre2 ???? <0,0,0> ; esta línea realmente crea la variable "nombre2" tipo
;*                 ; estructura ???? con valores iniciales 0
;*
;* Para macros: MPresionado,Mliberado,Mocultar
;* ???? STRUCUT      ;???? indica cualquier nombre
;*       var31 dw 0
;*       var32 dw 0
;*       var33 dw 0
```

```
;*      var34 dw 0
;* ????   ENDS
;* nombre3 ???? <0,0,0,0> ;esta línea realmente crea la variable "nombre3" tipo
;*                  ;estructura ???? con valores iniciales 0



; macro:       mreseteo
; Descripción: Llama al driver del ratón, reseteando el driver a las
;              condiciones por defecto y retornando el estado del ratón
;              en el registro AX (-1 *ffffh* si el ratón está presente,
;              0 si no está disponible). Si el ratón está presente,
;              el cursor del ratón se activa. El registro BX retorna el
;              número de botones del ratón (2 o 3). Ambos resultados son
;              retornados en la estructura Result (present,buttons).
;  Entrada   : Ninguno
;  Salida    : Resultado: var11 -> Mouse presente (FFFF presesnte)
;                                  (0000 ausente  )
;                       var12 ->  Número de botones
;
;


mreseteo macro Resultado,mvisible,presente1
  local finmreseteo
  push      ax
  push      bx
  mov       ax,0       ;función Cero
  mov       mvisible,ax  ;mvisible indica si el puntero está (1) 0
                         ;no (0) visible
  int       51
  mov       Resultado.var11,ax ;pasar datos de retorno a la variable
  mov       Resultado.var12,bx ;tipo estructura, Resultado
  cmp       ax,0
  je        finmreseteo
  mov       presente1,1  ;si ratón está presente entonces presente1 = 1
  Mmuestra 1,mvisible,presente1  ;llamada al macro que muestra el cursor
  finmreseteo:
  pop  bx
  pop  ax
Endm


;/* macro     : Mmuestra
;  Descripción: Muestra u Oculta el cursor del ratón invocando a las
;               funciones 1 y 2 del ratón, respectivamente. En el momento
;               que se actualiza la pantalla en el área donde se encuentra el
;               cursor del ratón, este cursor debe ser ocultado antes de
```

```
;               redibujar la pantalla y luego debe ser mostrado de nuevo. De
;               otra manera, los efectos son sorpresivos e indeseables.
;               El ocultamiento del ratón "no" afecta las operaciones de
;               seguimiento de su posición y de operación de los botones.
;  entrada:
;                  showstat    -> valores 1 o 0 (mostrar | ocultar)
;*/ Salida:     mvisible1   -> Indica si el ratón  está o no (1 o 0)
;                        visible
;Se debe tener cuidado las veces que se realiza la función de ocultar el ratón
;ya que por cada ocultamiento se debe hacer una función de mostrar. Si por
;algún motivo se realizan 4 llamadas a ocultar se deben hacer 4 llamadas a
;la función 1 para volver a mostrar.
;Por ésta razón se usa la variable Showstata que indica el estado del ratón
;con 0 se indica que está oculto y en caso de llamar a la función de ocultar,
;ésta no se realiza por que ya está oculto.

Mmuestra macro  showstat,mvisible1,presente2
  local finm,fin1m
  push      ax
  cmp       presente2,1         ;existe ratón?
  jne       fin1m               ;no
  mov       ax,showstat         ;se dese mostrar u ocultar?
  or        ax,ax
  jz        finm                ;ocultar
  mov       ax,1                ;función mostrar el puntero
  cmp       mvisible1,1         ;el puntero ya está visible?
  je        fin1m               ;si
  int       51                  ;no
  mov       mvisible1,1         ;indicar que ya está visible
  jmp       short fin1m
finm:
  mov       ax,2 ;              ;función ocultar el puntero
  cmp       mvisible1,0         ;ya estába oculto?
  je        fin1m               ;si
  int       51                  ;no
  mov       mvisible1,0         ;indicar que ya está oculto
fin1m:
  pop       ax
endm

;/* Macro     : mposicion
;  Descripción: Reporta la posición del cursor del ratón y el estado de
;               sus botones.  Las coordenadas de la posición son reportadas
;               en pixeles.  Los bits correspondientes al estado de los
;               botones, empezando por 0, son colocados en 1 si el botón
;               ha sido presionado y en 0 si está libre.
;  entrada   : Ninguno
```

```
;  Salida   : Pos:  var21 -> Indica el boton presionado
;                   var22 -> Indica la posición en x
;                   var23 -> Indica la posición en y
;ButtonStatus: el primer bit corresponde al botón izquierdo
;           (Buttonstatus = 1 se presionó botón izquierdo)
;           el segundo bit corresponde al botón derecho)
;           (Buttonstatus = 2 se presionó botón derecho)
;           el tercer bit corresponde al botón centtral
;           (Buttonstatus = 4 se presionó botón central)
;No siempres se cumple éste orden

mposicion macro Pos,presente3
  local     fin           ;variable local
  cmp       presente3,1   ;ratón presente?
  jne       fin           ;no
  push      ax            ;si
  push      bx
  push      cx
  push      dx
  mov       ax,3          ;función 3 obtener posición del puntero
  int       51 ;          ;y estado de los botones
  mov       Pos.var21,bx  ;estado de los botones
  mov       Pos.var22,cx  ;Coordenada x
  mov       Pos.var23,dx  ;Coordenada y
  pop       dx
  pop       cx
  pop       bx
  pop       ax
fin:
endm



;/* macro     : mmovera
;  Descripción: Mueve el cursor del ratón a una posición específica de la
;           pantalla.  Las coordenadas usadas son absolutas, en pixeles.
;  entrada: Mov1:
;         var11, var12 -> coordenadas x e y de la nueva
;                     posición del puntero del ratón
;Aunque las coordenadas se dan en pixels el puntero del ratón se ubica en las
;coordenadas del espacio del carácter más cercano.
;
;  Salida   : Ninguna

mmovera macro Mov1,presente4
  local fin
  cmp presente4,1     ;ratón presente?
```

```
    jne fin          ;no
    push    ax
    push    cx
    push    dx
    mov     ax,4        ;función 4, mover puntero del ratón
    mov     cx,Mov1.var11 ;
    mov     dx,Mov1.var12 ;
    int     51 ;
    pop  dx
    pop  cx
    pop  ax
fin:
endm


; /* Macro :  mpresionado
; Retorna información acerca del botón requerido:
;   Estado actual (liberado, presionado)
;   Número de veces que ha sido presionado desde la última llamada
;   a esta función
;   Posición del ratón al momento de presionar el botón
; Resetea información de conteo y posición
; Función 5 del ratón
; entrada: buttons        -> El botón del cual se requiere la información
;       (0 = izquierdo, 1 = derecho, 2 = centro)
; Salida : Pressed: var31 -> Estado de botón (Activado(1),Desactivado(0))
;               var32 -> Número de activaciones desde última llamada
;               var33,var34 -> Cordenadas x, y respectivamente al momento
;                       de activación del boton
; Pressed.var31 entrega el número de veces que se ha presionado el botón desde
; la última llamada a ésta función. Si las llamadas son muy cercanas entre
; ellas siempre se detctará una sola pulsación. Si se quiere utilizar el ratón
; con doble click se debe insertar retardo entre cada llamada para dar el
; tiempo necesario al doble click (retardos entre 300 y 600 milisegundos).

mpresionado macro buttons,Pressed,presente5
    local    fin          ;variable local para el macro
    cmp      presente5,1      ;ratón presente?
    jne      fin          ;no
    push     ax
    push     bx
    push     cx
    push     dx
    mov      ax,5          ;función 5, información de botón
    mov      bx,buttons      ;Requerimiento para el botón
    int      51          ;
    mov      Pressed.var31,ax  ;pasar resultados a la variable tipo
```

```
    mov     Pressed.var32,bx  ;estructura Pressed
    mov     Pressed.var33,cx
    mov     Pressed.var34,dx
    pop     dx
    pop     cx
    pop     bx
    pop     ax
fin:
endm


; /* macro  : mliberado
; Retorna información de liberación de botón del ratón
; Función 6 del ratón
;
; entrada: botón          -> El botón del cual se requiere la información
; Salida   : Relased: var31 -> Estado de botón (Activado(1),Desactivado(0))
;                var32 -> Número de activaciones desde última llamada
;                var33,var34 -> Cordenadas x, y respectivamente al
;                        momento de liberar el botón
;
;La diferencia de ésta función con la anterior es que entrega las coordenadas
;del puntero del ratón al momento de liberar el botón indicado
mliberado macro boton,Released,presente6
    local   fin
    cmp     presente6,1    ;ratón presente?
    jne     fin            ;no
    push    ax
    push    bx
    push    cx
    push    dx
    mov     ax,6 ;
    mov     bx,boton       ;Requerimiento para el botón
    int     51
    mov     Released.var31,ax
    mov     Released.var32,bx
    mov     Released.var33,cx ;
    mov     Released.var34,dx;
    pop     dx
    pop     cx
    pop     bx
    pop     ax
fin:
endm


; /* Macro  : Mxlimit
```

```
; Asigna rango horizontal mínimo y máximo para el puntero del ratón.
; El puntero del ratón solo se puede mover dentro del rango
; Función 7 del ratón
; entrada: Xlimit:    var11   -> cordenada x mínima de moviminto del cursor
;                     var12   -> cordenada x máxima de moviminto del cursor

Mxlimit macro  Xlimit,presente7
   local     fin
   cmp       presente7,1    ;ratón presente?
   jne       fin            ;no
   push      ax
   push      cx
   push      dx
   mov       ax,7           ;función 7, restringir área de mov. del puntero
   mov       cx,Xlimit.var11 ;
   mov       dx,Xlimit.var12 ;
   int       51
   pop       dx
   pop       cx
   pop       ax
fin:
endm

;/* Macro  : Mylimit
; Asigna rango vertical mínimo y máximo para el cursor.
; Mueve el cursor del ratón dentro del rango
; Función 8 del ratón
; entrada: Ylimit:    var11   -> cordenada y mínima de moviminto del cursor
;                             (en pixels)
;                     var12   -> cordenada y máxima de moviminto del cursor
;                             (en pixels)
; El cursor sólo puede desplazarce dentro de los límites indicados.

Mylimit macro Ylimit,presente8
   local     fin
   cmp       presente8,1    ;ratón presente?
   jne       fin            ;no
   push      ax
   push      cx
   push      dx
   mov       ax,8           ;finción 8, limites verticales del mov. del
   mov       cx,Ylimit.var11 ;puntero del ratón
   mov       dx,Ylimit.var12 ;
   int       51
   pop       dx
   pop       cx
   pop       ax
```

```
fin:
endm

;/* Macro   : formacursor
; Asigna el tipo de cursor en modo texto, 0 = software, 1 = hardware
; Cursor software:  entrada 2 y 3 son las máscaras de pantalla y cursor
; Cursor hardware:  entrada 2 y 3 especifican las líneas de inicio y fin
; Función 10 del ratón
;     entrada :   hardsoft       -> Software, hardware
;                 Cursor:     variable tipo estructura
;                       var11  -> máscara de pantalla/línea de inicio
;                       var12  -> máscara de cursor/línea de fin
;     salida  :  ninguna
;
; Para el cursor "hardware" var11 indica la línea de inicio del cur-
;sor dentro del cuadro que comprende un carácter, mientras que el segundo
;valor indica la línea  final. En este caso el cursor es rectangular. Para mo-
;nitor monocromático el rango va de 0 a 7 y para CGA de 0 a 14.
; Para el cursor "software" los bits de la máscara de pantalla hacen un AND
;lógico con el caracter existente en la pantalla, éste resultado hace un XOR
;lógico con la máscara del cursor y el resultado es el que se presentará en
;pantalla. Cada bit de la palabra de máscara de pantalla y de cursor repre-
;senta lo siguiente:
;     Bit              Descripción
;     0..7             Código de caracter Ascii extendido para la forma
;                      del cursor
;     8..10             Color de presentación(Ver atributos para color y
;                      B/N)
;     11               Intesnsidad de brillo (1=alta, 0=baja)
;     12..14            Color de fondo(ver atributos)
;     15               Parpadeo (1) sin parpadeop (0)
; Valores recomendados para la máscara del cursor son: 7F00h, FFFFh o en gene-
;ral los 8 bits menos significativos deben ser todos Uno o todos Cero. Para la
;máscara del cursor los 8 bits superiores deben ser cero, con los 8 menos sig-
;nificativos de acuerdo a la forma del cursor deseanda. Para obtener un cursor
;transparente se recomienda la máscara de pantalla 77FFh y la máscara de
;cursor 7700h
;
formacursor macro hardsoft,Cursor,presente9
   local fin
   cmp presente9,1  ;está presente el ratón?
   jne fin                  ;no
   push     ax
   push     bx
   push     cx
   push     dx
   mov      ax,10        ;función 0ah
```

```
   mov      bx,hardsoft ;
   mov      cx,cursor.var11 ;
   mov      dx,cursor.var12 ;
   int      51
   pop      dx
   pop      cx
   pop      bx
   pop      ax
fin:
endm

;/* macro : mmovimiento
; Reporta el movimiento en pasos del cursor desde la última llamada
; Función 11 del ratón
;  Entrada : Ninguna
;  Salida:  Movement: var21 -> conteo de pasos en x
;                     var22 -> conteo de pasos en y
;
;Los valores de salida indican el número de pasos que se ha movido el
;puntero en el sentido vertical y horizontal desde la última llamada a este
;macro.
;Para los ratones más antiguos cada paso representa 1/100 de pulgada
;(100 mikey/pulgada), los más actuales tienen 1/200 pulgada por paso (200
;mikey/pulgada, y hasta 320 mikey/pulgada para ratones de alta resolución.
;no es muy usada.
mmovimiento macro Movement,presente10
   local    fin          ;variable local para este macro
   cmp      presente10,1    ;ratón presente?
   jne      fin          ;no
   push     ax
   push     cx
   push     dx
   mov      ax,11        ;función 11
   int      51
   mov      Movement.var21,cx ;pasar parámetros de salida a la variable
   mov      Movement.var22,dx ;tipo estructura Movement
   pop      dx
   pop      cx
   pop      ax
fin:
endm


;/*  Macro :  mbrillante
; Funciones 13 y 14 del ratón
; entrada :  Set  -> activar(1), Desactivar(0)
; Activa/Desactiva la Emulación del funcionamiento de lápiz óptico
```

```
; (con AH=13 se activa)
;      Entrada:  Set: 0 = activar
;                     1 = desactivar
;esta función no es muy utilizada
mbrillante macro Set,presente11
   local     fin
   cmp       presente11,1
   jne       fin
   push      ax
   mov       ax,13   ; Encendido (ON)
   cmp       set,1
   je        finset
   mov       ax,14   ; Apagado (OFF)
finset:
   int       51
   pop       ax
fin:
endm

; macro  : MmoveRatio
; Movimiento pixel a pixel con radio R/8
; Función 15 del ratón
; entrada Ratio : var11 -> determina en la dirección horizontal
;                       cuántos pasos/pulgada requiere moverese
;                       fisicamente el mouse para que el puntero
;                       se desplace 8 pixels
;               var12 -> determina en la dirección vertical
;                       cuántos pasos/pulgada requiere moverese
;                       fisicamente el mouse para que el puntero
;                       se desplace 8 pixels
;
;Se recomienda para un ratón de 200Mikeys/pulgada usar para Xsize un valor de
;8 Mikey/8pixel y para Ysize 16 Mikey/8pixel, con esto el cursor cubre la pan-
;talla al rodar en un espacio de 3,2 x 2,0 pulgadas.


Mmoveratio macro Ratio,presente12
   local     fin
   cmp       presente12,1   ;ratón presente
   jne       fin            ;no
   push      ax
   pusx      cx
   push      dx
   mov       ax.15          ;función 15
   mov       cx,Ratio.var11  ;pasar parámetros de entrada a CX y DX
   mov       dx,Ratio.var12  ;
   int       51
```

```
   pop      dx
   pop      cx
   pop      ax
fin:
endm

; Macro  : mocultar
; Selecciona el área  donde el ratón permanece oculto
; Función 16 del ratón
; entrada: Conceal:  var31, var32, var33, var34
;                  -> demarcan los valores izquierdo, dercho, arriba, abajo
;                    del área de ocultamiento

mocultar macro Conceal,presente13
   local    fin
   cmp      presente13,1
   jne      fin
   push     ax
   push     cx
   push     dx
   push     si
   push     di
   mov      AX,16 ;
   mov      cx,Conceal.var31 ;
   mov      dx,Conceal.var32 ;
   mov      si,Conceal.var33 ;
   mov      di,Conceal.var34 ;
   int      51
   pop      di
   pop      si
   pop      dx
   pop      cx
   pop      ax
fin:
endm


;/* Macro  : mvelocidad
; Asigna la velocidad de desplazamiento del ratón, en mickeys/segundo
; para acelerar la respuesta al movimiento del ratón
; Función 19 del ratón
; entrada:  Speed  -> Valor umbral de velocidad desde la que se aplica
;                  la aceleración al movimiento del ratón.
;                  (Mikey/pulgada)
;Al momento que el cursor exeda la velocidad umbral la velocidad se duplica
;en algunos ratones mientras que en otros aumenta de acuerdo a una curva
;determinada.
```

```
nivelocidad macro speed,presente14

    local       fin
    cmp         presente14,1
    jne         fin
    push        ax
    push        dx
    mov         ax,19 ;
    mov         dx,speed ;
    int         51
    pop         dx
    pop         ax
fin:
endm

; ************* Fin del archivo de macros para el ratón ****************
```

```
        mov   al,0
        repne scasb
        jne   ConParam      ;si no se lo encuentra, hay parámetros
        add   cx,38         ;poner 38 espacios al final del unico mnemo
        mov   al,' '
        dec   di
        rep   stosb
        mov   al,0          ;poner un 0 al final
        stosb
        jmp   FinDesDet
ConParam:
        mov   di,bx         ;recuperar el comienzo del mnemotécnico
        mov   cx,8          ;buscar un espacio
        mov   al,' '
        repne scasb
        cmp   cx,1          ;en que posición esta el espacio?
        jb    sexta         ;si CX=0, octava posición; mover parámetros
                            ;una posición a la izquierda
        je    PonSpcFin     ;si CX=1, séptima posición, no se deben mover
        mov   cx,di         ;los parámetros
        mov   ax,es         ;ci CX>=2, sexta o menos.
        mov   ds,ax         ;en este caso, mover los parámetros a la
        sub   di,bx         ;derecha
        mov   dx,7
        sub   dx,di         ;DX=número de posiciones por mover a la derecha
        mov   di,bx
        mov   bx,cx         ;BX=comienzo de los párametros
        mov   cx,43
        mov   al,0
        repne scasb         ;buscar el fin de string
        dec   di
        mov   si,di
        add   di,dx
        std                 ;dirección inversa
CicDer: lodsb
        stosb
        cmp   si,bx         ;se llego al comienzo de los parámetros?
        jae   CicDer        ;mover el string
        mov   al,' '
CicSpc: cmp   di,si
        je    PonSpcFin
        stosb
        jmp   short CicSpc

Sexta:  mov   ax,es         ;mover los parámetros una posición a la izq.
        mov   ds,ax
        mov   si,di
```

MODULO ENSAMBLA.ASM

```
;***************PROGRAMA ENSAMBLADOR*****************

        .model  small
        .486


;_____________________________________________
;       AREA PARA DATOS
;_____________________________________________

        .data

transit db      32 dup (?)

mnemo   db      'ADd',0,'PUSh',6,'POp',7,'Or',8
        db      'ADc',10H,'SBb',18H,'ANd',20H,'DAa',27H
        db      'SUb',28H,'DAs',2FH,'XOr',30H,'AAa',37H
        db      'CMp',38H,'NEAr',39H,'FAr',3AH,'PTr',3BH
        db      'BYTe',3CH,'WORd',3DH,'DWORd',3EH,'AAs',3FH
        db      'INc',40H,'DEc',48H,'SHORt',50H,'QWORd',51H,'NOt',52H,'NEg',53H
        db      'MUl',54H,'IMUl',55H,'DIv',56H,'IDIv',57H

        db      'JNAe',72H,'JNb',73H,'Jb',72H,'JAe',73H
        db      'Jz',74H,'JNz',75H,'JNa',76H,'JNBe',77H
        db      'JPe',7AH,'JPo',7BH,'JNGe',7CH,'JNl',7DH
        db      'JNg',7EH,'JNLe',7FH,'Jo',70H,'JNo',71H
        db      'Jc',72H,'JNc',73H,'Je',74H,'JNe',75H
        db      'JBe',76H,'Ja',77H,'Js',78H,'JNs',79H
        db      'Jp',7AH,'JNp',7BH,'Jl',7CH,'JGe',7DH
        db      'JLe',7EH,'Jg',7FH,'Db',80H,'Dw',81H

        db      'TESt',84H,'XCHg',86H,'MOv',88H,'LEa',8DH
        db      'NOp',90H,'CBw',98H,'CWd',99H,'CALl',9AH,'WAIt',9BH
        db      'PUSHf',9CH,'POPf',9DH,'SAHf',9EH,'LAHf',9FH
        db      'MOVSb',0A4H,'MOVSw',0A5H,'CMPSb',0A6H,'CMPSw',0A7H
        db      'STOSb',0AAH,'STOSw',0ABH,'LODSb',0ACH,'LODSw',0ADH
        db      'SCASb',0AEH,'SCASw',0AFH,'REt',0C2H,'RETn',0C2H
        db      'LEs',0C4H,'LDs',0C5H,'RETf',0CAH,'INt',0CCH
        db      'INTo',0CEH,'IREt',0CFH,'ROl',0D0H,'ROr',0D1H

        db      'RCl',0D2H,'RCr',0D3H,'AAm',0D4H,'AAd',0D5H
```

```
        db    'XLAt',0D7H,'ESc',0D8H,'SHl',0DCH,'SHr',0DDH
        db    'SAl',0DCH,'SAr',0DFH,'LOOPNe',0E0H,'LOOPNz',0E0H
        db    'LOOPe',0E1H,'LOOPz',0E1H,'LOOp',0E2H,'JCXz',0E3H
        db    'In',0E4H,'OUt',0E6H,'JMp',0EAH,'LOCk',0F0H
        db    'REPNe',0F2H,'REPNz',0F2H,'REp',0F3H,'REPe',0F3H,'REPz',0F3H
        db    'HLt',0F4H,'CMc',0F5H,'CLc',0F8H,'STc',0F9H
        db    'CLi',0FAH,'STi',0FBH,'CLd',0FCH,'STd',0FDH
        db    0FFH

regs    db    'ALCLDLBLAHCHDHBH'
        db    'AXCXDXBXSPBPSIDI'
        db    'EAXECXEDXEBXESPEBPESIEDI'

sregs   db    'ESCSSSDSFSGS'

loners  db    27H,2FH,37H,3FH,90H,98H,99H,9BH,9CH,9DH,9EH,9FH
        db    0A4H,0A5H,0A6H,0A7H,0AAH,0ABH,0ACH,0ADH,0AEH,0AFH
        db    0CEH,0CFH,0D4H,0D5H,0D7H
        db    0F4H,0F5H,0F8H,0F9H,0FAH,0FBH,0FCH,0FDH

oners   db    6,7,40H,48H,52H,53H,54H,55H,56H,57H
        db    70H,71H,72H,73H,74H,75H,76H,77H
        db    78H,79H,7AH,7BH,7CH,7DH,7EH,7FH
        db    9AH,0C2H,0CAH,0CCH,0E0H,0E1H,0E2H,0E3H,0EAH

;_____________________________________________________
;       AREA PARA CODIGO
;_____________________________________________________
        .code

NEsp    equ   10
delspc  proc  near
;---------------------------------------------------------------
;Este procedimiento salta los espacios y los TABs en el string por
;ensamblar y deja SI apuntando al primer byte que no sea un espacio.
;
;Entrada:      DS:SI   apunta al punto del string desde el cual
;                      se desea proceder.
;Salida:       DS:SI   apunta al primer byte que no es un espacio.
;              CY      es 1 si hay más de NEsp espacios (error), en
;                      cuyo caso DS:SI no es alterado
;---------------------------------------------------------------
        push    ax
        push    cx
        push    bp
        mov     cx,NEsp             ;máximo NEsp espacios
        mov     bp,si
```

```
busc:   lodsb
        or      al,al
        jz      errdel          ;si se llega al final del string
        cmp     al,' '          ;hay un espacio?
        je      spac            ;si, saltarlo
        cmp     al,9            ;hay un TAB?
        jne     encont          ;no, hecho
spac:   loop    busc            ;saltar el espacio y ver sig. car.

errdel: mov     si,bp
        stc
        jmp     short findl
encont: dec     si              ;apuntar al carácter válido
        clc
findl:  pop     bp
        pop     cx
        pop     ax
        retn
delspc  endp

encnem  proc    near
;-----------------------------------------------------------------------
;Este procedimiento encuentra el pseudocódigo del mnemotécnico que está al
;comienzo del string por ensamblarse.
;
;Entrada:       DS:SI apunta al string, que debe terminar en 0.
;               ES:DI es la dirección en donde se pondrá el pseudocódigo.
;
;Salida:        DS:SI apunta al final del mnemotécnico + 2 en el string si
;                 este fue encontrado (pues debe terminar con espacio)
;               ES:DI apunta al final del pseudocódigo + 1
;               CY    es 1 si el mnemotécnico es inválido o si hay error
;                 como en DELSPC , en cuyo caso DS:SI y ES:DI no son
;                 alterados (excepto si había espacios desde DS:SI)
;               AX    contiene el pseudocódigo (AL=0, AH= cod.)
;-----------------------------------------------------------------------
        call    delspc          ;encontrar el mnemotécnico
        jc      finenc2         ;en caso de error, terminar
        push    bx
        push    dx
        push    es
        push    di
        mov     ax,@data
        mov     es,ax           ;hacer que ES:DI apunte a los mnemotécnicos
        mov     di,offset mnemo
        mov     dx,si           ;preservar el comienzo del mnemotécnico
```

```
        pop     es
finenc1:pop     dx
        pop     bx
finenc2:retn
encnem  endp

        public  encreg
encreg  proc    near
;---------------------------------------------------------------------
;Este procedimiento encuentra el código del registro al que apunta DS:SI.
;
;Entrada:       DS:SI apunta al ascii del registro
;
;Salida:        DS:SI es cambiado segun DELSPC
;               AH    es alterado
;               AL    código del registro
;               CY    1 si no es un registro o si hay error de DELSPC
;---------------------------------------------------------------------
        call    delspc
        jc      finencreg1      ;si hay demasiados espacios, fin
        push    cx
        push    bx
        push    es
        push    di
        mov     ax,@data
        mov     es,ax
        mov     di,offset regs  ;apuntar ES:DI a la tabla de registros
        mov     Eax,[si]        ;LO DIGITADO ascii del reg en AX
        and     Eax,0dfdfdfh    ;en mayúsculas
        mov     cx,24
srchreg:mov     Ebx,es:[di]             ;de la tabla
        cmp     cx,8
        jbe     rup0
        cmp     ax,bx
        je      foundreg
        inc     di
        inc     di
        loop    srchreg
        stc                     ;si no se lo encontró, error
        jmp     short errreg
rup0:   and     ebx,0ffffffh
        cmp     eax,ebx
        je      foundreg
        inc     di
        inc     di
        inc     di
        loop    srchreg
```

```
malsreg:stc                ;si no se lo encontró, error
        jmp    short errsreg
foundsreg:
        mov    al,[si+2]     ;tercer carácter
        and    al,0dfh       ;hacerlo mayuscúla
        cmp    al,'A'        ;si es una letra, mal
        jb     oksreg
        cmp    al,'Z'
        jbe    malsreg
oksreg: mov    al,6
        sub    al,cl         ;encontrar el código del registro
        or     al,40h
        clc
errsreg:pop    di
        pop    es
finencsreg:
        pop    cx
finencsreg1:
        retn
encsreg endp

ascahex proc   near
;-------------------------------------------------------------------
;Este procedimiento transforma el string en AX (con AL = MSD) a un byte
;en AL, CY = 1 si no hay tal hex en AX.
;-------------------------------------------------------------------
        push   cx
        cmp    ah,'f'
        ja     errhex
        cmp    ah,'a'
        jb     tstal
        and    ah,0dfh       ;transformar el carácter en AH a mayúscula
tstal:  cmp    al,'f'
        ja     errhex
        cmp    al,'a'
        jb     trns
        and    al,0dfh       ;transformar el carácter en AL a mayúscula
trns:   cmp    ah,'A'        ;ajustar el carácter en AH si es alfabético
        jb     numah
        sub    ah,'A'-'9'-1
numah:  cmp    al,'A'        ;lo mismo con AL
        jb     numal
        sub    al,'A'-'9'-1
numal:  sub    ax,'00'       ;pasar de ascii a numérico
        cmp    al,16
        jae    errhex
        cmp    ah,16
```

```
        jae     errhex
        mov     cl,4
        shl     al,cl
        or      al,ah           ;condensar el número
        clc
        jmp     short finhex
errhex: stc
finhex: pop     cx
        retn
ascahex endp

esletr? proc    near
;---------------------------------------------------------------------
;Este procedimiento determina si el carácter contenido en AL es un dígito
;o una letra, en cuyo caso CY = 0, y de lo contrario, CY = 1.
;AL es alterado.
;---------------------------------------------------------------------
        cmp     al,'0'
        jb      noeslt
        cmp     al,'9'
        jbe     sicslt
        and     al,0dfh         ;mayúscula
        cmp     al,'A'
        jb      noeslt
        cmp     al,'Z'
        jbe     sieslt
noeslt: stc
        retn
sieslt: clc
        retn
esletr? endp

        public  eshex?
eshex?  proc    near
;---------------------------------------------------------------------
;Este procedimiento determina si el string al que apunta DS:SI es un número
;hexadecimal, en cuyo caso encuentra su valor.
;
;Entrada:       DS:SI apunta al string, excluyendo signos +, - o espacios
;
;Salida:    AL  = 16 si el número es de 1 o 2 digitos, 32 en otro caso
;           AH  = longitud del string del número
;           BX  = valor del número
;           CY  = 1 si lo apuntado por DS:SI no es un número o si es un
;                 número de mas de 32 bits
;---------------------------------------------------------------------
```

```
        push    dx
        xor     bx,bx
trycero:cmp     byte ptr[si+bx],'0' ;buscar ceros iniciales
        jne     noceroin
        inc     bx              ;contarlos en BX
        jmp     short trycero
noceroin:
        mov     dh,bl           ;número de ceros en DH
        or      bx,bx
        jz      noproblem       ;si no hay ceros iniciales
        mov     ah,[si+bx]
        mov     al,'0'
        call    ascahex         ;si el string era solo ceros, ya no debe
        jnc     noproblem       ;haber mas caracteres hexadecimales válidos
        mov     ax,16
        xor     bx,bx           ;salir con cero
        jmp     finnum
noproblem:
        mov     al,[si+bx+1]
        call    esletr?
        jc      undig
        mov     al,[si+bx+2]
        call    esletr?
        jc      dosdig
        mov     al,[si+bx+3]
        call    esletr?
        jc      tresdig

        mov     al,[si+bx+4]
        call    esletr?             ;tiene más de 4 dígitos?
        jc      N4Dig               ;no, OK
        stc                         ;si, error
        jmp     short errnum
N4Dig:  mov     ax,[si+bx]          ;el número tiene 4 dígitos
        call    ascahex
        jc      errnum
        mov     dl,al
        mov     ax,[si+bx+2]
        call    ascahex
        jc      errnum
        mov     bl,al
        mov     bh,dl
        mov     ax,420h         ;AL=32, AH=4
        jmp     short finnum
tresdig:mov     al,'0'          ;número de tres dígitos
        mov     ah,[si+bx]
        call    ascahex
```

```
        jc      errnum
        mov     dl,al
        mov     ax,[si+bx+1]
        call    ascahex
        jc      errnum
        mov     bl,al
        mov     bh,dl
        mov     ax,320h         ;AL=32, AH=3
        jmp     short finnum
dosdig: mov     ax,[si+bx]              ;número de dos digitos
        call    ascahex
        jc      errnum
        mov     bl,al
        xor     bh,bh
        mov     ax,210h         ;AL=16, AH=2
        jmp     short finnum
undig:  mov     al,'0'
        mov     ah,[si+bx]
        call    ascahex
        jc      errnum
        mov     bl,al
        xor     bh,bh
        mov     ax,110h         ;AL=16, AH=1
finnum: add     ah,dh           ;incluir los ceros en la longitud
        cmp     ah,2
        jbe     longbyt
        mov     al,20h
longbyt:clc
errnum: pop     dx
        retn
eshex?  endp

esreg?  proc    near
;-----------------------------------------------------------------------
;Este procedimiento es similar al anterior: encuentra si el string al que
;apunta DS:SI es uno de los registros BX, BP, SI o DI.
;
;Entrada:       DS:SI apunta al string excluyendo signos (+ o -) y espacios
;
;Salida:    CY =1 si hay error, caso contrario:
;           AH = 2 (longitud obvia del string)
;           AL = 1 si SI
;                2 si DI
;                4 si BX
;                8 si BP
;-----------------------------------------------------------------------
        mov     ax,[si]
```

```
        shl     al,cl
        mov     dh,al           ;en dh
        call    delspc
        jc      errm1
        lodsb
        cmp     al,':'          ;hay los dos puntos reglamentarios?
        jne     errm1           ;no
        call    delspc
        jnc     corchetes
errm1:  jmp     errmem
nopref: xor     dh,dh           ;codificar inexistencia de sreg
corchetes:
        xor     dl,dl           ;inicializar primera parte del código
        xor     bx,bx           ;inicializar desplazamiento
        mov     cx,bx
        lodsb
        cmp     al,'['          ;el carácter actual debe ser el
        jne     errm1           ;corchete reglamentario
        call    delspc
        jc      errm1
        cmp     byte ptr[si],'+';inicia con un signo +?
        jne     tstmin          ;no
        inc     si              ;si, ignorarlo
        call    delspc
        jc      errm1
analiz: call    eshex?          ;hay un número hex?
        jc      noh1            ;no
        mov     cx,bx           ;si, preservar su valor
        jmp     short rdynum
noh1:   call    esreg?          ;hay un registro reglamentario?
        jnc     rdynum          ;si
errm2:  jmp     errmem          ;no, error
tstmin: cmp     byte ptr[si],'-';inicia con un signo -?
        jne     analiz          ;no
        inc     si              ;si, lo siguiente solo puede ser un número
        call    delspc
        jc      errm2
        call    eshex?
        jc      errm2
        neg     bx              ;encontrar el negativo
        mov     cx,bx           ;preservarlo
rdynum: xor     dl,al           ;poner el bit correspondiente en DL
        test    dl,al           ;estaba ya puesto, implicando duplicación?
        jz      errm2           ;si, error
        mov     al,ah           ;longitud del string
        xor     ah,ah
        add     si,ax           ;ajustar SI
```

```
        jnc     rdynum3
errm4:  jmp     errmem
tstmin3:cmp     al,'-'
        jne     errm4
        call    delspc
        jc      errm4
        call    eshex?          ;despues de un - solo puede haber un número
        jc      errm4
        neg     bx
        mov     cx,bx
rdynum3:xor     dl,al           ;poner el bit apropiado
        test    dl,al           ;duplicación?
        jz      errm4             ;si, error
        mov     al,ah           ;longitud leída hasta ahora
        xor     ah,ah
        add     si,ax           ;ajustar SI
        call    delspc
        jc      errm4

        lodsb                   ;ya se ha leído 3 cosas, debe haber
        cmp     al,']'          ;un corchete finalizador
        jne     errm4
condensar:
        mov     al,dl           ;determinar si el código
        and     al,3            ;en DL es incongruente
        cmp     al,3            ;si DL = 00FEDCBA, entonces
        je      errmem            ;error = ~A~B~C~D~E~F + AB + CD + EF
        mov     al,dl
        and     al,12
        cmp     al,12
        je      errmem
        mov     al,dl
        and     al,30h
        cmp     al,30h
        je      errmem
        mov     al,dl
        and     al,3fh
        jz      errmem

        xor     ah,ah           ;inilicializar código condensado
        test    dl,2            ;bit0 = ~AC + B
        jnz     sb0
        mov     al,dl
        xor     al,1
        and     al,5
        cmp     al,5
        jne     bit1
```

```
sb0:    or      ah,1
bit1:   test    dl,8            ;bit1 = ~A~B + D
        jnz     sb1
        mov     al,dl
        and     al,3
        jnz     bit2
sb1:    or      ah,2
bit2:   mov     al,dl           ;bit2 = ~A~B + ~C~D
        and     al,3
        jz      sb2
        mov     al,dl
        and     al,12
        jnz     bit3
sb2:    or      ah,4
bit3:   mov     al,dl           ;bit3 = ~A~B~CD~F + (A + B + C)E
        xor     al,27h
        and     al,2fh
        cmp     al,2fh
        je      sb3
        test    dl,16
        jz      bit4
        mov     al,dl
        and     al,7
        jz      bit4
sb3:    or      ah,8
bit4:   test    dl,32           ;bit4 = (A + B + C + D)F
        jz      fincond
        and     dl,15
        jz      fincond
        or      ah,16
fincond:mov     dl,ah
        mov     ax,dx
        or      al,60h
        stosw
        mov     ax,cx
        stosw
        clc
        jmp     short finmem
errmem: mov     si,bp
        stc
finmem: pop     bp
        pop     dx
        pop     cx
        pop     bx
        pop     ax
        retn
encmem  endp
```

```
        stosb
        mov     ax,bx
        stosw
        jmp     short noerrhex
unbyte: mov     ah,bl
        stosw
noerrhex:
        clc
finenchex:
        pop     dx
        pop     bx
        pop     ax
        retn
enchex  endp


enspaso1        proc    near
;-------------------------------------------------------------------------
;Este procedimiento ejecuta el primer paso de compilación, es decir, la
;traducción de un string alfanumérico a seudocódigos que serán transformados
;a código verdadero por otro procedimiento.
;
;Entrada:   DS:SI   apunta al string por compilarse, que debe terminar
;                   en 0.
;           ES:DI   apunta al lugar donde se desea el seudocódigo
;
;Salida:    DS:SI   es alterado
;           ES:DI   es alterado
;           CY      set si hay error
;-------------------------------------------------------------------------
        push    ax
        push    bx
        push    cx
        push    dx
        call    encsreg         ;es la instrucción un prefijo como ES: ?
        jc      nosregpref      ;no
        inc     si              ;si, proceder en consecuencia
        inc     si
        call    delspc
        mov     ah,al           ;preservar código
        lodsb
        cmp     al,':'          ;por sintaxis, debe haber :
        jne     errpaso1
        mov     al,ah
        stosb
        jmp     short bienpaso1
nosregpref:
```

```
        call    encnem          ;lo primero debe ser un mnemotécnico
        jc      errpaso1
        cmp     ah,0f0h         ;es un prefijo REP o LOCK?
        jb      nopreflr
        cmp     ah,0f3h
        ja      nopreflr
        call    delspc          ;si, buscar siguiente mnemotécnico
        jc      bienpaso1       ;si no hay más que el prefijo, terminar
        call    encnem
        jc      errpaso1
        cmp     ah,0f0h         ;es un segundo prefijo REP o LOCK?
        jb      nopreflr
        cmp     ah,0f3h
        ja      nopreflr
        call    delspc          ;si, buscar siguiente mnemotécnico
        jc      bienpaso1       ;si no hay mas que el segundo prefijo, terminar
        call    encnem
        jc      errpaso1
        cmp     ah,0f0h         ;no puede haber un tercer prefijo
        jb      nopreflr
        cmp     ah,0f3h
        jbe     errpaso1
nopreflr:
        cmp     ah,39h          ;el mnemotécnico inicial no puede ser NEAR,
        jb      bienmne         ;PTR, BYTE, etc
        cmp     ah,3eh
        jbe     errpaso1
        cmp     ah,50h
        je      errpaso1
        cmp     ah,51h
        je      errpaso1

bienmne:mov     dx,ds           ;preservar DS
        mov     bx,@data
        mov     ds,bx
        mov     bx,offset loners
        mov     ex,35           ;averiguar si el mnemotécnico es uno de los
lone?:  cmp     ah,[bx]         ;que no aceptan argumentos
        je      esloner         ;si
        inc     bx
        loop    lone?
        jmp     short hayarg
esloner:mov     ds,dx
        mov     byte ptr es:[di-2],1    ;marcarlo como loner
                                ;apuntar al punto inmediatamente despues
        call    delspc          ;del mnemotécnico y buscar el fin de string
        jnc     errpaso1        ;si hay mas caracteres válidos, error
```

```
bienpaso1:
        mov     al,0ffh
        stosb
        clc
        jmp     short finpaso1
errpaso1:
        stc
finpaso1:
        pop     dx
        pop     cx
        pop     bx
        pop     ax
        retn

hayarg: mov     bx,offset oners
        mov     cx,35           ;averiguar si el mnemotécnico es uno de los
one?:   cmp     ah,[bx]         ;que aceptan un solo argumento
        je      esoner          ;si
        inc     bx
        loop    one?
        jmp     short dosarg
esoner: mov     byte ptr es:[di-2],2    ;marcarlo como oner
        cmp     ah,9ah          ;CALL?
        je      admitedp        ;si, puede haber dos puntos
        cmp     ah,0eah         ;JMP?
        jne     solouno         ;no
admitedp:
        or      byte ptr es:[di-2],4    ;marcarlo como JMP o CALL
        mov     cl,2            ;marcar que puede haber un argumento doble
        jmp     short args      ;de dirección absoluta
solouno:
        mov     cl,1            ;marcar que debe haber solo un argumento
        jmp     short args
dosarg: xor     cl,cl           ;marcar que debe haber dos argumentos
args:   mov     ds,dx
        call    delspc
        jnc     enarg           ;confirmar presencia de argumentos
        cmp     ah,0c0h         ;su no presencia solo es admisible con
        je      bienpaso1       ;RET, RETN o RETF
        cmp     ah,0c2h
        je      bienpaso1
        cmp     ah,0cah
        je      bienpaso1
        jmp     short errpaso1
enarg:  call    encmem          ;buscar argumentos
        jnc     segarg
        call    encreg
```

```
        jc      noreg1
        stosb
        inc     si
        inc     si
        jmp     short segarg
noreg1: call    encsreg
        jc      nosreg1
        stosb
        inc     si
        inc     si
        jmp     short segarg
nosreg1:call    enchex
        jnc     segarg
        call    encnem          ;buscar por ejemplo BYTE PTR
        jc      errpaso1
        call    encnem          ;el PTR es opcional
        call    encmem
        jnc     segarg
        call    enchex          ;puede también ser un número (para saltos)
        jc      errpaso1
segarg: or      cl,cl           ;dos argumentos?
        jz      haydos          ;si
        call    delspc          ;no, uno solo
        jc      bien1
        cmp     cl,2            ;era JMP o CALL?
        jne     errp1           ;no
        lodsb                   ;solo se admiten dos argumentos aquí si están
        cmp     al,':'          ;separados por :
        jne     errp1
        call    enchex
        jc      errp1
        jmp     bienpaso1

haydos: call    delspc
        jc      errp1           ;si hay un solo argumento, error
        lodsb
        cmp     al,','          ;el separador entre argumentos es la coma
        jne     errp1
        call    encmem          ;buscar segundo argumento
        jnc     bien1
        call    encreg
        jc      noreg2
        stosb
        inc     si
        inc     si
bien1:  jmp     bienpaso1
noreg2: call    encsreg
```

```
        push    cx
        lodsb
        cmp     al,0ffh         ;fin de instrucción?
        je      fininst
        mov     ch,ah
        mov     ah,al
        mov     cl,5
        shr     al,cl           ;obtener el tipo
        mov     dl,al           ;en DL
        jz      esinstr
        dec     al
        jz      esregis
        dec     al
        jz      essregi
        dec     al
        jz      esmemor
        dec     al
        jz      esdatab
        lodsw                   ;es dato de word
        jmp     short rdytipo
esdatab:lodsb                   ;es dato de byte
        xor     ah,ah
        jmp     short rdytipo
esinstr:lodsb                   ;es mnemotécnico
        jmp     short rdytipo
esregis:mov     al,ah           ;es registro
        and     al,0fh
        mov     ah,ch
        jmp     short rdytipo
essregi:mov     al,ah           ;es registro de segmento
        and     al,3
        mov     ah,ch
        jmp     short rdytipo
esmemor:mov     al,ah           ;es memoria
        mov     cl,3
        shl     al,cl
        and     al,0c0h
        and     ah,7
        or      ah,al
        lodsb
        xchg    al,ah
        mov     bx,ax
        lodsw
        mov     cx,4
rdytipo:clc
        jmp     short fintipo
fininst:stc
```

```
fintipo:pop     cx
        retn
tipo    endp

getpref proc    near
;-------------------------------------------------------------------------
;Este procedimiento obtiene el prefijo de redefinición de segmento a partir
;del número de registro de segmento.
;
;Entrada:       BH      = xPSrxxxx con P = presencia de prefijo
;
;Salida:        AH      = 0 si no hay prefijo, o
;                         prefijo
;-------------------------------------------------------------------------
        xor     ah,ah
        test    bh,40h
        jz      fingtp
        mov     ah,bh
        shr     ah,1
        and     ah,18h
        or      ah,26h
fingtp: retn
getpref endp
pref?   proc    near
;-------------------------------------------------------------------------
;Este procedimiento determina si debe incluirse un prefijo de redefinición
;de segmento antes del opcode, del cual solo debe haber 1 byte, y ES:DI
;debe estar apuntando a la dirección siguiente a dicho byte. Hace la
;inclusión y los ajustes del caso de ser necesario.
;
;Entrada:       AH      0 si no debe haber prefijo
;                       opcode del prefijo en caso contrario
;               ES:DI   dirección del byte de opcode + 1
;
;Salida:        AL      es inalterado o contiene el opcode
;               DI      es ajustado
;               CX      incrementado en 1 si hay prefijo
;-------------------------------------------------------------------------
        cmp     ah,0            ;hay prefijo?
        je      noprefijo
        mov     al,ah           ;sí, ajustar
        xchg    al,es:[di-1]
        stosb
        inc     cx
noprefijo:
        retn
pref?           endp
```

```
sizedisp        proc    near
;-------------------------------------------------------------------------
;Determina dados los bits 6 y 7 de AL, que debe tener el byte de modregr/m,
;el tamano en bytes del desplazamiento u offset. Luego incluye este
;desplazamiento en el opcode.
;
;Entrada:       DL o DX desplazamiento
;               AL      byte de modo de direccionamiento
;               ES:DI   apunta al lugar del opcode donde va el desplazamiento
;
;Salida:        AX      es alterado
;               ES:DI   es ajustado
;               CX      incrementado segun el desplazamiento
;-------------------------------------------------------------------------
        test    al,0c0h         ;obtener el mod
        jz      despl?          ;si es 00, averiguar si hay desplazamiento
        and     al,0c0h
        cmp     al,0c0h         ;es 11?
        je      nodesp          ;si, tampoco hay desplazamiento
        inc     cx
        cmp     al,40h          ;es 01?
        jne     dosbyt          ;no
        mov     al,dl           ;si, desplazamiento de un byte
        stosb
        jmp     short nodesp
despl?: and     al,7            ;si mod = 00 y r/m = 110,
        cmp     al,6            ;el modo es directo
        jne     nodesp
dosbyt: mov     ax,dx           ;desplazamiento de dos bytes
        stosw
        inc     cx
nodesp: retn
sizedisp        endp

sizedata        proc    near
;-------------------------------------------------------------------------
;Similar al anterior: determina el tamaño del dato segun el bit 0 de AL, y lo
;coloca en el opcode.
;
;Entrada:       AL      su bit 0 es w
;               BP      dato
;               ES:DI   apunta al lugar del opcode donde debe ir el dato
;
;Salida:        AX      es alterado
;               DI      es ajustado
;               CX      incrementado segun los datos
;-------------------------------------------------------------------------
```

```
        inc     cx
        test    al,1            ;w = 1?
        mov     ax,bp
        jnz     dosdata         ;si
        stosb                   ;no
        jmp     short fsdata
dosdata:inc     cx
        stosw
fsdata: retn
sizedata        endp

modrm   proc    near
;-----------------------------------------------------------------------
;Este procedimiento analiza el pseudocódigo de modo de direccionamiento y
;genera datos necesarios para la compilación.
;
;Entrada:       DS:SI  debe apuntar al pseudocódigo de modo de dir.
;
;Salida:        AH     prefijo de redefinición, de haberlo, o 0
;               BL     = Md000r/m
;               BH     = h000000w; h=validez de w
;               DX     desplazamiento, de haberlo; si no, es alterado
;               CL     incrementado en 1 si no hay error
;               CH     = 0
;               CY     set si hay error, en cuyo caso los registros
;                      quedan inalterados
;-----------------------------------------------------------------------
        push    bx
        push    cx
        push    dx
        push    si
        push    bp
        push    ax
        lodsb
        dec     si
        test    al,0e0h         ;mnemo?
        jnz     mrmnomne        ;no
        call    tipo
        jc      errmodrm
        cmp     al,3ch          ;BYTE?
        jb      errmodrm
        je      bytptr
        cmp     al,3eh          ;WORD o DWORD?
        ja      errmodrm
        mov     dh,81h          ;marcar word
        jmp     short elptr
bytptr: mov     dh,80h          ;marcar byte
```

```
        pop     bp
        pop     si
        pop     dx
        pop     cx
        pop     bx
        stc
        retn
modrm   endp

modregrm        proc    near
;-----------------------------------------------------------------------
;Este procedimiento genera datos necesarios para el ensamblado de instrucciones
;que tienen dos operadores (como ADD), dados los pseudocódigos generados en el
;primer paso de compilación.
;
;Entrada:       DS:SI   debe apuntar a los pseudocódigos de los operadores.
;
;Salida:    AL      = 000000dw
;           AH      prefijo de redefinición, de haberlo, o 0
;           BL      byte de modo de direccionamiento
;           DX      desplazamiento de haberlo, si no, es alterado
;           CX      incrementado en 1 si no hay error
;           CY      set si hay error, en cuyo caso los registros quedan
;                   inalterados
;-----------------------------------------------------------------------
        push    ax
        push    cx
        push    dx
        push    si
        push    bp
        push    bx
        lodsb
        dec     si
        and     al,0e0h
        cmp     al,20h          ;reg?
        jne     mrrnoreg
        call    tipo
        call    modrm
        jc      errmrr
        call    arregla
        jc      errmrr
        or      al,2            ;d=1
        jmp     short bienmrr
mrrnoreg:
        call    modrm
        jc      errmrr
        mov     bp,dx
```

```
errarr: stc
        retn
modregrm        endp

modrmdata       proc    near
;-------------------------------------------------------------------------
;Similar al anterior, cuando el segundo operador es un dato inmediato. Produce
;resultados diferentes si el primer operador es un acumulador.
;
;Entrada:       AL      = xxCODxxx
;               DS:SI   debe apuntar a los pseudocódigos de los operadores
;
;Salida:        AL      = 000000sw
;               AH      prefijo de redefinición, de haberlo, o 0
;               BL      = md000r/m OR COD
;               DX      desplazamiento, de haberlo, si no, es alterado
;               BP      data
;               CX      incrementado en 1 si no hay error
;                       o:
;               AL      = 0000010w
;               BP      data
;               CX      inalterado
;
;               CY      set si hay error, en cuyo caso los registros quedan
;                       inalterados
;-------------------------------------------------------------------------
        push    cx
        push    dx
        push    si
        push    bp
        push    bx
        push    ax
        mov     ch,al
        lodsb
        dec     si
        mov     ah,al
        and     al,20h          ;reg?
        cmp     al,20h
        jne     noac
        test    ah,7
        jnz     noac
        lodsb
        mov     bl,al
        call    tipo
        cmp     dl,4            ;dato?
        jb      errmrd
        je      mrdok
```

```
        test    bl,8
        jnz     mrdok
        cmp     ax,255
        ja      errmrd
mrdok:  mov     bp,ax
        mov     al,bl
        and     al,8
        shr     al,1
        shr     al,1
        shr     al,1
        or      al,4
        mov     bl,al
        pop     ax
        mov     al,bl
        add     sp,6
        pop     dx
        pop     cx
        retn
noac:   mov     al,ch
        call    modrm
        jc      errmrd
        mov     ch,al
        mov     al,dl
        mov     bp,ax
        call    tipo
        cmp     dl,4            ;dato?
        jb      errmrd
        jne     nosext
        or      bh,2            ;sign extend: s=1
nosext: test    bh,80h          ;definido size?
        jz      mrdnoesp
        test    bh,1
        jnz     mrdmok
        cmp     dl,5
        jne     mrdmok
        cmp     ax,255
        ja      errmrd
        jmp     short mrdmok
mrdnoesp:
        and     bh,2
        or      bh,80h
        cmp     dl,5
        jne     mrdmok
        or      bh,1
mrdmok: xchg    bp,ax
        mov     dl,al
        and     ch,38h
```

```
        or      bl,ch
        mov     al,bh
        and     al,3
        add     sp,2
        mov     ch,bl
        pop     bx
        mov     bl,ch
        xor     ch,ch
        add     sp,8
        clc
        retn
errmrd: pop     ax
        pop     bx
        pop     bp
        pop     si
        pop     dx
        pop     cx
        stc
        retn
modrmdata       endp

delptr  proc    near
;---------------------------------------------------------------------
;El mnemotécnico PTR es innecesario en todos los casos. Si DS:SI apunta
;al pseudocódigo de este mnemotécnico, este procedimiento lo salta y deja
;a SI apuntando al siguiente pseudocódigo; en otro caso, no hace nada.
;---------------------------------------------------------------------
        push    ax
        push    bx
        push    dx
        push    bp
        mov     bp,si
        call    tipo
        jc      noptr
        or      dl,dl           ;mnemo?
        jnz     noptr
        cmp     al,3bh          ;PTR?
        je      siptr
noptr:  mov     si,bp
siptr:  pop     bp
        pop     dx
        pop     bx
        pop     ax
        retn
delptr  endp
```

```
jmpsh?  proc    near
;-------------------------------------------------------------------
;Este procedimiento determina si la distancia entre AX y DI esta entre
;+127 y -128, en cuyo caso pone esta distancia en AX y CY = 0; de lo
;contrario, CY = 1.
;-------------------------------------------------------------------
        sub     ax,di
        cmp     ax,127
        jg      notshrt
        cmp     ax,-128
        jl      notshrt
        clc
        retn
notshrt:stc
        retn
jmpsh?  endp

enspaso2        proc    near
;-------------------------------------------------------------------
;Este procedimiento ejecuta el segundo paso de compilación, es decir, la
;traducción de los pseudocódigos a un código verdadero.
;
;Entrada:       DS:SI   apunta al pseudocódigo
;               ES:DI   apunta al lugar donde se desea el código
;
;Salida:        DS:SI   es alterado
;               ES:DI   es ajustado
;               CX      longitud del código
;               CY      set si hay error, en cuyo caso los registros
;                       quedan inalterados
;-------------------------------------------------------------------
        push    ax
        push    bx
        push    dx
        push    bp
        push    si
        push    di
        push    cx
        xor     cx,cx           ;longitud inicial = 0
startp2:call    tipo            ;determinar tipo de pseudocódigo inicial
        jc      er2
        or      dl,dl           ;es una instrucción?
        jnz     prsrg?          ;no, puede ser sreg
        test    ah,1            ;es un loner?
        jz      noeslon         ;no
        cmp     al,0d4h         ;es AAM?
        je      dosb            ;si
```

```
        cmp     al,0d5h         ;es AAD?
        jne     unb             ;no
dosb:   mov     ah,0ah          ;es AAM o AAD, instrucciones cuyo opcode es
        stosw                   ;de 2 bytes
        add     cx,2            ;incrementar longitud en 2
        jmp     bienpaso2
unb:    stosb
        inc     cx
        jmp     bienpaso2
prsrg?: cmp     dl,2            ;sreg?
        jne     er2             ;no, error
        shl     al,1
        shl     al,1
        shl     al,1
        or      al,26h          ;hallar el prefijo
        stosb
        inc     cx
        jmp     bienpaso2
er2:    jmp     errp2

noeslon:cmp     al,70h          ;es un salto condicional?
        jb      nojcond
        cmp     al,7fh
        ja      nojcond
saltorel:                       ;si
        stosb
        call    tipo
        jc      er2
        cmp     dl,4            ;lo siguiente es data?
        jb      er2             ;no, error
        dec     ax
        call    jmpsh?          ;calcular distancia relativa
        jc      er2             ;y determinar si el salto es posible
        stosb
        add     cx,2
        jmp     bienpaso2

nojcond:cmp     al,0e0h         ;LOOP o JCXZ?
        jb      nolpjcx
        cmp     al,0e3h
        jbe     saltorel
nolpjcx:cmp     al,8dh          ;LEA?
        jne     diflea
regmemdisp:                     ;si
        stosb
        call    modregrm        ;hallar el código de los argumentos
        jc      er2
```

```
        cmp     al,3            ;d y w deben ser 1
        jne     er2
        call    pref?
        mov     al,bl           ;byte de modo de direccionamiento
        stosb
        call    sizedisp
        inc     cx
bp2:    jmp     bienpaso2
diflea: cmp     al,0c4h         ;LDS o LES?
        jb      noldsles
        cmp     al,0c5h
        jbe     regmemdisp
noldsles:
        cmp     al,0f0h         ;prefijo REP o LOCK?
        jb      alterables
        cmp     al,0f3h
        ja      alterables
        stosb                   ;si, incluir el prefijo
        inc     cx
        lodsb
        cmp     al,0ffh
        je      bp2
        dec     si
        jmp     startp2         ;empezar otra vez con lo que sigue al prefijo

alterables:
        test    ah,2            ;oner?
        jnz     sioner
        jmp     nooner
sioner: test    ah,4            ;JMP o CALL?
        jz      nojc
        jmp     jmpocall
nojc:   cmp     al,0c2h         ;es oner.
        jb      nori            ;si no es RET o INT
        jmp     retoint
nori:   cmp     al,7            ;PUSH o POP?
        ja      nopushpop       ;no
        mov     dh,al
        lodsb
        dec     si
        mov     bl,al
        and     bl,0e0h
        cmp     bl,20h          ;reg?
        je      ppreg           ;si
        mov     al,dh
        call    modrm           ;no, mem?
        jc      ppnomem         ;no
```

```
        cmp     bh,80h          ;si, w=1?
        je      er2a            ;no, error
        cmp     al,7            ;POP?
        mov     al,8fh
        je      espop1          ;si, dejar su código
        mov     al,0ffh         ;no, código de PUSH
espop1: stosb
        inc     cx
        call    pref?
        cmp     al,8fh          ;POP?
        je      espop2
        or      bl,30h          ;no, compensar byte de direcc.
espop2: mov     al,bl
        stosb
        call    sizedisp
        jmp     bienpaso2
er2a:   jmp     errp2

ppnomem:call    tipo
        cmp     dl,0
        je      er2a
        cmp     dl,4
        jae     er2a
        shl     al,1            ;es sreg
        shl     al,1
        shl     al,1
        or      al,7
        cmp     dh,7
        je      espop4
        and     al,0feh
espop4: stosb
        inc     cx
        jmp     bienpaso2

ppreg:  test    al,8            ;reg de 16 bits?
        jz      er2a            ;no, error
        and     al,7            ;dejar solo los tres LSBits
        or      al,58h          ;poner header del byte
        cmp     dh,7
        je      espop3
        and     al,0f7h
espop3: stosb
        inc     cx
        jmp     bienpaso2

nopushpop:
        cmp     al,48h          ;INC o DEC?
```

```
        ja      noincdec
        mov     ah,al
        lodsb
        dec     si
        and     al,0e8h
        cmp     al,28h          ;registro de 16 bits?
        je      incdecreg
        mov     al,ah
        call    modrm
        jc      er2a
        test    bh,80h          ;debe especificarse size
        jz      er2a
        or      bh,0feh
        xchg    bh,al
        stosb                   ;guardar código
        call    pref?
        cmp     bh,40h          ;INC?
        je      esinc1
        or      bl,8
esinc1: mov     al,bl
        stosb
        call    sizedisp
        inc     cx
        jmp     bienpaso2

incdecreg:
        lodsb
        and     al,7            ;aislar registro
        or      al,ah
        stosb
        inc     cx
        jmp     bienpaso2

noincdec:
        call    modrm           ;es NOT o NEG o etc.
        jc      er2a
        test    bh,80h          ;debe especificarse el size
        jz      er2b
        or      bh,0f6h
        xchg    al,bh
        stosb                   ;guardar código
        inc     cx
        mov     al,bh
        and     al,7
        shl     al,1
        shl     al,1
        shl     al,1
```

```
        or      al,bl           ;obtener byte de dir. completo
        stosb
        call    sizedisp
        jmp     bienpaso2
er2b:   jmp     err2
retoint:cmp     al,0cch         ;INT?
        jne     esret           ;no
        call    tipo            ;si
        cmp     dl,4            ;lo siguiente es un dato?
        jb      er2b
        cmp     ax,255
        ja      er2b
        inc     cx
        cmp     al,3            ;INT 3?
        je      esint3
        mov     ah,al           ;no, dos bytes
        mov     al,0cdh
        stosw
        inc     cx
        jmp     bienpaso2
esint3: mov     al,0cch         ;si, un bute
        stosb
        jmp     bienpaso2

esret:  inc     cx
        mov     bl,al           ;preservar código
        call    tipo
        jnc     conoff          ;si no hay un fin, con offset
        mov     al,bl
        inc     al              ;opcode adecuado
        stosb
bp2a:   jmp     bienpaso2
conoff: cmp     dl,4
        jb      er2b            ;error si lo siguiente no es un dato
        jne     retwo           ;si el dato es word
        mov     ah,al
        mov     al,bl
        stosw
        inc     cx
        cmp     al,0c2h         ;RETN?
        je      bp2a
        xor     al,al           ;no, un byte mas
        stosb
        inc     cx
        jmp     bienpaso2
retwo:  cmp     bl,0c2h         ;RETN?
        je      peque?
```

```
        jmp     short latalista
esfar:  mov     dh,3
latalista:
        call    delptr
        call    tipo
        jc      ej
        cmp     dl,2            ;sreg?
        je      ej
        cmp     dl,1            ;reg?
        jne     cualarg
        test    al,8            ;de 16 bits?
        jz      ej              ;no, error

cualarg:cmp     dl,4            ;datos?
        jae     jdat
        jmp     jnodat
jdat:   push    dx              ;si
        push    ax
        call    tipo
        jc      jdat8o16
        cmp     dl,4            ;dos datos?
        jae     jdat32
        pop     ax
        pop     dx
        jmp     errj
jdat32: mov     bx,ax           ;dato de 32 bits
        pop     ax
        pop     dx
        cmp     dh,1            ;short?
        je      errj
        cmp     dh,2            ;near?
        je      errj
        xchg    al,ch
        stosb                   ;solo en este caso, el código es inalterado
        xchg    ch,al
        xchg    ax,bx
        stosw
        mov     ax,bx
        stosw
        add     cl,5
        jmp     short jbien
jdat8o16:
        pop     ax
        pop     dx
        cmp     dh,3            ;far?
        je      errj
        cmp     dh,2            ;near?
```

```
        je      jnear
        cmp     dh,1
        je      jsh?
        cmp     ch,0eah         ;jmp?
        jne     jnear
        sub     ax,2
        call    jmpsh?          ;calcular distancia
        jnc     jshort
        dec     ax
        jmp     short yaresta
jnear:  sub     ax,3
        sub     ax,di
yaresta:mov     bl,0e8h
        cmp     ch,9ah          ;call?
        je      escall1
        inc     bl
escall1:xchg    bl,al
        stosb
        add     cl,3
        xchg    al,bl
        stosw
        jmp     short jbien
jsh?:   cmp     ch,0eah         ;jmp?
        jne     errj            ;solo se admite SHORT en JMP
        sub     ax,2
        sub     ax,di
        or      ah,ah
        jz      jshort
        cmp     ah,0ffh
        jne     errj
jshort: mov     ah,al
        mov     al,0ebh
        stosw
        inc     cl
        inc     cl

jbien:  xor     ch,ch
        jmp     bienpaso2
errj:   xor     ch,ch
        jmp     errp2

jnodat: cmp     dl,1            ;reg?
        jne     noreg16
        or      dh,dh           ;no debe haber restricciones de BYTE PTR, etc
        jne     errj
        mov     bl,al
        mov     al,0ffh
```

```
        stosb               ;opcode = FF
        inc     cl
        mov     al,bl
        and     al,7        ;aislar número del registro
        or      al,0c0h      ;mod = 11
        cmp     ch,0eah      ;jmp?
        je      esjmpreg
        or      al,10h
        jmp     short fjd
esjmpreg:
        or      al,20h
fjd:    stosb
        inc     cl
        jmp     short jbien
noreg16:cmp     dh,1        ;es mem, short es inadmisible
        je      errj
        sub     si,4        ;restaurar pseudocódigo
        mov     al,dh
        push    bp
        push    cx
        call    modrm
        mov     bp,sp
        mov     ch,[bp+1]
        inc     sp
        inc     sp
        pop     bp
        jc      errj
        test    bh,80h      ;especificado size?
        jz      noesps      ;no, word por default
        test    bh,1        ;word?
        jz      errj
noesps: cmp     al,3        ;far?
        jne     nojfar
        or      bl,8        ;marcar FAR
nojfar: mov     al,0ffh     ;opcode
        stosb
        inc     cl
        call    pref?
        cmp     ch,0eah      ;jmp?
        je      esjmpmem
        or      bl,10h
        jmp     short fjm
esjmpmem:
        or      bl,20h
fjm:    mov     al,bl
        stosb
        call    sizedisp
```

```
        jmp     short jbien

nooner: cmp     al,40h
        jae     mayque40h
        mov     bh,al
        call    modregrm
        jc      coninmed
        or      al,bh
        stosb
        call    pref?
        mov     al,bl
        stosb
        call    sizedisp
        inc     cx
        jmp     bienpaso2
er2h:   jmp     errp2
coninmed:
        call    modrmdata
        jc      er2h
        test    al,4            ;operación con acumulador?
        jz      sinacc          ;no
        or      al,bh
        stosb
        call    sizedata
        inc     cx
        jmp     bienpaso2
sinacc: or      al,80h
        stosb
        call    pref?
        xchg    bl,al
        stosb
        call    sizedisp
        mov     al,bl
        shr     al,1
        xor     al,1
        call    sizedata
        inc     cx
        jmp     bienpaso2
mayque40h:
        cmp     al,84h          ;TEST?
        jne     noestest
        call    modregrm
        jc      tdat?
        and     al,1
        or      al,84h          ;generar código
        stosb
        inc     cx
```

```
        call    pref?
        mov     al,bl
        stosb
        call    sizedisp
        jmp     bienpaso2
tdat?:  xor     al,al
        call    modrmdata
        jc      er2f
        test    al,4
        jnz     taccd
        or      al,0f6h
        stosb
        inc     cx
        call    pref?
        mov     bh,al
        mov     al,bl
        stosb
        call    sizedisp
        mov     al,bh
        call    sizedata
        jmp     bienpaso2
taccd:  and     al,1
        or      al,0a8h
        stosb
        inc     cx
        call    sizedata
        jmp     bienpaso2

nocstest:
        cmp     al,86h          ;XCHG?
        jne     noesxchg
        call    tipo
        jc      er2f
        or      dl,dl           ;mnemo?
        jz      er2f            ;si, error
        cmp     dl,2            ;sreg?
        je      er2f            ;si, error
        cmp     dl,4            ;datos?
        jae     er2f            ;si, error
        cmp     dl,1            ;reg?
        je      xreg
        sub     si,4            ;no
        jmp     short xmerg
xreg:   test    al,8            ;reg de 16 bits?
        jnz     xr16
        dec     si              ;no
        jmp     short xmerg
```

```
xr16:  mov    ah,al
       call   tipo
       jc     er2f
       or     dl,dl          ;el segundo operando tampoco
       jz     er2f           ;puede ser mnemo, sreg ni datos
       cmp    dl,2
       je     er2f
       cmp    dl,4
       je     er2f
       cmp    dl,1           ;reg?
       je     x2reg
       sub    si,5           ;no
       jmp    short xmerg
er2f:  jmp    errp2
x2reg: test   al,8           ;segundo reg. de 16 bits?
       jz     er2f           ;no, error por incompatibilidad
       cmp    ah,8           ;primer reg. acumulador?
       je     rgenal
       cmp    al,8           ;no, y el segundo?
       je     rgenah
       sub    si,2           ;no, ninguno es acumulador
       jmp    short xmerg
rgenah: mov   al,ah
rgenal: and   al,7
       or     al,90h
       stosb
       inc    cx
       jmp    bienpaso2
xmerg: call   modregrm
       jc     er2f
       or     al,86h
       stosb
       inc    cx
       call   pref?
       mov    al,bl
       stosb
       call   sizedisp
       jmp    bienpaso2
noesxchg:
       cmp    al,88h         ;MOV?
       je     siesmov
       jmp    noesmov

;-----Lo siguiente es el procesamiento de MOV, que es bastante complejo----
siesmov:push  si
       call   tipo
       jc     er2f
```

```
        or      dl,dl           ;mnemo?
        jnz     mnomne
        pop     si
        jmp     mesmem
mnomne: cmp     dl,1            ;reg?
        je      msireg
        jmp     mnoreg
msireg: test    al,7            ;si, acumulador?
        jnz     mnoac
        call    modrm           ;seg. oper. [disp]?
        cmp     bl,6
        jne     mnoac
        test    bh,80h          ;size especificado?
        jz      macany
        shl     bh,1
        shl     bh,1
        shl     bh,1
        mov     dl,al
        and     al,8
        xor     al,bh           ;sizes contradictorios?
        jnz     er2e            ;si, error
        mov     al,dl
macany: shr     al,1
        shr     al,1
        shr     al,1
        or      al,0a0h
        stosb
        inc     cx
        call    sizedisp
        pop     ax
        jmp     bienpaso2
mnoac:  pop     si
        call    modregrm
        jc      noc1
        jmp     mcaso1
noc1:   lodsb
        and     al,15           ;volver a obtener el reg
        mov     bl,al
        call    tipo
        jc      er2e
        cmp     dl,2            ;seg. oper. sreg?
        je      m2sreg
        cmp     dl,4            ;seg. oper. dato?
        jb      er2e
        cmp     dl,5            ;de word?
        jne     m2byte
        test    bl,8            ;si, compatible?
```

```
        jz      er2e
m2byte: mov     bp,ax       ;preservar dato
        mov     al,bl       ;caso 3
        or      al,0b0h
        stosb
        inc     cx
        mov     al,bl
        shr     al,1
        shr     al,1
        shr     al,1
        call    sizedata
        jmp     bienpaso2
m2sreg: test    bl,8        ;primer operador compatible?
        jz      er2e
        mov     bh,al
        mov     al,8ch
        stosb
        inc     cx
        mov     al,bh
        shl     al,1
        shl     al,1
        shl     al,1
        and     bl,7
        or      al,bl
        or      al,0c0h     ;generar byte de direccionamiento
        stosb
        inc     cx
        jmp     bienpaso2
er2e:   jmp     errp2
mnoreg: cmp     dl,2        ;primer oper. sreg?
        jne     mnosreg
        pop     dx          ;restaurar stack
        mov     dh,al
        call    tipo
        jc      er2e
        cmp     dl,1        ;segundo oper. reg?
        je      m2r
        cmp     dl,3        ;seg. oper. mem?
        je      m2m
        or      dl,dl       ;seg. oper mnemo?
        jnz     er2e
        cmp     dl,3dh      ;WORD o DWORD?
        jb      er2e
        cmp     dl,3eh
        ja      er2e
        call    delptr
        call    tipo
```

```
        cmp     dl,3            ;despues mem?
        jne     er2e
m2m:    mov     bp,ax
        mov     al,8eh
        stosb
        inc     cx
        call    getpref
        call    pref?
        mov     al,dh
        shl     al,1
        shl     al,1
        shl     al,1
        or      al,bl
        stosb
        mov     dx,bp
        call    sizedisp
        jmp     bienpaso2
m2r:    test    al,8
        jz      er2e
        mov     dl,al
        mov     al,8eh
        stosb
        inc     cx
        mov     al,dh
        shl     al,1
        shl     al,1
        shl     al,1
        and     dl,7
        or      al,dl
        or      al,0c0h         ;generar byte de direccionamiento
        stosb
        inc     cx
        jmp     bienpaso2
mnosreg:pop     si
        cmp     dl,3            ;mem?
        jne     er2d            ;no, no mas opciones
mesmem: push    si
        call    modrm
        jc      errm
        mov     bp,dx
        call    tipo
        cmp     dl,2            ;seg. oper. sreg?
        je      m2sr
        cmp     dl,1            ;seg. oper. reg?
        jne     mtry12
        test    al,7            ;si, es acumulador con [disp]?
        jnz     mtry12
```

```
        inc     cx
        call    pref?
        mov     al,bl
        stosb
        call    sizedisp
        jmp     bienpaso2
mtry2:  xor     al,al
        call    modrmdata
        jc      er2d
        test    al,4
        jnz     er2d            ;solo se admite una de las formas
        or      al,0c6h
        stosb
        inc     cx
        mov     bh,al
        call    pref?
        mov     al,bl
        stosb
        call    sizedisp
        mov     al,bh
        call    sizedata
        jmp     bienpaso2
;----------Fin del procesamiento de MOV-------------

noesmov:cmp     al,0d8h         ;ESC?
        jne     noesesc
        call    tipo
        cmp     dl,4            ;el primer operador debe ser un dato
        jb      er2d
        cmp     ax,3fh          ;de 6 bits
        ja      er2d
        mov     ah,al
        shr     al,1
        shr     al,1
        shr     al,1
        or      al,0d8h
        stosb
        inc     cx
        mov     al,ah
        call    modrm
        jc      er2c
        call    pref?
        and     al,7
        shl     al,1
        shl     al,1
        shl     al,1
        or      al,bl
```

```
errs:  xor     ch,ch
er2c:  jmp     errp2

esinout:cmp    al,0e6h      ;OUT?
        je     esout
        call   tipo         ;no, IN
        cmp    dl,1
        jne    er2c
        test   al,7
        jnz    errp2
        shr    al,1
        shr    al,1
        shr    al,1
        or     al,0e4h
        mov    bl,al
        call   tipo
        cmp    dl,1
        je     iconr
        cmp    dl,4
        jb     errp2
        cmp    ax,255
        ja     errp2
        mov    ah,al
        mov    al,bl
        stosw
        inc    cx
        inc    cx
        jmp    short bienpaso2
iconr:  cmp    al,10        ;DX?
        jne    errp2
        mov    al,bl
        or     al,8
        stosb
        inc    cx
        jmp    short bienpaso2
esout:  call   tipo
        cmp    dl,1
        je     oconr
        cmp    dl,4
        jb     errp2
        cmp    ax,255
        ja     errp2
        mov    ah,al
        call   tipo
        cmp    dl,1
        jne    errp2
        test   al,7
```

```
        jnz     errp2
        shr     al,1
        shr     al,1
        shr     al,1
        or      al,0e6h
        stosw
        inc     cx
        inc     cx
        jmp     short bienpaso2
oconr:  cmp     al,10           ;DX?
        jne     errp2
        call    tipo
        cmp     dl,1
        jne     errp2
        test    al,7
        jne     errp2
        shr     al,1
        shr     al,1
        shr     al,1
        or      al,0eeh
        stosb
        inc     cx
        jmp     short bienpaso2


errp2:  pop     cx
        pop     di
        pop     si
        stc
        jmp     short finp2
bienpaso2:
        add     sp,6
        clc
finp2:  pop     bp
        pop     dx
        pop     bx
        pop     ax
        retn
enspaso2        endp

        public  assem

assem   proc    near
;---------------------------------------------------------------
;Este es el ensamblador, que llama a los pasos 1 y 2 de compilación.
;
;Entrada:       DS:SI   apunta al string por compilarse
```

```
;           ES:DI  apunta al lugar donde se desea el código
;
;Salida:    DS:SI  es alterado
;           ES:DI  es alterado
;           CX     longitud del código
;           CY     set si hay error
;-------------------------------------------------------------------------
        push   es
        push   di
        push   ax
        cld
        mov    ax,ds
        mov    es,ax
        mov    di,offset transit
        call   enspaso1
        jc     error
        mov    si,offset transit
        pop    ax
        pop    di
        pop    es
        call   enspaso2
        retn
assem   endp
error:  pop    ax
        pop    di
        pop    es
        retn

        end
```

MODULO MEMORIA.ASM

```
        .model small
        .486

;_____________________________________________
;       AREA PARA CODIGO
;_____________________________________________

.code

      public  llenar
llenar   proc   near
;-------------------------------------------------------------
;Este procedimiento llena un bloque de memoria con un valor de byte
;determinado.
;Entrada:     ES:DI   apunta al inicio del bloque
;            CX      contiene el tamaño del bloque
;            AL      contiene el valor por ponerse en el bloque
;-------------------------------------------------------------
        cld
        push    cx
        push    di
        rep     stosb
        pop     di
        pop     cx
        retn
llenar   endp

      public    buscar
buscar  proc    near
;-------------------------------------------------------------------------
;Este procedimiento busca un string en un bloque de memoria.
;Entrada:  ES:DI   apunta al inicio del bloque
;            CX      máximo número de bytes que deben ser explorados
;            DS:SI   apunta al string por buscarse
;            BX      longitud del string
;
;Salida:     ES:DI   apunta al string que se encontro, o al final del bloque
;                    si no se lo encontro
;            CX      >0 si se encontro, 0 si no se encontro
;-------------------------------------------------------------------------
```

```
        cld
BuscaStr:
        push    cx
        push    si
        push    di
        mov     cx,bx
        repe    cmpsb
        pop     di
        pop     si
        pop     cx
        je      FinSrch
        inc     di
        dec     cx
        jnz     BuscaStr
FinSrch:retn
buscar  endp

        public  copiar
copiar  proc    near
;---------------------------------------------------------------------
;Este procedimiento copia un bloque de un lugar a otro en la memoria.
;Entrada:  DS:SI  apunta al inicio del bloque por copiarse
;          ES:DI  apunta al inicio del destino
;          CX     numero de bytes por copiar
;---------------------------------------------------------------------
        push    si
        push    di
        push    cx
        cmp     di,si
        je      fincopiar
        ja      invcopiar
        cld
        rep     movsb
        jmp     short fincopiar
invcopiar:add   si,cx
        add     di,cx
        dec     si
        dec     di
        std
        rep     movsb
        cld
fincopiar:pop   cx
        pop     di
        pop     si
        retn
copiar          endp
        end
```

PROCEDIMIENTO CHILD486.ASM

```
        dosseg
        .model large            ;small

;_____________________________________________
;       AREA PARA CODIGO
;_____________________________________________


        .code

;Este es un overlay que la permite a un programa bajar de nivel haciendo EXEC
;de este programa, tras haber instalado un handler de 63H que traslada el
;control al programa con la funcion 80h.

start:  mov     ax,8080h
        int     63h
          mov   ah,4ch          ;nunca llega aqui, esto es solo por si se
          int   21h             ;ejecuta este programa sin el handler de la
          end   start           ;interrupcion 21h instalado.
          end
```

MACROS INIMAC.ASM

```
;_____________________________________________________________
;          Macros usados en el archivo INICIAL.ASM
;_____________________________________________________________
;
;-------------------------------------------------------------
;Macro        Parametros      Funcion
;-------------------------------------------------------------
;actuar                      salta a una de las dir. en una tabla
;ayuda        dir,nlin,color  presenta ventana con instrucciones
;bufcla       pnclave,pbuf    encuentra la posicion de un buffer
;curs         x               pone un "cursor" en x,24
;explica      pnclave         muestra la explicacion(PNCLAVE)
;getkey                       recibe dato del teclado (BIOS)
;getnmac                      da el # del macro con nombre en NOMBUF
;getxpr                       obtiene ultimo nibble <>0 en BUFFER
;guarpantalla                 guarda una pantalla
;iluminaop                    resalta la opcion actual
;inbuff       dir             recibe string desde teclado (BIOS)
;indmac                       da una lista de los macros en el indice
;limppnt                      limpia la pantalla
;menaba                       cursor abajo en ventana de subopc.
;menarr                       cursor arriba
;mender                       cursor a la derecha
;menizq                       cursor a la izquierda
;menulinea    pnclave         muestra menu(PNCLAVE)
;menuvent                     muestra menu en forma de ventana
;mostrar      pdir,plin,pcol  muestra pregunta y espera respuesta
;movbuf       pbuf1,pbuf2     PBUF1 <-- PBUF2
;nocurs       x               pone atributo normal en x,24
;nunivel      pbuf,pniv,pnum  pone PNUM en PBUF(PNIV)
;opcion?                      espera que se elija una opcion
;pchar        car             escribe CAR (DOS)
;pmenu        pnlin,pcolor    muestra menu string apuntado por AX
;posdir       pvarx,pvary,dir convierte pos. de pantalla en direccion
;restpantalla                 restaura ultima pantalla salvada
;rutina                       pasa control al usuario y lo recibe
;salvareg                     push a todo
;statusf                      muestra fase del programa
;statusv                      muestra datos importante del usuario
;traereg                      pop a todo
;valida?                      ve si una opcion es del menu del usuario
```

```
;subrut                 subrutinas de todos los macros
;-------------------------------------------------------------------

;****** ACTUAR ***************
;Busca el AL-esimo elemento de la tabla de direcciones
;que ira despues de MTABLA para saltar a esa direccion.
actuar  macro
        local   mtabla
        push    bx
        mov     bx,offset mtabla
        call    mact
        pop     bx
        jmp     dword ptr diract
mtabla:
        endm

;****** AYUDA  ***************
;Escribe NLIN lineas separadas por ceros desde DIR en COLOR.
;Ocupa columnas 1 a 78 y las filas del centro de la pantalla.
ayuda   macro   dir,nlin,color
        push    ax
        push    cx
        mov     cl,nlin
        mov     ch,color
        mov     ax,offset dir
        call    may
        pop     cx
        pop     ax
        endm

;****** BUFCLA ***************
;Busca PBUF en TABBUF y pone su ubicacion en PNCLAVE
bufcla  macro   pnclave,pbuf
        push    si
        push    ax
        mov     si,offset pbuf
        call    mbc
        mov     al,mbcbyte
        mov     pnclave,al
        pop     ax
        pop     si
        endm

;****** CURS  ***************
;pone el atributo de (x,24) en inverso.
curs    macro   x
        push    ax
```

```
        endm

;****** INBUFF ***************
;Recibe desde el teclado el buffer DIR cuya longitud se indica en
;el primer byte del buffer. En el segundo byte se obtendra el
;numero de bytes recibidos. Solo se aceptan letras o numeros.
;Para edicion, solo Backspace. Pita si se ha llegado al limite.
inbuff macro  dir
        push   si
        mov    si,offset dir
        call   mib
        pop    si
        endm

;****** INDMAC ***************
;Llena la lista LISMAC en base al indice de macros
;en AREAMAC. Reemplaza los ceros por espacios (20H).
indmac  macro
        call   mim
        endm

;****** LMPPNT ***************
;Borra la pantalla
lmppnt  macro
        push   di
        push   cx
        push   ax
        mov    cx,2000
        mov    di,0
        mov    ax,0720h
        rep    stosw
        pop    ax
        pop    cx
        pop    di
        endm

;****** MENArr ***************
;Mueve cursor abajo en la ventana de subopciones.
menarr  macro
        call   mmab
        endm

;****** MENARR ***************
;Similar a MENABA.
menaba  macro
        call   mma
        endm
```

```
;****** MENDER ***************
;Mueve cursor de opcion actual a la derecha.
mender  macro
        call    mmd
        endm

;****** MENIZQ ***************
;Similar a MENDER.
menizq  macro
        call    mmi
        endm

;****** MENULINEA *************
;Muestra menu (PNCLAVE)
menulinea       macro   pnclave
        push    cx
        mov     cl,pnclave
        call    mml
        pop     cx
        endm

;****** MENUVENT **************
;Muestra menu(NCLAVE1) en forma de ventana.
menuvent        macro
        call    mmv
        endm

;****** MOSTRAR ***************
;Muestra pregunta apuntada por PDIR y espera respuesta. La pregunta
;debe tener esta forma: Mensaje,0,# de respuestas, lista de
;respuestas. En AL sale el numero de la respuesta elegida.
mostrar macro   pdir,plin,pcol
        push    ax
        mov     ah,plin
        mov     al,pcol
        mov     moslin,ah
        mov     moscol,al
        mov     ax,offset pdir
        call    mmos
        pop     ax
        mov     al,mnmos
        endm

;****** MOVBUF ***************
;Mueve PBUF1 a PBUF2.
movbuf  macro   pbuf2,pbuf1
        push    si
```

```
        push    di
        mov     si,offset pbuf1
        mov     di,offset pbuf2
        call    mmb
        pop     di
        pop     si
        endm

;****** NOCURS ***************
;pone el atributo de (x,24) en normal.
nocurs  macro   x
        push    ax
        mov     al,x
        mov     ah,bnnorm
        call    mcu
        pop     ax
        endm

;****** NUNIVEL ***************
;Pone PNUM en el buffer PBUF en el nivel PNIV.
nunivel macro   pbuf,pniv,pnum
        push    cx
        push    dx
        mov     cl,pniv
        mov     ch,pnum
        mov     dx,offset pbuf
        call    mnn
        pop     dx
        pop     cx
        endm

;****** OPCION? ***************
;Espera que se digite una opcion. Saca en MOPACT el numero de
;opcion. Las opciones son: Flechas,ENTER,^O,^K,^L,^P y otros.
;Ademas es aqui donde se almacenan los datos al grabar macros
;y se leen del macro en vez del teclado al correr o probarlos.
opcion? macro
        call    mop
        endm

;****** PCHAR ***************
;Usa funcion del DOS para escribir CAR en pantalla.
pchar   macro   car
        push    dx
        mov     dl,car
        call    mpch
        pop     dx
```

```
        endm

;****** PMENU  ***************
;Muestra menu apuntado por AX. Al salir MPMPRX
;da la direccion del primer byte luego del 0.
pmenu  macro  pnlin,pcolor
        push   cx
        push   dx
        mov    cl,pnlin
        mov    dh,pcolor
        call   mpm
        pop    dx
        pop    cx
        endm

;****** POSDIR  ***************
;Da la direccion PDIR correspondiente a (PVARX,PVARY)
;para acceso directo a la memoria de video.
posdir  macro  pvarx,pvary,pdir
        push   ax
        push   bx
        push   cx
        mov    cl,pvary
        mov    bl,pvarx
        call   mpd
        mov    bx,mpddir
        mov    pdir,bx
        pop    cx
        pop    bx
        pop    ax
        endm

;****** RESTPANTALLA **********
;Trae de PANBUF la ventana que fue salvada en GUARPANTALLA.
restpantalla   macro
        call   mrp
        call   funciones
        endm

;****** RUTINA  ***************
;Pone la direccion de la rutina(NCLAVE) de usuario en DIRSAL y la de
;regreso en MRAUX2 y MRAUX3. Pasa control al usuario, que debe regresar
;poniendo RETORNO al fin de su rutina para regresar a MRUTI.
rutina  macro
        mov    cl,nclave
        mov    al,ninfbuf
        mul    cl
```

```
        add   ax,offset tabinf      ;busco dir. de rutina(nclave)
        add   ax,4
        mov   si,ax
        mov   bx,[si]
        mov   al,[si+2]             ;y tipo de rutina.
        mov   tiprut,al             ;Preparo el salto.
        mov   mraux1,al             ;guardo tipo de rutina.
        mov   bp,offset dirsal
        mov   ds:[bp],bx
        mov   ds:[bp+2],cs
        mov   bx,offset mrut1       ;Preparo dir. de regreso.
        mov   mraux2,cs
        mov   mraux3,bx
        jmp   dword ptr dirsal      ;Control al usuario.
mrut1:  mov   al,tiprut             ;Regreso a FLUJO.
        cmp   al,mraux1             ;Ha cambiado TIPRUT?
        je    mrut2
        mov   contmcr,0ffh          ;si, cod. de error de macros,
        mov   x,1                   ;ademas X,Y previos pueden
        mov   y,0                   ;estar errados.
mrut2:
        endm

;****** SALVAREG **************
;Push a todo.
salvareg    macro
        push  eax
        push  ebx
        push  ecx
        push  edx
        push  esp
        push  ebp
        push  esi
        push  edi
        push  ds
        push  es
        push  ss
        endm

;****** STATUSF ***************
;Presenta 6 bytes apuntados por FASEPTR en (73,23)
;Se usa para indicar la fase actual del programa.
statusf macro
        call  msf
        endm

;****** STATUSV ***************
```

```
;Presenta 6 bytes apuntados por VALPTR en (73,24)
;Se usa para mostrar importantes datos del usuario.
statusv macro
        call    msv
        endm

;****** TRAEREG ***************
;Pop a todo.
traereg macro
        pop     ss
        pop     es
        pop     ds
        pop     edi
        pop     esi
        pop     ebp
        pop     esp
        pop     edx
        pop     ecx
        pop     ebx
        pop     eax
        endm

;****** VALIDA? ***************
;Si en OPCION? se digito una opcion distinta de las que maneja
;normalmente, ACTUAR manda el flujo aca para ver si es una opcion
;puesta por el usuario en el menu. Si es asi, AL=numero de opcion.
;Si el primer byte despues de la ultima opcion del menu es FF se
;supone que la ultima es la "opcion de escape" para todas las
;posibilidades que no consten en el menu.
valida? macro
        call    mv?
        mov     al,mv?aux
        endm

;****** SUBRUT *********
;Este es un macro que contiene todas las subrutinas llamadas por los
;otros macros de este archivo. Se hizo de esta manera para evitar el
;uso de demasiadas banderas, y que su ausencia no produzca "Phase
;errors" al momento del ensamblado.
subrut  macro

mact:
        salvareg
        dec     al              ;pongo el AL-esimo elemento de la
        mov     cl,2            ;tabla en DIRACT y CS en DIRACT+2
        mul     cl              ;para hacer el JMP al regresar.
        add     ax,bx
```

```
        mov     bp,offset diract
        push    si
        mov     si,ax
        mov     bx,cs:[si]
        pop     si
        mov     ds:[bp],bx
        mov     ds:[bp+2],cs
        traereg
        retn

may:
        push    dx
        push    bx
        mov     bx,ax
        mov     dl,23           ;centro el texto en pantalla:
        sub     dl,cl           ;linea inicial= (23-#lineas)/2 -1
        shr     dl,1
        dec     dl
        mov     maylin,dl
        dec     maylin
        mov     maycol,ch
        mov     ax,offset blanco
may1:   pmenu   maylin,maycol   ;primero una linea en blanco.
        inc     maylin
        mov     mpmprx,bx
may2:   mov     ax,mpmprx       ;Ahora el resto de lineas. MPMPRX
        pmenu   maylin,maycol   ;da la direccion del primer byte
        inc     maylin          ;despues del 0.
        dec     cl
        jnz     may2
        mov     ax,offset blanco
        pmenu   maylin,maycol   ;Linea en blanco al final.
        pop     bx
        pop     dx
        retn

mbc:
        salvareg
        mov     al,nmaxbuf      ;Busca un buffer en la tabla TABBUF
        cmp     al,0            ;y entrega el numero clave
        jne     mbc0            ;que servira para obtener toda la
        jmp     mbc4            ;informacion necesaria acerca de la
mbc0:   mov     mbcmenu,0       ;opcion representada por el buffer.
        mov     mbcbuf,si       ;Si el buffer no existe, clave=FF.
        mov     mbcpos,offset tabbuf
mbc1:   mov     mbcbyte,0
        mov     bx,mbcbuf
```

```
mbc3:  mov    ah,[bx]
       inc    bx
       mov    bp,mbcpos
       xor    ch,ch
       mov    cl,mbcbyte
       inc    mbcbyte
       mov    di,cx
       cmp    ah,ds:[bp+di]
       jnz    mbc2
       cmp    mbcbyte,8
       jnz    mbc3
       mov    al,mbcmenu
       jmp    mbc5
mbc2:  inc    mbcmenu
       mov    ah,nmaxbuf
       cmp    ah,mbcmenu
       je     mbc4
       add    mbcpos,8
       jmp    mbc1
mbc4:  mov    al,0ffh
mbc5:  mov    mbcbyte,al
       traereg
       retn

mcu:
       salvareg              ;Pone el atributo dado por AH en
       posdir 0,24,mcudir    ;la posicion (AL,24)
       mov    si,mcudir
       mov    bh,ah
       mov    cl,2
       mul    cl
       add    si,ax
       inc    si
       mov    es:[si],bh
       traereg
       retn

mex:
       salvareg              ;Busca en TABINF la direccion de la
       mov    al,ninfbuf     ;explicacion y la pone en linea 23.
       mul    cl
       add    ax,offset tabinf
       mov    bx,ax
       mov    ax,[bx]
       pmenu 24,colexp
       traereg
       retn
```

```
mgk:
        salvareg            ;Uso funcion del BIOS que espera
repetir:
        mov     ah,0bh
        int     21h
        cmp     al,0
        je      verraton
siguiente:
        mov     ah,0        ;que se pulse una tecla.
        int     16h
        jmp     short finmgk
verraton:
        call    pantalla
        cmp     bx,0ffh
        je      repetir
finmgk:
        cmp     ax,3b00h
        jne     nofl
        mov     vent,1
nofl:
        mov     getword,ax
        traereg
        ret

mgn:
        salvareg            ;Busca si el nombre en NOMBUF esta
        push    es          ;en la lista de macros y da su numero
        push    ds          ;+80h en NUMMAC si asi ocurre. Si es
        pop     es          ;nuevo, da el # del primero libre.
        mov     dh,0ffh     ;Si ya no hay espacio, da FF.
mgn1:   inc     dh
        cmp     dh,8
        je      mgn2
        mov     si,offset nombuf
        add     si,2        ;bytes 0 y 1 son de uso de otro macro.
        mov     di,offset areamac
        mov     al,8
        mul     dh
        add     di,ax
        inc     di          ;byte 0 es la longitud del macro.
        mov     cx,7
        repe    cmpsb       ;comparo.
        jne     mgn1        ;si no es igual, comparo otro.
        add     dh,80h      ;macro viejo.
        mov     nummac,dh
        jmp     mgn4
mgn2:   mov     dh,0ffh     ;Busco primero libre.
```

```
mgn21: inc    dh
       cmp    dh,8
       je     mgn3
       mov    di,offset areamac
       mov    al,8
       mul    dh
       add    di,ax
       mov    al,0
       mov    cx,8
       repe   scasb
       jne    mgn21
       mov    nummac,dh      ;macro nuevo.
       jmp    mgn4
mgn3:  mov    nummac,0ffh    ;macro nuevo sin sitio.
mgn4:  pop    es
       traereg
       retn
mgx:
       salvareg              ;se divide NIVEL/2, se suma a direccion
       mov    cl,nivel       ;inicial de BUFFER, y se obtiene el msn
       shr    cl,1           ;distinto de 0 en ese byte.
       xor    ch,ch          ;Ese es el ultimo nibble<>0 en BUFFER
       mov    si,offset buffer
       add    si,cx
       mov    al,[si]
       mov    ah,al
       shr    ah,1
       shr    ah,1
       shr    ah,1
       shr    ah,1
       cmp    ah,0
       jz     mgx1
       mov    al,ah
mgx1:  mov    xprev,al
       traereg
       retn

mgp:
       salvareg              ;paso pantalla a PANBUF.
       xor    si,si
       mov    di,offset panbuf
       mov    bx,ds
       mov    es,bx
       mov    ds,videoseg
       mov    cx,3680
       rep    movsb
       traereg
```

```
        retn

mio:
        salvareg            ;Resalta la opcion actual, dada por X.
        mov   es,videoseg
        mov   dl,collact        ;collact=video inverso
        mov   al,x
mio13:  posdir 1,1,miodir   ;trabajo directo en pantalla.
        mov   si,miodir     ;busco la X-esima palabra en la
        mov   di,si         ;linea 23 y cambio su atributo.
        push  es
        pop   ds
        mov   cl,al
mio131: lodsw
        cmp   al,20h        ;salto espacios iniciales.
        je    mio131
        sub   si,2
        dec   cl
        jz    mio16
mio14:  lodsw
        cmp   al,20h        ;salto palabras.
        jne   mio14
mio15:  lodsw
        cmp   al,20h
        je    mio15
        dec   si
        dec   si
        dec   cl
        jnz   mio14
mio16:  mov   di,si         ;resalto palabra.
        lodsw
        mov   ah,dl
        cmp   al,20h
        je    mio17
        cmp   al,0
        je    mio17
        stosw
        jmp   mio16
mio17:  traereg
        retn

mib:
        salvareg            ;obtengo string desde teclado.
        mov   cl,[si]       ;maximo # de caracteres
        mov   ch,0          ;contador
        mov   dx,si         ;puntero auxiliar
        add   si,2          ;bytes 0 y 1 de uso del macro
```

```
        posdir 1,24,mibdir    ;direccion de primer caracter
        mov    bx,mibdir
mib1:   mov    mibx,ch
        inc    mibx
        curs   mibx
mib2:   getkey
        cmp    al,08h         ;Backspace?
        jne    mib21
        jmp    mib4
mib21:  cmp    al,0dh         ;Enter?
        jne    mib22
        jmp    mib5
mib22:  cmp    al,30h         ;<"0"?
        jb     mib2
        cmp    al,7bh         ;>"z"?
        jnb    mib2
        cmp    al,3ah         ;es un numero?
        jb     mib3
        cmp    al,41h         ;"9"<al<"A"?
        jb     mib2
        cmp    al,5bh         ;es una mayuscula?
        jb     mib3
        cmp    al,61h         ;"Z"<al<"a"?
        jb     mib2
        sub    al,20h         ;es minuscula, hacer mayuscula
mib3:   nocurs mibx
        cmp    ch,cl          ;contador=maximo?
        jne    mib31
        pchar  07h            ;si, beep.
        jmp    mib1
mib31:  mov    [si],al        ;AL al buffer
        inc    si             ;apunto prox. direccion
        mov    es:[bx],al     ;AL a pantalla
        add    bx,2           ;apunto prox. direccion
        inc    ch             ;contador+1
        jmp    mib1
mib4:   nocurs mibx
        cmp    ch,0           ;buffer vacio?
        jne    mib41
        pchar  07h            ;si, beep.
        jmp    mib1
mib41:  dec    si             ;borro ultimo caracter
        mov    byte ptr[si],0 ;en el buffer
        sub    bx,2           ;y en pantalla
        mov    byte ptr es:[bx],20h
        dec    ch             ;contador-1
        jmp    mib1
```

```
mib5:  nocurs  mibx
       mov     si,dx          ;fin de rutina: byte 1
       inc     si             ;indica # de bytes recibidos.
       mov     [si],ch
       traereg
       retn

mim:
       salvareg               ;pone la lista de macros actual
       push    es             ;en LISMAC. Reemplaza los 00 con
       push    ds             ;2E (puntos).
       pop     es
       mov     si,offset areamac
       mov     di,offset lismac
       mov     dl,8
mim1:  inc     si
       mov     al,20h
       stosb
       mov     cl,7
mim2:  lodsb
       cmp     al,0
       jne     mim3
       mov     al,2eh
mim3:  stosb
       dec     cl
       jnz     mim2
       dec     dl
       jnz     mim1
       pop     es
       traereg
       retn

mmab:
       salvareg
       mov     cl,nclave1     ;Busca el # de subopciones que tiene
       mov     al,ninfbuf     ;el menu de la opcion actual dada por
       mul     cl             ;NCLAVE1. Si Y supera ese #, Y=0.
       add     ax,offset tabinf
       inc     ax
       inc     ax
       push    bx
       mov     bx,ax
       mov     si,[bx]
       pop     bx
mmab1: lodsb
       cmp     al,0
       jne     mmab1
```

```
        lodsb
        inc     al
        inc     y
        cmp     al,y
        jne     mmab2
        mov     y,0
mmab2:  traereg
        retn
mma:
        salvareg
        mov     cl,nclave1      ;Busca # de subopc. de menu(NCLAVE1)
        mov     al,ninfbuf      ;Si Y=FF, Y igual a ese #.
        mul     cl
        add     ax,offset tabinf
        inc     ax
        inc     ax
        push    bx
        mov     bx,ax
        mov     si,[bx]
        pop     bx
mma1:   lodsb
        cmp     al,0
        jne     mma1
        lodsb
        dec     y
        cmp     y,0ffh
        jne     mma2
        mov     y,al
mma2:   traereg
        retn
mmd:
        salvareg
        mov     cl,nclave       ;Busca # de opciones de menu actual.
        mov     al,ninfbuf      ;Si X supera ese valor, X=1.
        mul     cl              ;Siempre Y=0.
        add     ax,offset tabinf
        inc     ax
        inc     ax
        push    bx
        mov     bx,ax
        mov     si,[bx]
        pop     bx
mmd1:   lodsb
        cmp     al,0
        jne     mmd1
        lodsb
        inc     al
```

```
        inc     x               ;a la derecha.
        cmp     al,x
        jne     mmd2
        mov     x,1
mmd2:   mov     y,0
        traereg
        retn
mmi:
        salvareg
        mov     cl,nclave       ;Busca # de opciones de menu actual.
        mov     al,ninfbuf      ;Si X=0, X igual a ese #. Siempre Y=0.
        mul     cl
        add     ax,offset tabinf
        inc     ax
        inc     ax
        push    bx
        mov     bx,ax
        mov     si,[bx]
        pop     bx
mmi1:   lodsb
        cmp     al,0
        jne     mmi1
        lodsb
        dec     x
        cmp     x,0
        jne     mmi2
        mov     x,al
mmi2:   mov     y,0
        traereg
        retn
mml:
        salvareg
        mov     al,ninfbuf      ;Muestra el menu(CL) y resalta la
        mul     cl              ;opcion actual.
        add     ax,offset tabinf
        inc     ax
        inc     ax
        mov     bx,ax
        mov     ax,[bx]
      push      si
      push      ax
      mov       si,ax
otraletra:
      lodsb
      cmp       al,0
      jne       otraletra
      lodsb
```

```
        mov     numeroop,al
        pop     ax
        pop     si
        pmenu   1,collin
        iluminaop
          traereg
          retn

mmv:
        salvareg                ;Muestra ventana se subopc.
        mov     cl,nclave1        ;primero veo si la opcion
        mov     al,ninfbuf        ;tiene algun submenu.
        mul     cl              ;Si es asi, la rutina
        add     ax,offset tabinf    ;debe ser de tipo 3.
        add     ax,6
        mov     bx,ax
        cmp     byte ptr[bx],3
        jz      mmv3
        jmp     mmv7
mmv3:   mov     cl,nclave         ;Ahora busco la direccion
        mov     al,ninfbuf        ;del menu al que pertenece
        mul     cl              ;esta opcion en TABINF.
        add     ax,offset tabinf
        inc     ax
        inc     ax
        push    bx
        mov     bx,ax
        mov     si,[bx]           ;SI apunta al inicio del menu.
        pop     bx              ;Voy a contar los bytes hasta
        mov     cl,0            ;la opcion actual (x) para saber
        mov     dl,x            ;columna inicial de ventana.
mmv30:  lodsb                   ;Salto espacios iniciales
        inc     cl              ;pero los cuento.
        cmp     al,20h
        je      mmv30
        cmp     x,1             ;si x=1 ya tengo la columna
        jnz     mmv31
        dec     cl
        mov     mmvx1,cl
        jmp     mmv4
mmv31:  dec     dl              ;DL: cuantas opc mas debo saltar.
mmv32:  lodsb                   ;Cuento bytes hasta encontrar
        inc     cl              ;un espacio (CL contador).
        cmp     al,20h
        jne     mmv32
mmv33:  lodsb                   ;sumo espacios entre opciones
        inc     cl
```

```
        cmp   al,20h
        je    mmv33
        dec   dl              ;una opc menos que saltar
        jnz   mmv32
        dec   cl              ;Termino. Ventana empieza en
        mov   mmvx1,cl            ;columna anterior.
mmv4:   mov   mmvcar,0
        mov   cl,nclave1          ;Ahora busco direccion de menu
        mov   al,ninfbuf          ;asociado a la opcion actual
        mul   cl              ;en TABINF
        add   ax,offset tabinf
        inc   ax
        inc   ax
        push  bx
        mov   bx,ax               ;SI apunta a primer byte
        mov   si,[bx]             ;de ese menu
        pop   bx
        push  si
mmv40:  lodsb                 ;no tomo en cuenta los
        cmp   al,20h          ;espacios iniciales.
        je    mmv40
        dec   si
        mov   cx,1            ;Veo cuantas filas y columnas
mmv41:  lodsb                 ;ocupa la ventana
        cmp   al,20h              ;Los espacios no cuentan
        jz    mmv42
        cmp   al,0            ;termina si AL=0
        jne   mmv410
        mov   mmvlin,cl           ;# de subopciones del menu
        jmp   mmv5
mmv410: inc   ch              ;CH: # de car de subopcion
        cmp   ch,mmvcar           ;MMVCAR: maximo CH logrado
        jb    mmv41
        mov   mmvcar,ch
        jmp   mmv41
mmv42:  lodsb                 ;saltar espacios
        cmp   al,20h
        je    mmv42
        cmp   al,0            ;termina si AL=0
        jne   mmv43
        mov   mmvlin,cl           ;# de subopciones del menu
        jmp   mmv5
mmv43:  inc   cl              ;otra subopcion
        mov   ch,0            ;# de car = 0
        dec   si
        jmp   mmv41
mmv5:   pop   si
```

```
        mov   al,mmvlin          ;Busco la linea inicial
        inc   al                 ;de la ventana incluidas
        inc   al                 ;dos lineas de recuadro.
      mov     ah,1               ;
      add     ah,al              ;
        mov   mmvlin,ah
        mov   mmvlin1,ah         ;servira despues
        inc   mmvcar
        posdir mmvx1,mmvlin,mmvdir   ;dir. de esquina superior
        mov   di,mmvdir          ;izquierda en pantalla
        mov   es,videoseg
      mov     al,esqii           ;dibujar esa esquina
        mov   ah,colven          ;en color de ventana
        stosw
        mov   al,linsup          ;dibuja linea superior
        mov   cl,mmvcar
        xor   ch,ch
        rep   stosw
      mov     al,esqid           ;dibuja esq. sup. der.
        stosw
mmv50: DEC    mmvlin             ;proxima linea
        mov   bl,mmvlin
      cmp     bl,2               ;linea inferior si BL=22
        je    mmv6
        posdir mmvx1,mmvlin,mmvdir   ;escribir las subopciones
        mov   di,mmvdir
        mov   al,linlat          ;pero antes el recuadro.
        mov   bl,mmvcar          ;long de la linea
        stosw
mmv501: lodsb                    ;no tomo en cuenta los
        cmp   al,20h             ;espacios iniciales
        je    mmv501
        dec   si
mmv51:  lodsb                    ;leo byte del menu
        stosw                    ;lo escribo en pantalla
        dec   bl
        cmp   al,20h             ;He terminado la opcion?
        je    mmv52
        cmp   al,0               ;He terminado el menu?
        jne   mmv51
        dec   di                 ;en ese caso borro el 00H
        dec   di                 ;que escribi
        mov   al,20h             ; y pongo un " "
        stosw
mmv52:  cmp   bl,0               ;lleno el resto de la linea
        je    mmv53              ;con espacios
        mov   al,20h
```

```
mmv521: stosw
        dec     bl
        jnz     mmv521
mmv53:  mov     al,linlat           ;y termino con recuadro.
        stosw
mmv54:  lodsb                       ;salto espacios
        cmp     al,20h
        jz      mmv54
        dec     si
        jmp     mmv50
mmv6:   posdir  mmvx1,mmvlin,mmvdir     ;dibuja linea inferior
        mov     di,mmvdir
      mov       al,esqsi
        mov     ah,colven
        stosw
        mov     al,linsup
        mov     cl,mmvcar
        xor     ch,ch
        rep     stosw
      mov       al,esqsd
        stosw                       ;fin de dibujo de ventana.
        cmp     y,0                 ;existe subopcion seleccionada?
        jz      mmv7                    ;no, termina trabajo.
      mov       al,y                ;si, resaltarla cambiando
mmv63:  mov     bl,mmvlin1              ;el atributo de la linea.
      sub       bl,al
      mov       mmvlin,bL
      posdir    mmvx1,mmvlin,mmvdir
        mov     di,mmvdir
        mov     ax,3
        add     di,ax
        mov     bl,mmvcar
        mov     al,colact
mmv64:  stosb
        inc     di
        dec     bl
        jnz     mmv64
mmv7:   traereg
        retn
mmos:
        salvareg
        pmenu   moslin,moscol   ;Presenta pregunta y espera respuesta.
        mov     si,ax
mmos0:  lodsb
        cmp     al,0
        jne     mmos0
        mov     mdirop,si
```

```
mmos1:  lodsb               ;Obtengo el # de opciones.
        mov     cl,al
        mov     dl,al
        getkey              ;obtengo respuesta.
        cmp     al,'a'      ;minusculas --> mayusculas.
        jc      mmos11
        cmp     al,'z'+1
        jnc     mmos11
        sub     al,20h
mmos11: mov     bx,ax
mmos2:  mov     al,[si]     ;comparo y si no es ninguna de las
        inc     si          ;posibles, vuelve a pedir respuesta.
        sub     cl,1
        jc      mmos4
        cmp     al,0
        je      mmos3
mmos3:  cmp     al,bl
        jne     mmos2
        jmp     mmos5
mmos4:  mov     si,mdirop
        jmp     mmos1
mmos5:  sub     dl,cl
        mov     mnmos,dl    ;# de respuesta en MNMOS.
        traereg
        retn
mmb:
        salvareg            ;mueve 8 bytes.
        push    ds
        pop     es
        mov     cx,8
        rep     movsb
        traereg
        retn

mnn:
        salvareg
        test    cl,1        ;pongo CH en el CL-esimo nibble
        jz      mnn1        ;del buffer apuntado por DX.
        mov     al,16
        mul     ch
        cmp     al,0
        jnz     mnn4
        mov     ch,0fh
        mov     al,0ffh
        jmp     mnn1
mnn4:   mov     ch,al
mnn1:   shr     cl,1
```

```
        mov    bl,cl
        xor    bh,bh
        add    bx,dx
        cmp    ch,0
        jz     mnn5
        cmp    al,0ffh
        jnz    mnn2
mnn5:   and    [bx],ch
        jmp    mnn3
mnn2:   or     [bx],ch
mnn3:   traereg
        retn

mop:
        salvareg                ;maneja opciones.
        mov    cl,nclave1       ;Primero veo de que tipo es la
        mov    al,ninfbuf       ;opcion actual. (Esto es usado
        mul    cl               ;al salir de este macro).
        add    ax,offset tabinf
        add    ax,6
        mov    si,ax
        mov    al,[si]
        mov    tiprut1,al
        cmp    bmac,1           ;Corriendo/probando macro?
        je     mop1
        jmp    mop2
mop1:   mov    al,longmac       ;si.
        inc    al
        cmp    al,contmac       ;termino?
        jne    mop10
        mov    bmac,0           ;si,
        mov    faseptr,offset fnorm   ;fase normal
        statusf
        jmp    mop2
mop10:  mov    si,offset buffmac      ;estoy en macro,
        push   ax               ;leo dato de BUFFMAC
        mov    al,contmac
        xor    ah,ah
        add    ax,ax
        add    si,ax
        pop    ax
        mov    bx,[si]
        inc    contmac
        cmp    contmcr,0ffh     ;fin de ejecucion por
        je     mop12            ;flujo ambiguo?
        cmp    bver,0           ;probando macro?
        je     mop13
```

```
mop11: getkey                   ;si, espero ya sea
       cmp    al,cr             ;ENTER para seguir
       je     mop13
       cmp    al,ctrlk          ;o ^K para terminar.
       jne    mop11
mop12: mov    bmac,0            ;termino prueba,
       mov    bver,0
       mov    bx,0ff00h         ;no ejecuto ultima opcion
       mov    faseptr,offset fnorm   ;fase normal.
       statusf
mop13: mov    ax,bx             ;tomo el valor del buffer
       jmp    mop3              ;en vez del teclado.
mop2:  getkey                   ;no es macro,dato de teclado.
       cmp    bgmac,0           ;grabando macro?
       jne    mop21
       jmp    mop3
mop21: mov    bx,0ff00h
       cmp    contmcr,0ffh      ;si, fin macro por razones de
       je     mop22             ;rutina de macros de usuario?
       cmp    al,ctrlk          ;si,termino de grabar?
       je     mop22
       mov    bl,contmac        ;grabo dato en BUFFMAC
       xor    bh,bh
       add    bx,bx
       add    bx,offset buffmac
       mov    ds:[bx],ax
       inc    contmac           ;y cuento # de datos. No eje-
       cmp    contmac,maxmac    ;cuto la primera en exceso.
       je     mop23
       jmp    mop3
mop23: mov    bx,ax
       dec    contmac           ;excedi el limite.
       mov    ax,offset blanco
       pmenu  24,colexp
       mov    ax,offset escmcr
       pmenu  23,collact
mop231: getkey
       mov    ax,getword
       cmp    al,1bh            ;ESC?
       jne    mop231
mop22: mov    al,nummac         ;paso datos al area de macros.
       mov    cl,8
       mul    cl
       add    ax,offset areamac
       push   es
       push   ds
       pop    es
```

```
        mov     di,ax
        mov     al,contmac          ;long. del macro,
        mov     [di],al
        inc     di
        mov     si,offset nombmac   ;y nombre
        mov     cx,7
        rep     movsb
        mov     al,nummac           ;ahora paso BUFFMAC
        mov     cl,2*maxmac         ;al area que empieza
        mul     cl                  ;con MAC1. Cada macro
        add     ax,offset mac1      ;ocupa 2*MAXMAC bytes
        mov     di,ax               ;o MAXMAC words
        mov     si,offset buffmac
        mov     cx,maxmac
        rep     movsw
        pop     es
        mov     bgmac,0             ;termina grabacion
        mov     faseptr,offset fnorm    ;fase normal.
        statusf
        mov     ax,bx               ;ejecuto la ultima opcion
mop3:   cmp     al,0                ;es un car.especial?
        je      mop32
mop31:  cmp     al,cr               ;no, es Enter?
        jne     mop311
        mov     al,5
        jmp     mop4
mop311: cmp     al,ctrlo            ;es ^O?
        jne     mop312
        mov     al,7
        jmp     mop4
mop312: cmp     al,ctrlp            ;es ^P?
        jne     mop313
        mov     al,8
        jmp     mop4
mop313: cmp     al,ctrlk            ;es ^K?
        jne     mop314
        mov     al,9
        jmp     mop4
mop314: cmp     al,1bh              ;es ESC?
        jne     mop315
        mov     al,10
        jmp     mop4
mop315: mov     mopaux,al           ;es otro?
        mov     al,6
        jmp     mop4
mop32:  cmp     ah,fleizq           ;es <--?
        jne     mop321
```

```
        mov   al,1
        jmp   mop4
mop321: cmp   ah,fleder          ;es -->?
        jne   mop322
        mov   al,2
        jmp   mop4
mop322: cmp   ah,flearr          ;es flecha arriba?
        jne   mop323
        mov   al,3
        jmp   mop4
mop323: cmp   ah,fleaba          ;es flecha abajo?
        jne   mop324
        mov   al,4
        jmp   mop4
mop324: mov   mopaux1,ah         ;es otro.
        mov   mopaux,0
        mov   al,6
mop4:   mov   mopact,al
        cmp   bopc,1             ;tengo ventana de opciones?
        jne   mop5
        restpantalla             ;si,recupero pantalla.
mop5:   traereg
        retn

mpch:
        salvareg
        mov   ah,2       ;Uso rutina del DOS para escribir DL.
        int   21h
        traereg
        retn

mpm:
        salvareg
        push  dx         ;pone un string terminado en 0
        xor   ch,ch      ;en linea CL con atributo DH.
        mov   si,ax      ;EL string es apuntado por AX.
        mov   al,longlin
        mul   cl
        mov   es,videoseg
        mov   di,ax
        pop   dx
        mov   ah,dh
        push  di
        mov   al,20h
        mov   cx,80      ;borro la linea.
        rep   stosw
        pop   di
```

```
        inc     di
        inc     di
mpm0:   lodsb                   ;pongo el string.
        cmp     al,0
        je      mpm1
        stosw
        jmp     mpm0
mpm1:   mov     mpmprx,si       ;MPMPRX apunta a proximo string.
        traereg
        retn
mpd:
        salvareg
        mov     al,160          ;Direccion = CL*160 + BL.
        mul     cl
        xor     bh,bh
        add     ax,bx
        add     ax,bx
        mov     mpddir,ax
        traereg
        retn
mrp:
        salvareg                ;Paso PANBUF a pantalla.
        xor     di,di
        mov     es,videoseg
        mov     si,offset panbuf
        mov     cx,3680
        rep     movsb
        traereg
        retn

msf:
        salvareg
        mov     si,faseptr      ;FASEPTR tiene la direccion de un
        posdir  73,24,msfdir    ;string de 6 bytes que debe ser
        mov     di,msfdir       ;mostrado en la posicion (73,24).
        mov     cx,6
        mov     ah,collact
msf1:   lodsb
        stosw
        dec     cx
        jnz     msf1
        traereg
        retn

msv:
        salvareg
    mov     si,valptr       ;VALPTR tiene la direccion de un string
```

```
        posdir 67,24,msfdir   ;de 6 bytes en el que el usuario puede
        mov    di,msfdir      ;poner algo para aparecer en (67,24).
        mov    cx,6
        mov    ah,collact
msv1:   lodsb
        stosw
        dec    cx
        jnz    msv1
        traereg
        retn

mv?:
        salvareg                ;Busca si la eleccion esta
        mov    cl,nclave          ;en el menu del usuario.
        mov    al,ninfbuf
        mul    cl
        add    ax,offset tabinf
        inc    ax
        inc    ax
        push   bx
        mov    bx,ax
        mov    si,[bx]            ;SI apunta al menu
        pop    bx
mv?1:   lodsb
        cmp    al,0
        jne    mv?1
        lodsb                   ;SI apunta a primera opcion
        mov    cl,al              ;AL tiene el # de opciones
        mov    dl,al
        push   si                 ;si el primer byte despues
        push   ax                 ;de las opciones validas
        mov    mv?aux1,0
        xor    ah,ah              ;es FF, entonces evacuar
        add    si,ax              ;las teclas no validas
        cmp    byte ptr[si],0ffh      ;por la ultima opcion.
        jne    mv?11
        mov    mv?aux1,al             ;dejo MV?AUX1 como bandera
mv?11:  pop    ax
        pop    si
        mov    al,mopaux              ;A las minusculas
        cmp    al,'a'
        jb     mv?2
        cmp    al,'z'
        ja     mv?2
        sub    al,20h             ;las hago mayusculas.
mv?2:   mov    bl,al
mv?21:  lodsb
```

```
        cmp     al,bl           ;veo si opcion es valida
        je      mv?3
        dec     cl
        jnz     mv?21
        cmp     mv?aux1,0         ;no fue valida, veo si hay
        je      mv?22             ;rutina de escape
        mov     al,mv?aux1          ;si hay esa rutina
        mov     mv?aux,al           ;salgo con respuesta
        xor     ah,ah             ;y ZF=1
        jmp     mv?4
mv?22:  or      al,1            ;no hay esa rutina,ZF=0
        jmp     mv?4
mv?3:   dec     cl              ;opcion valida,
        sub     dl,cl           ;guardo respuesta
        mov     mv?aux,dl
        xor     ah,ah             ;y salgo con ZF=1
mv?4:   traereg
        retn

        endm
```

MACRO MPRO.ASM

```
;-------------------------------------------------------------------
;Macro  Parametros          Funcion
;-------------------------------------------------------------------
;actuar                    salta a la AL-esima dir. de una tabla
;finmacro                  termina macro y da un mensaje
;getchar                   lee un ASCII
;getdata x,y,buff,tipo     lee un bin, hex o dec
;getdata2 x,y,buff,tipo    lee un dato en hex (para mamoria relativa)
;getline buff              lee un string del teclado
;limpln num,atrib          borra una linea
;lmppnt                    borra pantalla
;print  x,y,dir,atrib      pone un texto en pantalla
;rest23                    restaura linea 23
;salva23                   salva linea 23
;usrpro                    intercambia pantallas de usuario y PRO
;sbrpro                    subrutinas de los otros macro

;****************************************
;Macro-instrucciones para macro-comandos
;****************************************
;-------------------------------------------------------------------
;Macro  Parametros          Funcion
;-------------------------------------------------------------------
;cerrmcr                   cierra archivo de macros
;creamcr dirnom            crea archivo de macros
;grabmcr                   graba y cierra archivo de macros
;getkey                    lee teclado o buffer de macros
;inimcr dirnom             abre y lee (o crea) archivo de macros
;lmpmcr                    pone 0 en las areas de macros
;rtodo                     pop todo
;stodo                     push todo
;sbrmcr                    subrutinas de los otros macros
;-------------------------------------------------------------------

;****** ACTUAR ***************
;Lee el AL-esimo elemento de la tabla TABACT y salta a la direccion
;dada por ese elemento. AL=0 para el primer elemento de la tabla.
actuar  macro
        local   tabact
        push    si
        mov     ah,0
```

```
        add     ax,ax
        mov     si,offset tabact
        add     si,ax
        mov     ax,cs:[si]
        pop     si
        jmp     ax
tabact:
        endm

;****** FINMACRO **************
;Da un mensaje de fin de macro y espera un ESC para seguir
finmacro macro
        call    rfinm
        endm

;****** GETCHAR ***************
;Lee en GCASCII un ASCII del teclado o buffer de macros. En GCCOD
;sale el siguiente codigo: 0, hay dato valido en GCASCII. 1, ENTER. 2,
;TAB. 3, SHIFT TAB. 4, ESC. 5, alguna funcion o tecla especial, cuyo
;ASCII extendido sale en GCASCII. Para que ENTER, TAB, SHIFT TAB y
;ESC sean aceptados como datos normales, se debe presionar antes F10.
getchar macro
        call    rgch
        endm

;****** GETDATA ***************
;Lee datos del teclado o buffer de macros en BUFF. En BUFF se tiene:
;# de bytes pedidos, # de datos leidos, datos, # convertido en hex
;(4 bytes). Hace eco en pantalla en (X,Y). De acuerdo al TIPO
;recibe datos en bin ('b'), hex ('h'), dec ('d'), La longitud de
;BUFF debe ser adecuada: bin, cualquiera; hex, 2 0 4 bytes; dec, 3
;o 5 bytes. Si se presiona DEL o BS se borra el ultimo dato ingresado.
;ENTER es esperado para confirmar un dato y convertirlo en hex. En
;GCCOD sale un codigo si ocurre lo siguiente: 0, hay un dato correcto al
;final de BUFF. 1, flecha arriba. 2, flecha abajo. 3, flecha derecha.
;4, flecha izquierda. 5, TAB. 6, SHIFT TAB. 7, ESC. 8, PGUP. 9 PGDN.
;Cualquier otro dato es procesado para ver si es valido incluirlo en el
;buffer. Se produce un BEEP si un dato es invalido o al llegar a los
;limites. Si al ingreso a la subrutina GDRESET es 1, buffer y variables
;son inicializadas. Si es 0, no lo son, y se puede seguir recibiendo
;el mismo dato. Los numeros binarios no son convertidos en hex.
getdata macro   x,y,buff,tipo
        push    bx
        push    cx
        push    dx
        mov     bx,offset buff
        mov     cl,x
```

```
        mov   ch,y
        mov   dl,tipo
        call  rgd
        pop   dx
        pop   cx
        pop   bx
        endm
;****** GETDATA2 ***************
;Lee datos del teclado. En BUFF se tiene:
;# de bytes pedidos, # de datos leidos, datos, # convertido en hex
;(4 bytes). Hace eco en pantalla en (X,Y). De acuerdo al TIPO
;recibe datos en bin ('b'), hex ('h'), dec ('d'), La longitud de
;BUFF debe ser adecuada: bin, cualquiera; hex, 2 0 4 bytes; dec, 3
;o 5 bytes. Si se presiona DEL o BS se borra el ultimo dato ingresado.
;ENTER es esperado para confirmar un dato y convertirlo en hex. En
;GCCOD sale un codigo si ocurre lo siguiente: 0, hay un dato correcto al
;final de BUFF. 1, flecha arriba. 2, flecha abajo. 3, flecha derecha.
;4, flecha izquierda. 5, TAB. 6, SHIFT TAB. 7, ESC. 8, PGUP. 9 PGDN.
;Cualquier otro dato es procesado para ver si es valido incluirlo en el
;buffer. Se produce un BEEP si un dato es invalido o al llegar a los
;limites. Si al ingreso a la subrutina GDRESET es 1, buffer y variables
;son inicializadas. Si es 0, no lo son, y se puede seguir recibiendo
;el mismo dato. Los numeros binarios no son convertidos en hex.
getdata2 macro  x,y,buff,tipo
        push  bx
        push  cx
        push  dx
        mov   bx,offset buff
        mov   cl,x
        mov   ch,y
        mov   dl,tipo
        jmp   rgdJ          ;sucede algo extranio si se lo hace con call
virus:  pop   dx            ;en el retorno no regresa aca (se cuelga)
        pop   cx
        pop   bx
        endm

;****** GETLINE ***************
;Lee del teclado un buffer BUFF (ASCII), que tiene este formato: Numero
;de bytes pedidos, numero de bytes realmente ingresados, texto. Para
;edicion se acepta Backspace o Delete. ENTER para terminar el texto.
;En GCCOD sale un codigo: 1, se digito ENTER (buffer valido). 2, TAB.
;3, SHIFT TAB. 4, ESC y 5, especial. Este macro usa GETCHAR para
;recibir datos. En  X, Y recibe la posicion en que debe hacer eco.
getline macro  x,y,buff
        push  si
        push  cx
```

```
        mov     cl,x
        mov     ch,y
        mov     si,offset buff
        call    rgl
        pop     cx
        pop     si
        endm

;****** LIMPLN ***************
;Borra la linea NUM y pone en ella el atributo ATRIB
limpln  macro   num,atrib
        push    ax
        mov     al,num
        mov     ah,atrib
        call    rll
        pop     ax
        endm

;****** LMPPNT ***************
;Borra la pantalla
lmppnt  macro
        push    cx
        push    di
        push    ax
        mov     cx,2000
        mov     di,0
        mov     ax,0720h
        rep     stosw
        pop     ax
        pop     di
        pop     cx
        endm

;****** PRINT ***************
;Escribe el texto apuntado por DIR en la pantalla a partir de
;(X,Y) con atributo ATRIB. El texto debe terminar con un 0.
print   macro   x,y,dir,atrib
        posdir  x,y
        push    si
        push    ax
        mov     si,offset dir
        mov     ah,atrib
        call    rprt
        pop     ax
        pop     si
        endm
```