```
;void blit(Uint8 *sprite, Uint32 ancho, Uint32 alto, Uint x, Uint y, Color color
-off)

%include 'asm/macros_globales.inc'
%include 'asm/macros_pixels.inc'

%define ptrSprite       [ebp+8]
%define anchoSprite     [ebp+12]
%define altoSprite      [ebp+16]
%define coord_x         [ebp+20]
%define coord_y         [ebp+24]
%define color_off       [ebp+28]

%define ancho_screen_bytes [ebp-4]
%define ancho_sprite_bytes [ebp-8]
%define basura_sprite [ebp-12]
%define final  [ebp-16]

extern screen_pixeles

global blit

blit:
entrada_funcion 16

completo:

    mov edi, ptrSprite                      ;edi apunta todo el tiempo a la posi
cion dentro de sprite

    ;esi <-- coord_y*(3*SCREEN_W + basura) + coord_x*3 + screen_pixeles

    calcular_pixels ebx, anchoSprite
    calcular_basura edx,ebx
    mov basura_sprite,edx
    mov ancho_sprite_bytes,ebx

    mov edx, SCREEN_W*3           ;cargamos el ancho de la pantalla en edx y lo
multiplicamos por 3
    calcular_basura ebx,edx                 ;calculo la basura en ebx, desde edx
    add edx, ebx                            ;sumo el valor de la basura a edx
    mov ancho_screen_bytes,edx

    mov esi, [screen_pixeles]      ;cargo el puntero a pantalla en esi
    calcular_pixels ecx, coord_x  ;cargamos la coor x en edx y lo multiplicamos
por 3
    add esi, ecx                  ;le addiciono el valor de la coord_x a screen_
pixeles
    mov eax, coord_y              ;cargo la coord y en eax

    mul edx                        ; (pierdo edx)
    add esi, eax                  ;eax posee la cantidad de bytes q hay q sumarl
e al puntero a screen

    mov edx, ancho_screen_bytes
    mov eax, altoSprite
    mul edx                        ;guardo en ecx la cantidad de bytes q usa el
sprite
    add eax, esi                  ;sumo el punto (0,0)
    mov final, eax

;edi apunta todo el tiempo a la posicion dentro de la pantalla
;las coordenadas (x, y) (x+p, y) (x, y+q) (x+p, y+q)
nueva_fila:
  mov ecx, anchoSprite
while:
  ;edi es el puntero al byte actual del sprite
  ;reviso q el primer byte (red) sea igual
  mov bl, [edi]
```



```
  mov al, color_off
  cmp al, bl
  jne no_cambio_color

  ;reviso q los 2 ultimos bytes (green-blue) sean iguales
  mov bx, [edi + 1]
  mov eax, color_off
  ror eax, 8                        ;realizo un desplazamiento para q los bytes gr
een-blue queden en ax
  cmp ax, bx
  jne no_cambio_color

  ;cambio el color_off por el fondo
  mov bl, [esi]     ;esi es el puntero al byte actual del screen
  mov [edi], bl

  mov bx, [esi + 1]
  mov [edi + 1], bx

no_cambio_color:
  add edi, 03h
  add esi, 03h
  loopne while

  add edi, basura_sprite
  sub esi, ancho_sprite_bytes
  add esi, ancho_screen_bytes  ; edx queda apuntando al principio de la siguient
e fila

  cmp esi, final
  je finBlit

  jmp nueva_fila

finBlit:

salida_funcion 16
```

```
global smooth

smooth:
    ret
```

```
%include "./asm/macros_globales.inc"

global constructor_iterador
global hay_proximo
global proximo
global item
global liberar_iterador

section .text

%define const_it_lista [ebp + 8]
; Iterador* constructor_iterador(Lista *lista)
constructor_iterador:
    entrada_funcion 0

    mov eax, 4

    push eax
    call malloc
    add esp, 4

    cmp eax, 0

    ; si malloc no me pudo dar memoria
    je retornar

    mov ebx, const_it_lista     ; ebx = direccion que apunta a la Lista
    mov ebx, [ebx]              ; ebx = direccion que apunta al Nodo
    mov [eax], ebx              ; En el espacio creado en memoria guardo
                                ; la direccion que apunta al nodo.

retornar:
    salida_funcion 0


%define hay_prox_pit [ebp + 8]
; bool hay_proximo(Iterador *iter)
; Recordar: La especificacion de esta funcion en el enunciado esta _mal_.
; hay_proximo() es, mas bien, hay_actual()
hay_proximo:
    entrada_funcion 0
    xor eax, eax
    mov ebx, hay_prox_pit       ; ebx = direccion que apunta al Iterador
    mov ebx, [ebx]              ; ebx = direccion que apunta al Nodo actual
    ;cmp dword [ebx + prox], 0        ; el proximo es NULL?
    cmp ebx, 0            ; el actual es NULL?

    je es_null
    ; si no lo es, retorno 1
    mov eax, 1
es_null:
    salida_funcion 0

%define prox_pit [ebp + 8]
; void proximo(Iterador *iter)
proximo:
    entrada_funcion 0

    mov eax, prox_pit       ; eax = direccion que apunta al Iterador
    mov ebx, [eax]          ; ebx = direccion que apunta al Nodo actual
    mov ebx, [ebx + prox]   ; ebx = direccion que apunta al Nodo proximo
    cmp ebx, 0

    mov [eax], ebx

    salida_funcion 0

%define item_pit [ebp + 8]
; Nodo* item(Iterador *iter)
```



```
item:
    entrada_funcion 0

    mov eax, prox_pit       ; eax = direccion que apunta al Iterador
    mov eax, [eax]          ; ebx = direccion que apunta al Nodo actual
    salida_funcion 0


%define lib_pit [ebp + 8]
; void liberar_iterador(Iterador *iter)
liberar_iterador:
    entrada_funcion 0

    mov eax, lib_pit
    push eax
    call free
    add esp, 4

    salida_funcion 0
```

```
%include "./asm/macros_globales.inc"

global constructor_lista

; inicializar_nodo es opcional. En nuestro caso, podriamos haberla usado
; en agregar_item_ordenado pero no lo hicimos.
global inicializar_nodo

global verificar_id
global agregar_item_ordenado
global borrar
global liberar_lista


section .text

; Lista* constructor_lista()
constructor_lista:

    entrada_funcion 0

    push 4
    call malloc
    add esp, 4

    cmp eax, 0

    ; si malloc no me pudo dar memoria
    je retornar

    mov dword [eax], 0

retornar:
    salida_funcion 0


%define verif_lista [ebp + 8]
%define verif_id [ebp + 12]

; bool verificar_id (Lista* la_lista, Uint32 id)
verificar_id:
    entrada_funcion 0

    mov eax, verif_lista ;aca tengo el nodo* primero

ver_seguir:

    mov ebx, [eax]          ;cargo la parte menos significativa del Id del nodo
    mov ecx, [eax+4]        ;porq ID es de 64 bits

    cmp ebx, verificar_id
    jne siguiente
    cmp ecx, 0
    jne siguiente

    mov eax, 0              ;se encontro
    salida_funcion 0

siguiente:
    mov eax, [eax+prox]
    cmp eax, 0
    jne ver_seguir
    mov eax, 1

    salida_funcion 0

%macro connect_nodos 2   ; 1 y 2 registros apuntando a nodos
    mov [%1 + prox], %2
    mov [%2 + prev], %1
```



```
%endmacro


%macro asignar_miembro 3-5 edi,esi   ; 1->2 = 3 (4 y 5 reg auxiliares)
    lea %4, [%1 + %2]
    mov %5, %3
    mov dword [%4],%5
%endmacro

%define ag_lista [ebp + 8]
%define ag_surf_pers [ebp + 12]
%define ag_surf_gen [ebp + 16]
%define ag_x [ebp + 20]
%define ag_y [ebp + 24]
%define ag_id [ebp + 28]
; void agregar_item_ordenado(Lista* la_lista, SDL_Surface* surfacePers,
; SDL_Surface* surfaceGen, Uint32 x, Uint32 y, Uint32 ID);
agregar_item_ordenado:

    entrada_funcion 0
    push 32
    call malloc                   ; creo el nodo que voy a agregar
    add esp, 4
    cmp eax, 0
    jne inicializar
    salida_funcion 0

inicializar:
    ; inicializo la estructura del nodo
    asignar_miembro eax,parte_baja_id,ag_id

    mov dword[eax + parte_alta_id], 0

    asignar_miembro eax,surf_gen,ag_surf_gen

    asignar_miembro eax,surf_pers,ag_surf_pers

    asignar_miembro eax,coord_x,ag_x

    asignar_miembro eax,coord_y,ag_y

    mov dword [eax + prox], 0

    mov dword [eax + prev], 0


; en eax esta todo el tiempo el puntero al nodo nuevo y en ebx esta el puntero a
l nodo actual
inicio:
    mov edx, ag_lista           ; cargo en edx el puntero a la lista
    mov ebx, [edx]              ; cargo en ebx el puntero al primer nodo de la l
ista
    cmp ebx, 0                  ; reviso si la lista esta vacia

    jz insertar_primer_nodo     ; si no hay ningun nodo, agregar el nuevo (eax)
al principio

    mov ecx, [ebx + coord_x]    ; guardo en ecx la coord x del primer nodo
    cmp ag_x, ecx               ; reviso si la coord x del primer nodo es menor
a la que me pasaron por parametro
    jg ag_seguir                ; mayor sin signo?

;esta guardado en edx la dir de la lista y en ebx la dir del primer nodo
caso_va_primero:
    connect_nodos eax,ebx       ; pongo el elemento en eax antes del q esta en e
bx
    jmp insertar_primer_nodo    ; guardo en la lista un puntero al nuevo nodo (e
ax)
    salida_funcion 0
```

```
ag_seguir:                          ; ebx tiene un puntero al nodo actual
    cmp dword [ebx + prox], 0       ; me fijo si hay prox
    je caso_va_al_final             ; No hay proximo

    mov edx, ebx                    ; salvo en edi el nodo actual
    mov ebx, [ebx + prox]           ; Muevo ebx al proximo elemento
    mov ecx, [ebx + coord_x]        ; Guardo en ecx la coord x del siguiente nodo
    cmp ag_x, ecx
    jg ag_seguir                    ; Si nodo_actual.x > nodo_nuevo.x sigo buscando

    connect_nodos edx,eax            ; pongo el elemento nuevo (eax) despues del nod
o actual (edx)
    connect_nodos eax,ebx            ; pongo el elemento nuevo (eax) antes del proxi
mo (ebx)
    salida_funcion 0

caso_va_al_final:                   ; ebx tiene un puntero al ultimo
    connect_nodos ebx,eax            ; pongo el elemento nuevo (eax) despues del nod
o actual (ebx)
    salida_funcion 0

insertar_primer_nodo:
    mov [edx], eax                  ; guardo en la lista el puntero al nodo nuevo
    salida_funcion 0



%macro en_rango 2-3 50 ; 1: direccion de memoria del centro del rango 2: direcci
on de memoria del valor a chequear
    mov edi, %1
    sub edi, %3 ; en edi esta la cota inferior

    mov esi, %2 ; en esi esta el valor a chequear
    cmp edi, esi
    jg %%no_esta

    add edi, %3*2 ; ahora en edi esta la cota superior
    cmp %2, edi
    jg %%no_esta

    mov edi, 0
    jmp %%esta
%%no_esta:
    mov edi, 1
%%esta:
    cmp edi, 0

%endmacro

%define b_lista [ebp + 8]
%define b_x [ebp + 12]
%define b_y [ebp + 16]
; void borrar(Lista* la_lista, Uint32 x, Uint32 y)
borrar:
    entrada_funcion 0

    mov edx, b_lista            ; cargo en edx el puntero a la lista
    mov ebx, [edx]              ; cargo en ebx el puntero al primer nodo de la l
ista
b_seguir:
; asumo q en ebx esta siempre el puntero al nodo actual y en edx el puntero a la
 lista
    cmp ebx, 0
    jne revisar_rango           ; reviso si la lista esta vacia
    salida_funcion 0

; si no esta vacia
revisar_rango:
```



```
        mov ecx, [ebx + prox]       ; guardo en ecx el nodo siguiente al actual

    en_rango b_x,[ebx + coord_x]
    jne ir_avanzar
    en_rango b_y,[ebx + coord_y]
    jne ir_avanzar

; eax - ebx - ecx y elimino ebx
eliminar_elemento:
    push dword [ebx + prev]
    push dword [ebx + prox]
    push ebx
    call free
    add esp, 4
    pop ecx
    pop eax

    cmp eax, 0
    je caso_primer_elemento
    cmp ecx, 0
    je caso_ultimo_elemento

caso_elemento_intermedio:
    connect_nodos eax,ecx
ir_avanzar:
    jmp avanzar

caso_primer_elemento:
    mov edx, b_lista            ; cargo en edx el puntero a la lista (por si lo
destruyo free)
    mov [edx], ecx              ; edx es la pos de memoria donde esta la lista
    cmp ecx, 0
    je avanzar                  ; si no hay proximo elemento sigo de largo
    mov dword [ecx + prev], 0   ; pongo en null al prev del nuevo primero
    jmp avanzar

caso_ultimo_elemento:
    mov dword [eax + prox], 0   ; pongo en null al prev del nuevo primero

avanzar:
    mov ebx, ecx
    jmp b_seguir


%define l_lista [ebp + 8]
; void liberar_lista(Lista* l)
liberar_lista:
    entrada_funcion 0

    mov edx, l_lista            ; cargo en edx el puntero a la lista
    mov ebx, [edx]              ; cargo en ebx el puntero al primer nodo de la l
ista
l_seguir:
; asumo q en ebx esta siempre el puntero al nodo actual y en edx el puntero a la
 lista
    cmp ebx, 0
    jne l_eliminar_elemento; reviso si la lista esta vacia

    mov edx, l_lista            ; cargo en edx el puntero a la lista (por si lo
perdi)
    push edx
    call free
    add esp, 4

    salida_funcion 0

; si no esta vacia

; elimino ebx
```

```
l_eliminar_elemento:
    mov esi, [ebx + prox]
    push ebx
    call free
    add esp, 4
        mov ebx, esi

    jmp l_seguir
```

```
%include "./asm/macros_globales.inc"

extern screen_pixeles

; lleva registro %1 al multiplo de 4 mayor mas cercano, usando %2 como
; registro auxiliar
%macro multiplo_de_4 2
%%chequear:
    mov %2, %1
    and %2, 0x00000003
    jz %%salir

    inc %1
    jmp %%chequear
%%salir:
%endmacro

global generarFondo

section .text

%define fondo [ebp + 8]
%define fondo_w [ebp + 12]
%define fondo_h [ebp + 16]
%define coord [ebp + 20]
; void generarFondo (Uint8 *fondo, Uint32 fondo_w, Uint32 fondo_h, Uint32 screen
AbsPos)

generarFondo:
    entrada_funcion 0

    mov eax, coord
    mov edx, eax        ; edx = coord
    add eax, SCREEN_W   ; eax = coord + SCREEN_W

    mov ebx, fondo_w    ; ebx = fondo_w

    cmp eax, ebx
    jle seguir

    mov edx, ebx
    sub edx, SCREEN_W   ; edx = "coordenada posta"

seguir:
    mov ecx, [screen_pixeles]      ; ecx es la base en la pantalla

    mov esi, edx
    shl edx, 1
    add edx, esi        ; multiplicacion por 3

    add edx, fondo      ; edx es la base en el fondo

    mov esi, ebx
    shl ebx, 1
    add ebx, esi        ; multipliacion por 3
    multiplo_de_4 ebx,esi

    xor esi, esi        ; esi es la fila actual

    push ebp
    mov ebp, SCREEN_W*SCREEN_H*3
    add ebp, ecx

recorrer_y:
    xor edi, edi        ; edi es el offset (del fondo y la pantalla)

recorrer_x:
    mov eax, [edx + edi]
    mov [ecx + edi], eax
```



```
    mov eax, [edx + edi + 1]
    mov [ecx + edi + 1], eax

    mov eax, [edx + edi + 2]
    mov [ecx + edi + 2], eax


    add edi, 3
    cmp edi, SCREEN_W*3 - 3
    jl recorrer_x           ; ver si hay que pasar por x = SCREEN_W

    add edx, ebx            ; ebx era fondo_w*3 llevado a multiplo de 4

    mov esi, SCREEN_W*3

    add ecx, esi

    cmp ecx, ebp
    jl recorrer_y       ; ver si hay que pasar por y = SCREEN_H

    pop ebp

    salida_funcion 0
```

```
%include "asm/macros_globales.inc"

extern screen_pixeles
extern colores
extern g_ver0
extern g_ver1
extern g_hor0
extern g_hor1

; color_de_fondo escribe en el valor %1 del pixel (i, j) de la pantalla
; el numero %2. %1 puede ser: 0 (R), 1 (G) o 2 (B) (debe ser inmediato). %2
; puede ser un registro o un inmediato de 8 bits. color_de_fondo utiliza
; internamente eax y edx para realizar calculos.
%macro color_de_fondo_old 2

    load_screenw_pixels

    ; uso edx porque total ya lo arruine con el mul...
    lea edx, [j + j * 2] ; edx = j + j*2 = j*3
    add edx, eax        ; edx = j*3 + i*SCREEN_W*3

    mov eax, [screen_pixeles]

    mov byte [eax + edx +%1], %2

%endmacro

%macro color_de_fondo 3
    mov byte [%1 + %2], %3
%endmacro

%macro load_screenw_pixels 0
    cmp dword res_mult, mult_invalid
    je %%calcular
    mov eax, res_mult
    jmp %%mult_salida
%%calcular:
    mov eax, SCREEN_W*3
    ; tener en cuenta que esto toca edx
    mul i                           ; eax = j*SCREEN_W*3
    mov res_mult, eax               ; cacheo el resultado de mult
%%mult_salida:
%endmacro

%define mult_invalid 0xFFFFFFFF

global generarPlasma

%define i esi
%define j edi

%define rgb [ebp + 8]
%define res_mult [ebp - 4]
generarPlasma:
    entrada_funcion 4

    mov dword res_mult, mult_invalid    ;resetear valor de mult
    xor i, i
loop_i:

    xor j, j
loop_j:

    lea ecx, [j + j*4]

    xor edx, edx
    mov dx, [g_ver0]
    add ecx, edx
```



```
    and ecx, 0x000001FF

    mov ecx, [colores + ecx*4]

    lea eax, [j + j*2]

    xor edx, edx
    mov dx, [g_ver1]
    add eax, edx

    and eax, 0x000001FF

    add ecx, [colores + eax*4]

    lea eax, [i + 2*i]

    xor edx, edx
    mov dx, [g_hor0]
    add eax, edx

    and eax, 0x000001FF

    add ecx, [colores + eax*4]

    mov eax, i
    xor edx, edx
    mov dx, [g_hor1]
    add eax, edx

    and eax, 0x000001FF

    add ecx, [colores + eax*4]

    sar ecx, 4
    add ecx, 128                        ; ecx es index
    and ecx, 0xFF

    load_screenw_pixels

    mov ebx, [screen_pixeles]
    lea edx, [ j + j * 2 ]
    add ebx, edx                          ; ebx = [screen_pixeles] + 3*i

    mov dh, [ebx + eax + 2]
    shl edx, 8
    mov dx, [ebx + eax]
    mov eax, edx

    and eax, 0x00FFFFFF
    mov ebx, rgb
    and ebx, 0x00FFFFFF                 ; me quedo con los 3 bytes menos sign.
    cmp eax, ebx
    jne ir_a_seguir
    jmp entrar_al_switch

ir_a_seguir:
    jmp seguir

    ; aca viene el switch
entrar_al_switch:

    load_screenw_pixels ;cargo en eax, el desplazamiento vertical (en bytes)
    lea edx, [j + j * 2] ; edx = j + j*2 = j*3
    add eax, edx        ; eax = j*3 + i*SCREEN_W*3
    add eax, [screen_pixeles]   ;queda todo el offset completo en eax (y se calc
ula solo la primera vez)

case_1:
    cmp cl, 64
```

```
    jae case_2                          ; fallaba por q jge es signed

    shl cl, 2                           ; cl = index << 2
    mov bl, 255
    sub bl, cl
    dec bl                              ; bl = 255 - ((index << 2) + 1)

    color_de_fondo eax,0,bl
    color_de_fondo eax,1,cl
    color_de_fondo eax,2,0

    jmp seguir

case_2:
    cmp cl, 128
    jae case_3

    shl cl, 2
    inc cl                              ; cl = (index << 2) + 1

    color_de_fondo eax,0,cl
    color_de_fondo eax,1,255
    color_de_fondo eax,2,0

    jmp seguir

case_3:
    cmp cl, 192
    jae case_4

    shl cl, 2
    mov bl, 255
    sub bl, cl
    dec bl                             ; bl = 255 - ((index << 2) + 1)

    color_de_fondo eax,0,bl
    color_de_fondo eax,1,bl
    color_de_fondo eax,2,0

    jmp seguir

case_4:
    cmp cx, 256                        ;256 no entra en 8 bits (por eso us cx)
    jae case_5

    shl cl, 2
    inc cl                             ; cl = (index << 2) + 1

    color_de_fondo eax,0,cl
    color_de_fondo eax,1,0
    color_de_fondo eax,2,0

    jmp seguir

case_5:

    color_de_fondo eax,0,0
    color_de_fondo eax,1,0
    color_de_fondo eax,2,0


seguir:

    inc j
    cmp j, SCREEN_W
    jl loop_j

    mov dword res_mult, mult_invalid    ;resetear valor de mult
```



```
    inc i
    cmp i, SCREEN_H
    jl loop_i

    add word [g_ver0], 9
    add word [g_hor0], 8

    salida_funcion 4
```

```
;void recortar(Uint8* sprite, Uint32 instancia, Uint32 ancho_instancia, Uint32 a
ncho_sprite, Uint32 alto_sprite, Uint8* res, bool orientacion);

%include 'asm/macros_globales.inc'
%include 'asm/macros_pixels.inc'

%define ptrSprite [ebp+8]
%define instancia [ebp+12]
%define ancho_instancia [ebp+16]
%define ancho_sprite [ebp+20]
%define alto_sprite [ebp+24]
%define ptrResultado [ebp+28]
%define orientacion [ebp+32]

%define ancho_sprite_bytes [ebp-4]
%define basura_instancia [ebp-8]
%define defasaje [ebp-12]
%define final [ebp-16]

global recortar

recortar:

entrada_funcion 12

    mov esi, ptrSprite
    mov edi, ptrResultado
    calcular_pixels ebx,ancho_instancia      ;ebx: ancho de la instancia sobre e
l sprite en pixeles (sin la basura)
    calcular_basura eax, ebx                 ;basura para la instancia
    mov basura_instancia,eax

    mov eax, instancia
    mul ebx                                  ;tengo en edx:eax la cant de bytes
hasta la primera instancia
    add esi, eax                             ;en esi tengo el comienzo de la ins
tancia dentro del sprite

    sub ebx, 03h                             ;cantidad de bytes para avanzar del
 primer al ultimo pixel de una fila
    mov defasaje, ebx

    calcular_pixels ecx, ancho_sprite        ;cantidad de pixeles q ocupa el spr
ite
    calcular_basura ebx,ecx                  ;basura del sprite
    add ecx, ebx
    mov ancho_sprite_bytes, ecx              ;ancho del sprite en pixeles (ecx)

    mov eax, alto_sprite                     ;cantida de filas que tiene el spri
te (eax)
    mul ecx                                  ;en eax, queda la cantidad de bytes
 q ocupa el sprite
    add eax,esi                              ;a esto le sumo el principio de la
instancia para obtener donde termina esta
    mov final, eax                           ;salvo el final de la instancia

    mov eax, -03h                            ;guardar en eax, el sentido en que
se mueve esi
    cmp dword orientacion, 0
    je seguir
    mov dword defasaje, 0                    ;si se mueve hacia la derecha:
    neg eax
seguir:
    mov ecx, ancho_instancia                 ;ecx funciona de contador, indica p
or q pixel de la fila se encuenta el bucle
    mov edx, esi                             ;guardar en edx, la pos al principi
o de la iteracion
    add esi, defasaje                        ;si hay q espejar, ir hasta el ulti
mo elemento de la fila
```



```
ciclo:
    mov bl, [esi]                            ;voy copiando los bytes de cada fil
a de la instancia y dejandolos en *res
    mov [edi], bl                            ;edi es el puntero al byte actual d
e la instancia

    mov bx, [esi + 1]
    mov [edi + 1], bx

    add edi, 03h                             ;anvanzo un pixel en el buffer dest
ino
    add esi, eax                             ;anvanzo o retrocedo un pixel en el
 sprite origen
    loopne ciclo                             ;cuando el contador se hace 0 salir
 del bucle
finalizacion:
    add edi, basura_instancia
    mov esi, edx                             ;recupero el valor al principio del
 iteracion
    add esi, ancho_sprite_bytes              ;y sumo para pasar a la siguiente f
ila
    cmp esi, final                           ;si se llego al final del sprite, t
erminar
    jne seguir

    salida_funcion 12
```