UNIVERSIDAD DE SAN CARLOS DE GUATEMALA

FACULTAD DE INGENIERIA, ESCUELA DE VACACIONES DE JUNIO

LABORATORIO ARQUITECTURA DE COMPUTADORAS Y ENSAMBLADORES 1

Proyecto 2

Sokoban en ensamblador masm x86

Carlos Javier Martínez Polanco

201709282

MANUAL TÉCNICO

1. Herramientas de desarrollo:

- Se utilizó el emulador DOSBox para emular el sistema operativo DOS.
- Se utilizó el gestor Make para compilar y ejecutar el código.
- Se utilizó código ensamblador masm x86 como entorno de desarrollo.
- Se utilizó una computadora con sistema operativo Windows 11 instalado.
- Se utilizó la interrupción DOS número 21 para el manejo de servicios del procesador.
- Se utilizó la interrupción 16 para el manejo de la detección de teclas presionadas.
- Se utilizo la interrupción 10 para el manejo del modo video.
- Se utilizó la directiva Radix 16 para utilizar sistema hexadecimal en la configuración.

2. Variables:

Control de los elementos del juego

```
NADA        equ     00
JUGADOR     equ     01
PARED       equ     02
CAJA        equ     03
OBJETIVO    equ     04
SUELO       equ     05
```

```
dim_sprite_jug    db   08, 08
data_sprite_jug   db   02, 02, 04, 04, 04, 02, 02, 02
                  db   02, 02, 04, 04, 04, 04, 02, 02
                  db   02, 02, 42, 42, 42, 00, 02, 02
                  db   02, 02, 42, 42, 42, 42, 42, 02
                  db   02, 04, 20, 04, 04, 20, 04, 02
                  db   02, 04, 0e, 20, 20, 0e, 04, 02
                  db   02, 42, 20, 02, 02, 20, 42, 02
                  db   02, 06, 06, 02, 02, 06, 06, 02
```

Sprite para mostrar al jugador (MARIO BRO'S)

```
dim_sprite_suelo  db   08, 08
data_sprite_suelo db   02, 02, 02, 02, 02, 02, 02, 02
                  db   02, 02, 02, 02, 02, 02, 02, 02
                  db   02, 02, 02, 02, 02, 02, 02, 02
                  db   02, 02, 02, 02, 02, 02, 02, 02
                  db   02, 02, 02, 02, 02, 02, 02, 02
                  db   02, 02, 02, 02, 02, 02, 02, 02
                  db   02, 02, 02, 02, 02, 02, 02, 02
                  db   02, 02, 02, 02, 02, 02, 02, 02
```

Sprite para pintar el suelo (pasto verde)

```
dim_sprite_pared  db   08, 08
data_sprite_pared db   35, 35, 0f, 0f, 35, 35, 0f, 0f
                  db   35, 0f, 0f, 35, 35, 0f, 0f, 35
                  db   0f, 0f, 35, 35, 0f, 0f, 35, 35
                  db   0f, 35, 35, 0f, 0f, 35, 35, 0f
                  db   35, 35, 0f, 0f, 35, 35, 0f, 0f
                  db   35, 0f, 0f, 35, 35, 0f, 0f, 35
                  db   0f, 0f, 35, 35, 0f, 0f, 35, 35
                  db   0f, 35, 35, 0f, 0f, 35, 35, 0f
```

Sprite para pintar la pared delimitadora para que se mueva el jugador (MAR)

```
dim_sprite_obj   db   08, 08
data_sprite_obj   db  02,02,00,0e,02,02,02,02
                  db  02,02,00,0e,0e,02,02,02
                  db  02,02,00,0e,0e,0e,02,02
                  db  02,02,00,0e,0e,0e,0e,02
                  db  02,02,00,0e,0e,0e,0e,0e
                  db  02,02,00,02,02,02,02,02
                  db  02,02,00,02,02,02,02,02
                  db  02,02,00,02,02,02,02,02
```

Sprite para pintar el objetivo (BANDERA) a donde se llevaran los objetos que el jugador debe arrastrar (koopa tropas)

```
dim_sprite_caja   db   08, 08
data_sprite_caja  db  02,02,02,06,06,02,02,02
                  db  02,06,06,06,06,06,06,02
                  db  06,06,00,42,42,00,06,06
                  db  06,06,06,42,42,06,06,06
                  db  06,06,06,06,06,06,06,06
                  db  02,06,42,42,42,42,06,02
                  db  02,00,00,02,02,00,00,02
                  db  02,00,00,02,02,00,00,02
```

Sprite para pintar el objeto que el jugador debe arrastrar hacia los objetivos (KOOPA TROPA)

```
dim_sprite_vacio  db   08, 08
data_sprite_vacio db   00, 00, 00, 00, 00, 00, 00, 00
                  db   00, 00, 00, 00, 00, 00, 00, 00
                  db   00, 00, 00, 00, 00, 00, 00, 00
                  db   00, 00, 00, 00, 00, 00, 00, 00
                  db   00, 00, 00, 00, 00, 00, 00, 00
                  db   00, 00, 00, 00, 00, 00, 00, 00
                  db   00, 00, 00, 00, 00, 00, 00, 00
                  db   00, 00, 00, 00, 00, 00, 00, 00
```

Sprite de 8x8 para limpiar.

```
dim_sprite_flcha  db   08, 08
data_sprite_flcha   db   00, 00, 28, 00, 00, 00, 00, 00
                    db   00, 00, 28, 28, 00, 00, 00, 00
                    db   28, 28, 28, 28, 28, 00, 00, 00
                    db   28, 28, 28, 28, 28, 28, 00, 00
                    db   28, 28, 28, 28, 28, 28, 00, 00
                    db   28, 28, 28, 28, 28, 00, 00, 00
                    db   00, 00, 28, 28, 00, 00, 00, 00
                    db   00, 00, 28, 00, 00, 00, 00, 00
```

Sprite para pintar la flecha de navegación de los menús.

```
mapa                db    3e8 dup (0)

;Definir datos
iniciar_juego db "INICIAR JUEGO$"
cargar_nivel  db "CARGAR NIVEL$"
configuracion db "CONFIGURACION$"
puntajes      db "PUNTAJES ALTOS$"
salir         db "SALIR$"
iniciales     db "CJMP - 201709282$"
;; JUEGO
xJugador      db 0
yJugador      db 0
puntos        dw 0
;; MENÚS
opcion3       db 0
opcion2       db 0
opcion        db 0
maximo        db 0
xFlecha       dw 0
yFlecha       dw 0
;; CONTROLES
control_arriba    db  48
control_abajo     db  50
control_izquierda db  4b
control_derecha   db  4d
control_f2        db  3c
```

Variables para el control del juego

```
;; NIVELES
nivel_1           db  "vacas/pro2/NIV.00",00
nivel_x           db  "vacas/pro2/NIV.TXT",00
handle_nivel      dw  0000
linea             db  100 dup (0)
elemento_actual   db  0
xElemento         db  0
yElemento         db  0
;; TOKENS
tk_pared      db  05,"pared"
tk_suelo      db  05,"suelo"
tk_jugador    db  07,"jugador"
tk_caja       db  04,"caja"
tk_objetivo   db  08,"objetivo"
tk_coma       db  01,","
;;
numero        db  5 dup (30)

;;MENU DE CONFIGURACION
tit_configuracion db "Controles actuales$"
tit_up db "Arriba: $"
tit_dw db "Abajo: $"
tit_lf db "Izquierda: $"
tit_rt db "Derecha: $"
val_up db "FLECHA ARRIBA   $"
val_dw db "FLECHA ABAJO    $"
val_lf db "FLECHA IZQUIERDA$"
val_rt db "FLECHA DERECHA  $"
```

Variables para el control de los menús.

Subrutinas y procesos:

- MAIN:

Movimiento de los registros:

Mov AX, @data: Mueve la dirección del segmento de datos a AX.

Mov DS, AX: Carga el valor de AX en el registro de segmento de datos (DS), estableciendo así el segmento de datos para acceder a las variables y datos del programa.

Etiqueta "inicio":

Int 03: Genera una interrupción de depuración, lo que permite la depuración del programa en un entorno de desarrollo.

Configuración del modo de video:

Mov AH, 00: Carga el número de función 0 en AH, que se utiliza para realizar operaciones de video en la BIOS.

Mov AL, 13: Carga el número de modo de video 13 en AL, que establece un modo gráfico VGA de 320x200 píxeles y 256 colores.

Int 10: Llama a la interrupción de la BIOS con el valor cargado en AH, lo que activa el modo de video especificado.

Salto a "menu_inicial":

Jmp menu_inicial: Realiza un salto incondicional a la etiqueta "menu_inicial", que representa el inicio de la lógica de tu programa.

- MENU_INICIAL:

Llamada a "menu_principal":

Call menu_principal: Llama a la subrutina "menu_principal", que probablemente contenga la lógica para mostrar el menú en pantalla y obtener la opción seleccionada por el usuario.

Obtención de la opción seleccionada:

Mov AL, [opcion]: Mueve el contenido de la variable "opcion" al registro AL, lo que te permite acceder al valor seleccionado por el usuario en el menú.

Verificación de la opción seleccionada:

Cmp AL, 0: Compara el valor en AL con 0 para determinar si la opción seleccionada es "INICIAR JUEGO".

Je ciclo_juego: Si la comparación es verdadera (igual a 0), salta a la etiqueta "ciclo_juego", que probablemente representa el bucle principal del juego.

Cmp AL, 1: Compara el valor en AL con 1 para determinar si la opción seleccionada es "CARGAR NIVEL".

Je cargar_un_nivel: Si la comparación es verdadera (igual a 1), salta a la etiqueta "cargar_un_nivel", que probablemente representa la lógica para cargar un nivel específico del juego.

Cmp AL, 2: Compara el valor en AL con 2 para determinar si la opción seleccionada es "CONFIGURACION".

Je opciones_menu_configuracion: Si la comparación es verdadera (igual a 2), salta a la etiqueta "opciones_menu_configuracion", que representa la lógica para gestionar las opciones de configuración del juego.

Cmp AL, 4: Compara el valor en AL con 4 para determinar si la opción seleccionada es "SALIR".

Je fin: Si la comparación es verdadera (igual a 4), salta a la etiqueta "fin", que representa el final del programa.

- CICLO_JUEGO:

Llamada a "pintar_mapa":

Call pintar_mapa: Llama a la subrutina "pintar_mapa", que probablemente contiene la lógica para dibujar el mapa o la pantalla del juego.

Llamada a "entrada_juego":

Call entrada_juego: Llama a la subrutina "entrada_juego", que probablemente contiene la lógica para obtener y procesar la entrada del jugador, como las teclas presionadas o los movimientos realizados.

Salto a "ciclo_juego":

Jmp ciclo_juego: Realiza un salto incondicional a la etiqueta "ciclo_juego", lo que permite que el ciclo se repita continuamente y el juego siga actualizándose.

- MENU_PRINCIPAL:

Llamada a "clear_pantalla":

Call clear_pantalla: Llama a la subrutina "clear_pantalla", que probablemente contiene la lógica para limpiar la pantalla antes de mostrar el menú.

Reinicio de variables:

Mov AL, 0: Carga el valor 0 en AL.

Mov [opcion], AL: Guarda el valor de AL en la variable "opcion", reiniciando la variable de salida.

Mov AL, 5: Carga el valor 5 en AL.

Mov [maximo], AL: Guarda el valor de AL en la variable "maximo", que probablemente representa el número máximo de opciones en el menú.

Configuración de posición de flecha:

Mov AX, 50: Carga el valor 50 en AX, que representa la posición horizontal de la flecha en el menú.

Mov BX, 28: Carga el valor 28 en BX, que representa la posición vertical de la flecha en el menú.

Mov [xFlecha], AX: Guarda el valor de AX en la variable "xFlecha", que representa la posición horizontal de la flecha.

Mov [yFlecha], BX: Guarda el valor de BX en la variable "yFlecha", que representa la posición vertical de la flecha.

Impresión de opciones del menú:

Configuración del formato de impresión: Establece los valores de los registros para imprimir las opciones del menú en pantalla.

Mov DX, offset <nombre_opcion>: Carga la dirección de la cadena de texto de cada opción del menú en DX.

Mov AH, 09: Carga el valor 09 en AH para realizar la impresión de la cadena de texto en pantalla.

Int 21: Llama a la interrupción 21h del sistema operativo para imprimir la cadena de texto en pantalla.

Llamada a "pintar_flecha":

Call pintar_flecha: Llama a la subrutina "pintar_flecha", que probablemente contiene la lógica para dibujar la flecha que indica la opción seleccionada en el menú.

Lectura de tecla:

Mov AH, 00: Carga el valor 00 en AH para realizar la lectura de una tecla presionada por el usuario.

Int 16: Llama a la interrupción 16h del sistema operativo para leer la tecla presionada.

Comparaciones y saltos condicionales: Compara el valor en AH con diferentes códigos de tecla para determinar la acción a realizar.

Jmp entrada_menu_principal: Salta a la etiqueta "entrada_menu_principal" para continuar leyendo teclas hasta que se realice una acción determinada.

- CAMBIO_CONTROLES:

Llamada a "clear_pantalla":

Call clear_pantalla: Llama a la subrutina "clear_pantalla", que probablemente contiene la lógica para limpiar la pantalla antes de mostrar las solicitudes de control.

Solicitar control de la derecha:

Mov DL, 02: Carga el valor 02 en DL, que representa la columna de la pantalla donde se mostrará la solicitud del control de la derecha.

Mov DH, 05: Carga el valor 05 en DH, que representa la fila de la pantalla donde se mostrará la solicitud del control de la derecha.

Mov DX, offset promp_rt: Carga la dirección de la cadena de texto "promp_rt" en DX, que contiene la solicitud del control de la derecha.

Mov AH, 09: Carga el valor 09 en AH para realizar la impresión de la cadena de texto en pantalla.

Int 21: Llama a la interrupción 21h del sistema operativo para imprimir la cadena de texto en pantalla.

Leer ingreso de teclado y validar:

Mov AH, 00: Carga el valor 00 en AH para realizar la lectura de una tecla presionada por el usuario.

Int 16: Llama a la interrupción 16h del sistema operativo para leer la tecla presionada.

Push AX: Guarda el valor de AX en la pila.

Call validar_nuevo_control: Llama a la subrutina "validar_nuevo_control" para validar el ingreso del control.

Comparaciones y saltos condicionales:

Cmp AH, 48: Compara el valor en AH con 48, que representa el código ASCII de la tecla '0'.

Je no_finalizacion_derecha: Salta a la etiqueta "no_finalizacion_derecha" si las teclas presionadas no coinciden con los códigos esperados.

Finalización del control de la derecha:

Mov CX, 10: Carga el valor 10 en CX, que representa la longitud de la cadena de texto "val_rt".

Mov DI, offset val_rt: Carga la dirección de la cadena de texto "val_rt" en DI.

Call memset: Llama a la subrutina "memset" para llenar la cadena de texto "val_rt" con caracteres vacíos.

Mov SI, offset val_rt: Carga la dirección de la cadena de texto "val_rt" en SI.

Mov DI, offset finalizacion: Carga la dirección de la cadena de texto "finalizacion" en DI.

Mov DL, [DI]: Carga el contenido de la dirección DI en DL.

Mov [SI], DL: Guarda el valor de DL en la dirección SI, copiando el carácter de finalización en "val_rt".

Mov control_derecha, AH: Guarda el valor de AH en la variable "control_derecha" para cambiar el control de la derecha.

Repetición del proceso para los demás controles:

Se repiten los pasos 2-5 para los controles de abajo, arriba y izquierda, con las correspondientes cadenas de texto y variables.

Llamada a "clear_pantalla":

Call clear_pantalla: Llama a la subrutina "clear_pantalla" nuevamente para limpiar la pantalla después de la configuración de los controles.

Salto a "menu_inicial":

Jmp menu_inicial: Salta a la etiqueta "menu_inicial" para volver al menú inicial del programa.

- OPCIONES_MENU_CONFIGURACIÓN:

Llamada a "menu_configuracion":

Call menu_configuracion: Llama a la subrutina "menu_configuracion" para mostrar el menú de configuración al usuario.

Obtención de la opción seleccionada:

Mov DL, [opcion2]: Obtiene el valor de la variable "opcion2" que contiene la opción seleccionada por el usuario.

Comparaciones y saltos condicionales:

Cmp DL, 0: Compara el valor en DL con 0, que representa la opción de cambio de configuración.

Je cambio_controles: Salta a la etiqueta "cambio_controles" si la opción seleccionada es 0, lo que indica que se debe realizar un cambio de configuración.

Realizar cambio de configuración:

Jmp cambio_controles: Salta a la etiqueta "cambio_controles" para ejecutar la subrutina "cambio_controles" y permitir al usuario configurar los controles del juego.

Comparación adicional y salto condicional:

Cmp DL, 1: Compara el valor en DL con 1, que representa la opción de regresar al menú principal.

Je menu_inicial: Salta a la etiqueta "menu_inicial" si la opción seleccionada es 1, lo que indica que se debe regresar al menú principal del programa.

Fin de la subrutina:

La subrutina "opciones_menu_configuracion" termina en este punto y el control regresa al punto donde se llamó a esta subrutina.

- MENU_CONFIGURACION:

Limpieza de pantalla:

Call clear_pantalla: Llama a la subrutina "clear_pantalla" para limpiar la pantalla antes de mostrar el menú de configuración.

Inicialización de variables:

Mov AL, 0: Establece el valor 0 en AL.

Mov [opcion2], AL: Reinicia la variable "opcion2" estableciendo su valor en 0.

Mov AL, 2: Establece el valor 2 en AL.

Mov [maximo], AL: Inicializa la variable "maximo" con el valor 2.

Mov AX, 01: Establece el valor 01 en AX.

Mov BX, 14b: Establece el valor 14b en BX.

Impresión de títulos y valores:

Se utilizan instrucciones de interrupción de video (int 10) y de interrupción de DOS (int 21) para imprimir los títulos y valores en diferentes ubicaciones de la pantalla.

Pintar la flecha de selección:

Call pintar_flecha: Llama a la subrutina "pintar_flecha" para dibujar la flecha de selección en la ubicación especificada por las variables "xFlecha" y "yFlecha".

Leer la tecla presionada:

Se utiliza la interrupción de teclado (int 16) para leer la tecla presionada por el usuario.

Se comparan los valores de las teclas presionadas (AH) con diferentes opciones y se toman acciones correspondientes.

Restar o sumar opciones:

Se decrementa o incrementa el valor de la variable "opcion2" según la tecla presionada por el usuario.

Se realizan comparaciones para verificar si se alcanzó el valor mínimo o máximo permitido.

Mover la flecha:

La flecha de selección se mueve a la opción correspondiente según el valor de la variable "opcion2".

Se utiliza un bucle para calcular las coordenadas de la flecha en función del valor de "opcion2" y se llama a la subrutina "pintar_flecha" para dibujarla.

Finalizar el menú de configuración:

Si se presiona la tecla F1 (valor 3b en AH), se finaliza la subrutina y se retorna al punto donde se llamó.

En caso contrario, se vuelve al paso 5 para leer la siguiente tecla presionada.

- MENU_PAUSA:

Llamada a la subrutina "menu_pausa_seleccion":

Call menu_pausa_seleccion: Se realiza una llamada a la subrutina "menu_pausa_seleccion" para mostrar el menú de selección dentro del menú de pausa.

Obtención de la opción seleccionada:

Mov DL, [opcion3]: Se mueve el contenido de la variable "opcion3" a DL, que representa la opción seleccionada por el jugador.

Comparación de la opción seleccionada:

Se utiliza la instrucción cmp para comparar el contenido de DL con diferentes valores:

Cmp DL, 0: Compara si la opción seleccionada es igual a 0.

Je ciclo_juego: Si la comparación es verdadera (igual a 0), se salta a la etiqueta "ciclo_juego" para realizar un cambio de configuración.

Cmp DL, 1: Compara si la opción seleccionada es igual a 1.

Je menu_inicial: Si la comparación es verdadera (igual a 1), se salta a la etiqueta "menu_inicial" para regresar al menú principal.

- MENU_PAUSA_SELECCION:

Limpieza de la pantalla:

Call clear_pantalla: Se realiza una llamada a la subrutina "clear_pantalla" para limpiar la pantalla antes de mostrar el menú de pausa.

Reinicio de la variable de salida y configuración de valores iniciales:

Mov AL, 0: Se mueve el valor 0 a AL.

Mov [opcion3], AL: Se reinicia la variable de salida "opcion3" estableciéndola en 0.

Mov AL, 2: Se mueve el valor 2 a AL.

Mov [maximo], AL: Se establece el valor máximo de opciones en 2.

Impresión de las opciones del menú:

Se utilizan las instrucciones mov, int 10 y int 21 para imprimir los títulos y las opciones del menú en diferentes posiciones de la pantalla.

Llamada a la subrutina "pintar_flecha":

Call pintar_flecha: Se realiza una llamada a la subrutina "pintar_flecha" para dibujar la flecha en la posición inicial del menú.

Lectura de la tecla presionada:

Entrada_menu_pausa: Se inicia un bucle para leer la tecla presionada por el jugador.

Mov AH, 00: Se mueve el valor 00 a AH para leer la tecla.

Int 16: Se realiza una interrupción para leer la tecla presionada.

Cmp AH, 48: Se compara el valor leído con 48 para verificar si se presionó la tecla "2" (tecla de abajo).

Je restar_opcion_menu_pausa: Si la comparación es verdadera, se salta a la etiqueta "restar_opcion_menu_pausa" para restar una opción.

Cmp AH, 50: Se compara el valor leído con 50 para verificar si se presionó la tecla "8" (tecla de arriba).

Je sumar_opcion_menu_pausa: Si la comparación es verdadera, se salta a la etiqueta "sumar_opcion_menu_pausa" para sumar una opción.

Cmp AH, 3b: Se compara el valor leído con 3B para verificar si se presionó la tecla F1 (tecla de salida).

Je fin_menu_pausa: Si la comparación es verdadera, se salta a la etiqueta "fin_menu_pausa" para finalizar el menú de pausa.

Manipulación de la opción seleccionada:

Restar_opcion_menu_pausa: Se resta 1 al valor de la variable "opcion3" y se realizan comprobaciones adicionales para asegurarse de que no exceda los límites.

Sumar_opcion_menu_pausa: Se suma 1 al valor de la variable "opcion3" y se realizan comprobaciones adicionales para asegurarse de que no exceda los límites.

Volver_a_cero_pausa: Se establece la variable "opcion3" en 0 si alcanza el valor mínimo.

Volver_a_maximo_pausa: Se establece la variable "opcion3" en el valor máximo si alcanza el valor máximo.

Mover_flecha_menu_pausa: Se actualiza la posición de la flecha en función de la opción seleccionada.

Pintar_flecha_menu_pausa: Se dibuja la flecha en la posición correspondiente.

Retorno y finalización del menú de pausa:

Fin_menu_pausa: Se realiza un retorno de la subrutina para salir del menú de pausa.

- CARGAR_UN_NIVEL:

Apertura del archivo:

Mov AL, 00: Se mueve el valor 00 a AL.

Mov DX, offset nivel_x: Se carga la dirección del nombre del archivo en DX.

Mov AH, 3d: Se realiza una llamada a la interrupción 21h, función 3Dh para abrir el archivo.

Int 21: Se realiza la interrupción 21h para abrir el archivo.

Jc main: Si se produce un error durante la apertura del archivo (indicado por el flag de acarreo), se salta a la etiqueta "main" para manejar el error.

Almacenamiento del handle del archivo:

Mov [handle_nivel], AX: Se almacena el handle del archivo abierto en la variable "handle_nivel".

Ciclo para leer las líneas del archivo:

Etiqueta "ciclo_lineas": Se inicia un bucle para leer cada línea del archivo.

Mov BX, [handle_nivel]: Se mueve el handle del archivo a BX para usarlo en la lectura de la siguiente línea.

Call siguiente_linea: Se realiza una llamada a la subrutina "siguiente_linea" para leer la siguiente línea del archivo.

Cmp DL, 0ff: Se compara el valor de DL (carácter leído) con 0xFF para verificar si se alcanzó el final del archivo.

Je fin_parseo: Si la comparación es verdadera, se salta a la etiqueta "fin_parseo" para finalizar el proceso de carga del nivel.

Lógica de parseo de cada línea:

Etiquetas "es_pared", "es_caja", "es_suelo", "es_objetivo" y "es_jugador": Se utilizan para verificar si el primer elemento de la línea coincide con una cadena específica (pared, caja, suelo, objetivo o jugador).

Mov AL, valor_elemento: Se mueve el valor correspondiente al tipo de elemento (PARED, CAJA, SUELO, OBJETIVO o JUGADOR) a AL.

Mov [elemento_actual], AL: Se almacena el tipo de elemento actual en la variable "elemento_actual".

Jmp continuar_parseo0: Se salta a la etiqueta "continuar_parseo0" para continuar con el parseo de la línea.

Continuación del parseo de la línea:

Etiqueta "continuar_parseo0": Se realiza una serie de operaciones para ignorar espacios en blanco y obtener el valor numérico correspondiente a las coordenadas x del elemento.

Call leer_cadena_numerica: Se realiza una llamada a la subrutina "leer_cadena_numerica" para leer una cadena numérica de la línea.

Push DI: Se guarda el valor de DI en la pila.

Mov DI, offset numero: Se carga la dirección de la cadena numérica en DI.

Call cadenaAnum: Se realiza una llamada a la subrutina "cadenaAnum" para convertir la cadena numérica a un valor numérico.

Mov [xElemento], AL: Se almacena el valor numérico obtenido en la variable "xElemento".

Pop DI: Se recupera el valor original de DI desde la pila.

Mov AL, [DI]: Se carga el siguiente carácter de la línea en AL para verificar si hay espacios en blanco.

Cmp AL, 20: Se compara el valor de AL con 20 (espacio en blanco).

Jne ciclo_lineas: Si la comparación es falsa, se salta de regreso al inicio del bucle para leer la siguiente línea.

Obtención de las coordenadas y finalización del parseo de la línea:

Etiqueta "continuar_parseo1": Se realiza una serie de operaciones para ignorar espacios en blanco y obtener el valor numérico correspondiente a las coordenadas y del elemento.

Etiqueta "continuar_parseo2": Se realiza una serie de operaciones para ignorar espacios en blanco y obtener una coma de la línea.

Cmp AL, 20: Se compara el valor de AL con 20 (espacio en blanco) para verificar si hay espacios en blanco después de la coma.

Jne ciclo_lineas: Si la comparación es verdadera, se salta de regreso al inicio del bucle para leer la siguiente línea.

Call leer_cadena_numerica: Se realiza una llamada a la subrutina "leer_cadena_numerica" para leer una cadena numérica de la línea.

Push DI: Se guarda el valor de DI en la pila.

Mov DI, offset numero: Se carga la dirección de la cadena numérica en DI.

Call cadenaAnum: Se realiza una llamada a la subrutina "cadenaAnum" para convertir la cadena numérica a un valor numérico.

Mov [yElemento], AL: Se almacena el valor numérico obtenido en la variable "yElemento".

Pop DI: Se recupera el valor original de DI desde la pila.

Mov AL, [DI]: Se carga el siguiente carácter de la línea en AL para verificar si hay espacios en blanco o si es el final de la línea.

Cmp AL, 20: Se compara el valor de AL con 20 (espacio en blanco).

Je continuar_parseo1: Si la comparación es verdadera, se salta a la etiqueta "continuar_parseo1" para ignorar espacios en blanco.

Jne ver_final_de_linea: Si la comparación es falsa, se salta a la etiqueta "ver_final_de_linea" para verificar si es el final de la línea.

Uso de la información obtenida:

Mov DL, [elemento_actual]: Se mueve el tipo de elemento actual a DL.

Mov AH, [xElemento]: Se mueve la coordenada x del elemento a AH.

Mov AL, [yElemento]: Se mueve la coordenada y del elemento a AL.

Call colocar_en_mapa: Se realiza una llamada a la subrutina "colocar_en_mapa" para colocar el elemento en el mapa.

Mov AL, JUGADOR: Se carga el valor del tipo de elemento "JUGADOR" en AL.

Cmp AL, [elemento_actual]: Se compara el valor en AL con el tipo de elemento actual para verificar si es el jugador.

Je guardar_coordenadas_jugador: Si la comparación es verdadera, se salta a la etiqueta "guardar_coordenadas_jugador" para guardar las coordenadas del jugador.

Guardado de las coordenadas del jugador:

Etiqueta "guardar_coordenadas_jugador": Se realiza una serie de operaciones para guardar las coordenadas del jugador en las variables "xJugador" y "yJugador".

Jmp ciclo_lineas: Se salta de regreso al inicio del bucle para leer la siguiente línea.

Finalización del parseo y cierre del archivo:

Etiqueta "fin_parseo": Se llega a esta etiqueta cuando se ha alcanzado el final del archivo y se ha completado el parseo de todas las líneas.

Mov AH, 3e: Se mueve el valor 3Eh a AH para realizar una llamada a la interrupción 21h, función 3Eh para cerrar el archivo.

Mov BX, [handle_nivel]: Se carga el handle del archivo en BX.

Int 21: Se realiza la interrupción 21h para cerrar el archivo.

Int 03: Se genera una interrupción 3 para detener la ejecución y permitir la depuración.

Jmp ciclo_juego: Se salta a la etiqueta "ciclo_juego" para continuar con el ciclo del juego.

Jmp fin: Se salta a la etiqueta "fin" para finalizar la subrutina.

- VALIDAR_NUEVO_CONTROL:

Limpieza del buffer de finalización:

Mov CX, 10: Se carga el valor 10 en CX, que representa la longitud del buffer de finalización.

Mov DI, offset finalizacion: Se carga la dirección de memoria del buffer de finalización en DI.

Call memset: Se realiza una llamada a la subrutina "memset" para llenar el buffer de finalización con caracteres nulos (0).

Validación de las teclas con nombres compuestos:

Cmp AH, 48: Se compara el valor en AH con 48 (código ASCII de '0') para verificar si se presionó la tecla "0" (arriba).

Je init_es_arriba: Si la comparación es verdadera, se salta a la etiqueta "init_es_arriba" para iniciar el proceso de validación de la tecla "arriba".

Cmp AH, 4b: Se compara el valor en AH con 4B (código ASCII de 'K') para verificar si se presionó la tecla "K" (izquierda).

Je init_es_izquierda: Si la comparación es verdadera, se salta a la etiqueta "init_es_izquierda" para iniciar el proceso de validación de la tecla "izquierda".

Cmp AH, 50: Se compara el valor en AH con 50 (código ASCII de 'P') para verificar si se presionó la tecla "P" (abajo).

Je init_es_abajo: Si la comparación es verdadera, se salta a la etiqueta "init_es_abajo" para iniciar el proceso de validación de la tecla "abajo".

Cmp AH, 4d: Se compara el valor en AH con 4D (código ASCII de 'M') para verificar si se presionó la tecla "M" (derecha).

Je init_es_derecha: Si la comparación es verdadera, se salta a la etiqueta "init_es_derecha" para iniciar el proceso de validación de la tecla "derecha".

Jmp es_cualquiera: Si ninguna de las comparaciones anteriores es verdadera, se salta a la etiqueta "es_cualquiera" para realizar la validación de cualquier otra tecla.

Inicialización de los punteros de validación:

Etiqueta "init_es_arriba": Se mueve la dirección de la cadena "val_up" a SI y la dirección de la cadena "arrow_up" a DI, preparando los punteros para la validación de la tecla "arriba".

Etiqueta "init_es_izquierda": Se mueve la dirección de la cadena "val_lf" a SI y la dirección de la cadena "arrow_lf" a DI, preparando los punteros para la validación de la tecla "izquierda".

Etiqueta "init_es_abajo": Se mueve la dirección de la cadena "val_dw" a SI y la dirección de la cadena "arrow_dw" a DI, preparando los punteros para la validación de la tecla "abajo".

Etiqueta "init_es_derecha": Se mueve la dirección de la cadena "val_rt" a SI y la dirección de la cadena "arrow_rt" a DI, preparando los punteros para la validación de la tecla "derecha".

Validación de la tecla "arriba":

Etiqueta "es_arriba": Se copia el contenido de la memoria apuntada por DI (cadena "arrow_up") en el registro DL.

Mov [SI], DL: Se guarda el valor de DL en la memoria apuntada por SI (cadena "val_up").

Inc SI: Se incrementa el valor de SI para apuntar al siguiente elemento de la cadena "val_up".

Inc DI: Se incrementa el valor de DI para apuntar al siguiente elemento de la cadena "arrow_up".

Loop es_arriba: Se repite el proceso anterior hasta que CX (inicializado en 11) llegue a cero.

Jmp fin_conversion: Se salta a la etiqueta "fin_conversion" para finalizar el proceso de validación.

Validación de la tecla "abajo":

Etiqueta "es_abajo": Funciona de manera similar a "es_arriba", pero con las cadenas "arrow_dw" y "val_dw".

Validación de la tecla "izquierda":

Etiqueta "es_izquierda": Funciona de manera similar a "es_arriba", pero con las cadenas "arrow_lf" y "val_lf".

Validación de la tecla "derecha":

Etiqueta "es_derecha": Funciona de manera similar a "es_arriba", pero con las cadenas "arrow_rt" y "val_rt".

Validación de cualquier otra tecla:

Etiqueta "es_cualquiera": Se mueve el valor en AL (valor de la tecla presionada) al buffer de finalización apuntado por SI.

Jmp fin_conversion: Se salta a la etiqueta "fin_conversion" para finalizar el proceso de validación.

Finalización del proceso de validación:

Ret: Se retorna de la subrutina.

- ENTRADA_JUEGO:

Lectura de la entrada del teclado:

Mov AH, 01: Se establece el valor 01 en AH para realizar una llamada a la función de BIOS para verificar si hay una tecla presionada en el buffer de entrada.

Int 16: Se realiza una interrupción de software con el valor 16h para llamar a la función de BIOS.

Jz fin_entrada_juego: Si la tecla no está presionada (el flag zero está activado), se salta a la etiqueta "fin_entrada_juego" para finalizar la subrutina.

Obtención del scan code:

Mov AH, 00: Se establece el valor 00 en AH para realizar una llamada a la función de BIOS para obtener el scan code de la tecla presionada.

Int 16: Se realiza una interrupción de software con el valor 16h para llamar a la función de BIOS.

Cmp AH, [control_arriba]: Se compara el valor obtenido en AH con el valor almacenado en la memoria apuntada por [control_arriba] para verificar si se presionó la tecla correspondiente al control "arriba".

Je mover_jugador_arr: Si la comparación es verdadera, se salta a la etiqueta "mover_jugador_arr" para ejecutar el código correspondiente al movimiento del jugador hacia arriba.

Cmp AH, [control_abajo]: Se compara el valor obtenido en AH con el valor almacenado en la memoria apuntada por [control_abajo] para verificar si se presionó la tecla correspondiente al control "abajo".

Je mover_jugador_aba: Si la comparación es verdadera, se salta a la etiqueta "mover_jugador_aba" para ejecutar el código correspondiente al movimiento del jugador hacia abajo.

Cmp AH, [control_izquierda]: Se compara el valor obtenido en AH con el valor almacenado en la memoria apuntada por [control_izquierda] para verificar si se presionó la tecla correspondiente al control "izquierda".

Je mover_jugador_izq: Si la comparación es verdadera, se salta a la etiqueta "mover_jugador_izq" para ejecutar el código correspondiente al movimiento del jugador hacia la izquierda.

Cmp AH, [control_derecha]: Se compara el valor obtenido en AH con el valor almacenado en la memoria apuntada por [control_derecha] para verificar si se presionó la tecla correspondiente al control "derecha".

Je mover_jugador_der: Si la comparación es verdadera, se salta a la etiqueta "mover_jugador_der" para ejecutar el código correspondiente al movimiento del jugador hacia la derecha.

Cmp AH, [control_f2]: Se compara el valor obtenido en AH con el valor almacenado en la memoria apuntada por [control_f2] para verificar si se presionó la tecla correspondiente al control "F2".

Je menu_pausa: Si la comparación es verdadera, se salta a la etiqueta "menu_pausa" para ejecutar el código correspondiente al acceso al menú de pausa.

Ret: Si ninguna de las comparaciones anteriores es verdadera, se retorna de la subrutina.

Movimiento del jugador hacia arriba:

Etiqueta "mover_jugador_arr": Se obtienen las coordenadas actuales del jugador (xJugador en AH, yJugador en AL).

Dec AL: Se decrementa el valor de AL para mover el jugador hacia arriba.

Push AX: Se guarda el valor de AX (coordenadas del jugador) en la pila para su posterior uso.

Call obtener_de_mapa: Se realiza una llamada a una función externa para obtener el elemento en el mapa correspondiente a las nuevas coordenadas del jugador.

Pop AX: Se recupera el valor original de AX (coordenadas del jugador) de la pila.

Cmp DL, PARED: Se compara el elemento obtenido en DL con el valor de "PARED" para verificar si hay una pared en las nuevas coordenadas.

Je hay_pared_arriba: Si la comparación es verdadera, se salta a la etiqueta "hay_pared_arriba" para ejecutar el código correspondiente al caso de colisión con una pared.

Mov [yJugador], AL: Se guarda el nuevo valor de AL (coordenada y del jugador) en la memoria apuntada por [yJugador].

Mov DL, JUGADOR: Se establece el valor de "JUGADOR" en DL para indicar la presencia del jugador en el nuevo elemento del mapa.

Push AX: Se guarda el valor de AX (coordenadas del jugador) en la pila para su posterior uso.

Call colocar_en_mapa: Se realiza una llamada a una función externa para colocar el jugador en el nuevo elemento del mapa.

Pop AX: Se recupera el valor original de AX (coordenadas del jugador) de la pila.

Mov DL, SUELO: Se establece el valor de "SUELO" en DL para indicar la ausencia del jugador en el elemento anterior del mapa.

Inc AL: Se incrementa el valor de AL para obtener las coordenadas del elemento anterior del mapa.

Call colocar_en_mapa: Se realiza una llamada a una función externa para colocar "SUELO" en el elemento anterior del mapa.

Ret: Se retorna de la subrutina.

Casos de colisión con pared:

Etiquetas "hay_pared_arriba", "hay_pared_abajo", "hay_pared_izquierda", "hay_pared_derecha": Se utilizan para indicar que el jugador ha chocado con una pared en las respectivas direcciones.

Ret: Se retorna de la subrutina.

Movimiento del jugador hacia abajo, izquierda y derecha:

Funcionan de manera similar a "mover_jugador_arr", pero con las respectivas etiquetas, coordenadas y comparaciones correspondientes a cada dirección de movimiento.

Etiqueta "fin_entrada_juego":

Ret: Se retorna de la subrutina cuando no hay teclas presionadas en el buffer de entrada.

- COLOCAR_EN_MAPA:

Preparación de los registros:

Mov CX, AX: Se copia el contenido de AX en CX para preservar el valor original.

Mov BL, 28: Se carga el valor 28 en BL, que representa el ancho del mapa.

Cálculo de la posición en el mapa:

Mul BL: Se realiza la multiplicación de AL por BL, almacenando el resultado en AX. Esto se hace para calcular el desplazamiento horizontal (x) en el mapa.

Mov CL, CH: Se copia el contenido de CH en CL para preservar el valor original de CH.

Mov CH, 00: Se establece CH en 00 para representar el desplazamiento vertical (y) en el mapa.

Add AX, CX: Se suma CX al valor de AX para obtener la posición final en el mapa.

Mov DI, offset mapa: Se carga en DI la dirección base del mapa.

Acceso a la posición del mapa y colocación del elemento:

Add DI, AX: Se suma AX al valor en DI para apuntar a la posición correcta en el mapa.

Mov [DI], DL: Se mueve el contenido de DL (el elemento a colocar) a la posición indicada por DI en el mapa.

Retorno de la subrutina:

Ret: Se retorna de la subrutina.

- PINTAR_MAPA:

Inicialización de variables:

Mov AL, 00: Se inicializa AL con el valor 00, que representa la fila actual del mapa.

Ciclo vertical:

Cmp AL, 19: Se compara AL con el valor 19 para verificar si se ha alcanzado la última fila del mapa.

Je fin_pintar_mapa: Si se ha alcanzado la última fila, se salta a la etiqueta "fin_pintar_mapa".

Mov AH, 00: Se inicializa AH con el valor 00, que representa la columna actual del mapa.

Ciclo horizontal:

Cmp AH, 28: Se compara AH con el valor 28 para verificar si se ha alcanzado la última columna del mapa.

Je continuar_v: Si se ha alcanzado la última columna, se salta a la etiqueta "continuar_v".

Push AX: Se guarda el contenido de AX (fila y columna) en la pila para preservarlos durante la llamada a otras subrutinas.

Call obtener_de_mapa: Se llama a la subrutina "obtener_de_mapa" para obtener el elemento en la posición actual del mapa. El resultado se guarda en DL.

Pop AX: Se recupera el contenido original de AX desde la pila.

Verificación del elemento y pintado correspondiente:

Se realizan comparaciones entre DL (el elemento del mapa) y diferentes valores (NADA, JUGADOR, PARED, CAJA, OBJETIVO, SUELO) para determinar qué sprite se debe pintar en la posición actual.

Dependiendo del elemento, se llama a la subrutina correspondiente para pintar el sprite adecuado en la pantalla.

Actualización de la posición horizontal y salto al siguiente ciclo horizontal:

Inc AH: Se incrementa el valor de AH para pasar a la siguiente columna del mapa.

Jmp ciclo_h: Se salta de vuelta al ciclo horizontal para procesar la siguiente columna.

Actualización de la posición vertical y salto al siguiente ciclo vertical:

Inc AL: Se incrementa el valor de AL para pasar a la siguiente fila del mapa.

Jmp ciclo_v: Se salta de vuelta al ciclo vertical para procesar la siguiente fila.

Fin de la subrutina:

Ret: Se retorna de la subrutina.

- PINTAR_PIXEL:

Guardar registros:

Se hace un push de los registros AX, BX, CX, DX, DI y SI en la pila para preservar su contenido durante la ejecución de la subrutina.

Configuración del segmento de destino:

Mov DX, 0a000: Se carga el valor 0a000 en DX, que representa el segmento de memoria de video (modo gráfico) en la memoria VGA.

Cálculo de la posición de memoria:

Mov SI, AX: Se guarda el valor de AX en SI, que contiene el desplazamiento horizontal del píxel.

Mov AX, 140: Se carga el valor 140 en AX, que representa el ancho de la pantalla (320 píxeles) dividido por 2.

Mul BX: Se multiplica el valor de BX, que contiene el desplazamiento vertical del píxel, por AX.

Add AX, SI: Se suma el valor de SI al resultado anterior para obtener la posición de memoria final.

Mov DI, AX: Se guarda la posición de memoria en DI.

Pintar el píxel:

Mov [DI], CL: Se guarda el valor de CL (color del píxel) en la posición de memoria correspondiente, lo que pinta el píxel en la pantalla.

Restaurar registros y salir de la subrutina:

Se hace un pop inverso de los registros y se realiza un ret para salir de la subrutina.

- PINTAR_SPRITE:

Guardar registros:

Se hace un push de los registros DI, SI, AX, BX y CX en la pila para preservar su contenido durante la ejecución de la subrutina.

Configuración de las dimensiones del sprite:

Inc SI: Se incrementa SI para apuntar a la dimensión vertical del sprite.

Mov DH, [SI]: Se mueve el valor de la dimensión vertical del sprite a DH.

Dec SI: Se decrementa SI para apuntar a la dirección de la dimensión horizontal del sprite.

Inicio del bucle de pintado:

Se compara DH con 00 para verificar si se ha alcanzado el final del sprite.

Je fin_pintar_sprite: Si DH es igual a 00, se salta a la etiqueta "fin_pintar_sprite".

Pintar fila del sprite:

Se guarda el valor de AX en la pila para preservar el desplazamiento horizontal actual.

Mov DL, [SI]: Se carga el valor de la dimensión horizontal actual del sprite en DL.

Pintar píxeles de la fila:

Se compara DL con 00 para verificar si se ha alcanzado el final de la fila.

Je pintar_siguiente_fila: Si DL es igual a 00, se salta a la etiqueta "pintar_siguiente_fila".

Mov CL, [DI]: Se carga el color del píxel actual en CL.

Se llama a la subrutina "pintar_pixel" para pintar el píxel en la pantalla.

Se incrementa AX, DI y se decrementa DL para avanzar a la siguiente posición de píxel en la fila.

Se salta de vuelta a "pintar_fila" para pintar el siguiente píxel de la fila.

Pintar siguiente fila del sprite:

Se hace un pop de AX para restaurar el desplazamiento horizontal original.

Se incrementa BX para avanzar a la siguiente fila del sprite.

Se decrementa DH para indicar que falta una fila por pintar.

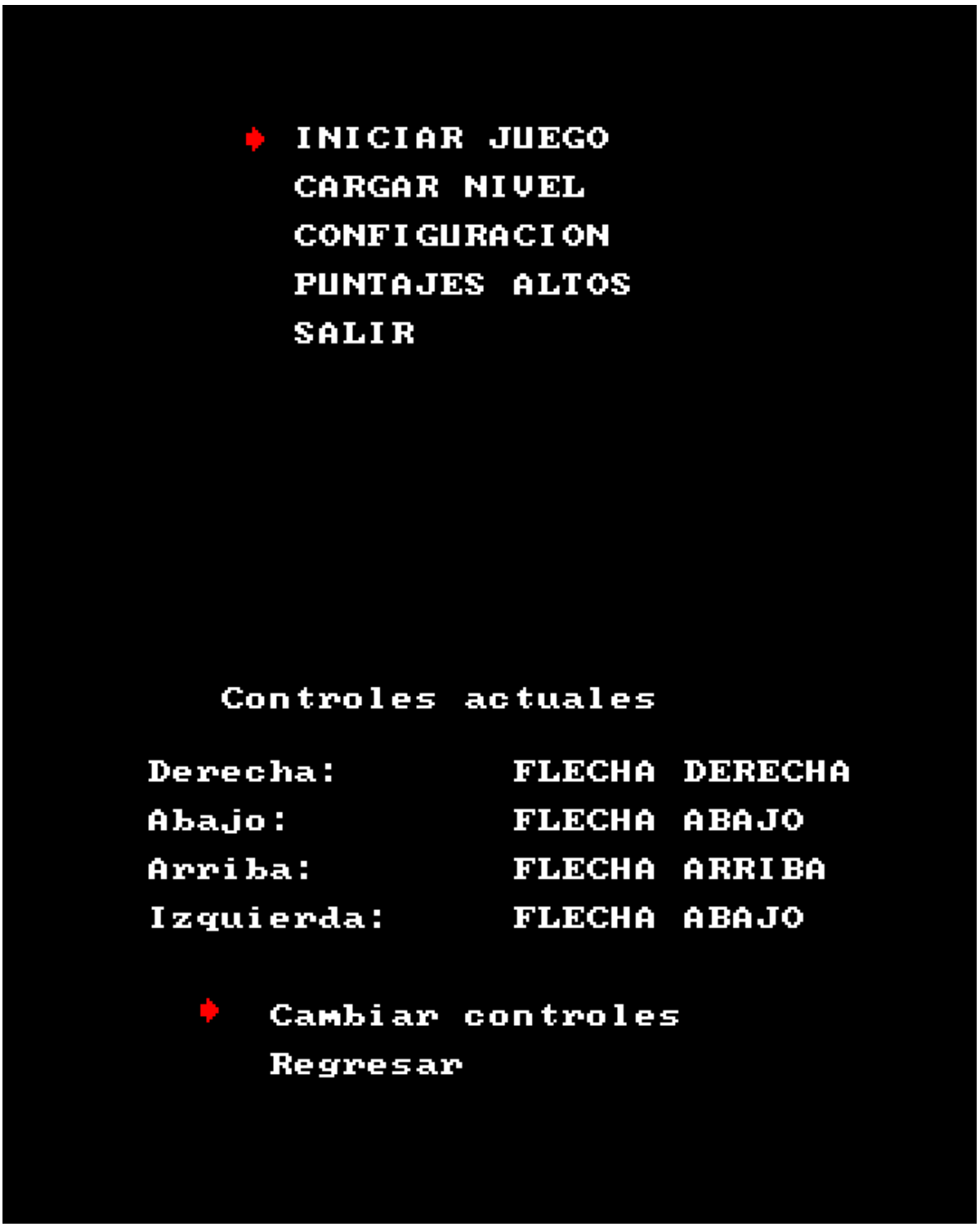
Se salta de vuelta a "inicio_pintar_fila" para pintar la siguiente fila del sprite.

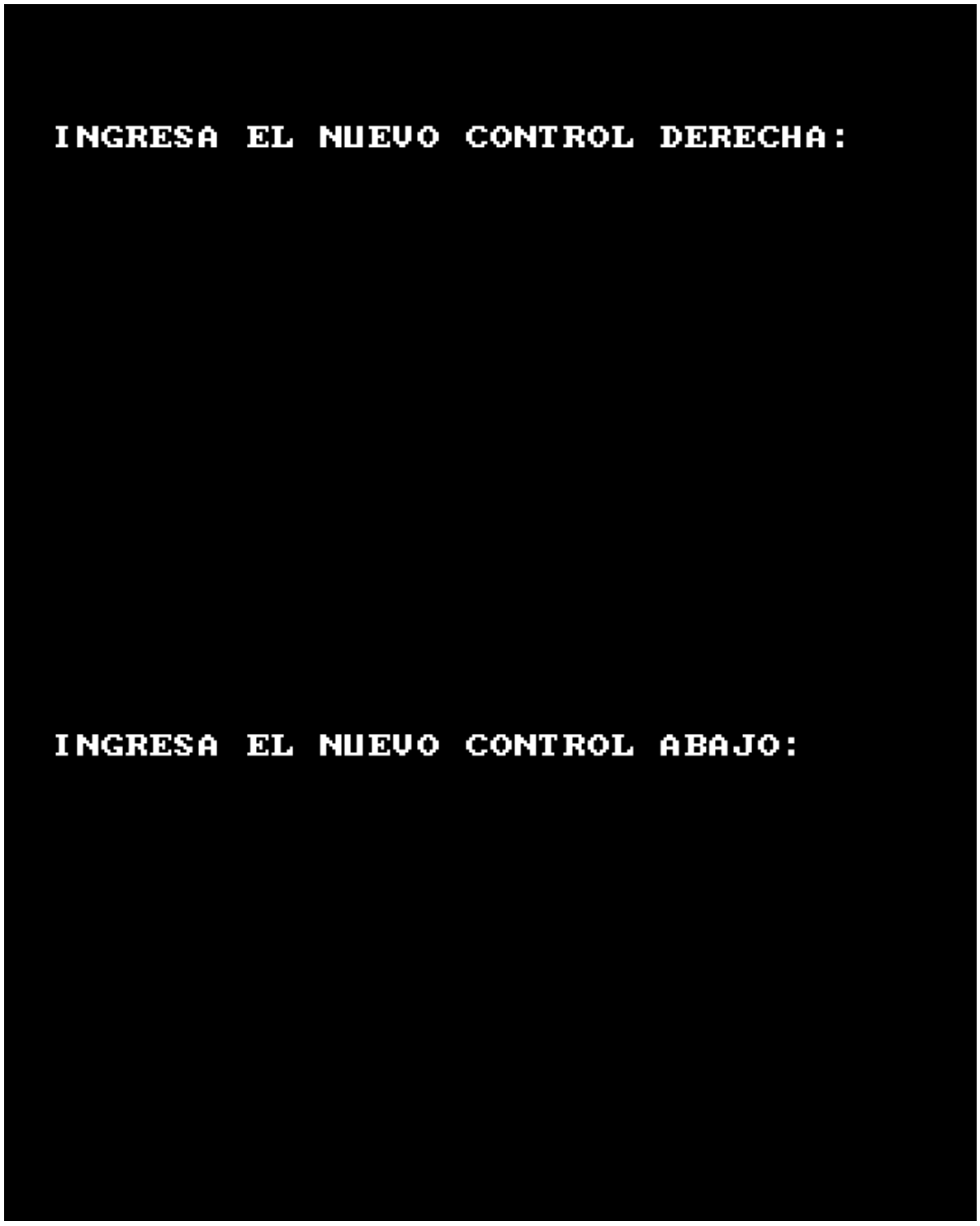
Fin del pintado del sprite:

Se hace un pop de CX, BX, AX, SI y DI para restaurar los registros originales.

Se realiza un ret para salir de la subrutina.

MANUAL DE USUARIO




INGRESA EL NUEVO CONTROL DERECHA:
INGRESA EL NUEVO CONTROL ABAJO:


INGRESA EL NUEVO CONTROL ARRIBA:
INGRESA EL NUEVO CONTROL IZQUIERDA:


PAUSA
CONTINUAR
REGRESAR AL MENU