Universidad Nacional Autónoma de México

Facultad de Ingeniería

Tarea 7

Alumno: Hernandez Chavez Samuel
Profesor: DR. ABRAHAM PEREZ ALONZO
Grupo: 4

Ciudad de México, México

Índice

1. Introduccion

Para el proyecto de Estructura y Programacion de Computadoras se implementara un juego de Tetris en Assembly x86 con el fin de mostrar el dominio de sobre la maipulacion de bajo nivel del hardware, la gestion de memoria segmentada y la logica de la programacion estructurada en un entorno sin las abstracciones de los lenguajes de alto nivel

Principalmente se busca implementar las siguientes acciones

1. Bucles
2. Entrada por teclado
3. Manejo del teclado
4. Salida por pantalla
5. Menús en modo texto
6. Operaciones aritméticas binarias y conversión ASCII <- ->binario

Con el juego de tetris se consiguio implementar todos estos puntos En forma general el programa se comporta de la siguiente manera

1. Menu
2. Loop principal del juego

Asi que empezaremos abordando los menus

2. Menu en modo texto

Antes de poner el juego de una implemente un sencillo menu de tres opciones que imprime en pantalla las opciones de

- Jugar
- Instrucciones
- Salir

Figura 1: Menu

Este menu solo acepta los numeros 1 , 2 y 3 si se recibe cualquier otra input no ahce nada

3. Tablero

Pasando al game loop del juego primero debemos manejar el entorno donde vamos a jugar asi que primero dibujaremos el entorno Para ello lo primero que se hice es que pasamos al modo 13h para poder usar colores, lo siguiente es dibujar nuestro tablero para esto dibuje un rectangulo en medio de la pantalla y luego dibuje los bordes de bloques de 1x1, en este caso sera un tablero de 10x16, con cuadros de 12x12 pixeles

La logica para dibujar los rectangulos y bordes son simples loops definidos en este codigo

```
1
2  DibujarTablero PROC
3      MOV AH, 0Ch
4      MOV AL, background_colour
5      MOV DX, play_ground_start_row
6  .LOOP1:
7      MOV CX, play_ground_start_col
8  .LOOP2:
9      INT 10h
10     INC CX
11     CMP CX, play_ground_finish_col
12     JNZ .LOOP2
13     INC DX
14     CMP DX, play_ground_finish_row
15     JNZ .LOOP1
16     RET
17 DibujarTablero ENDP
```

Listing 1: Funcion para el dibujar el tablero

```
1  DibujarBorde PROC
2      MOV BX, play_ground_start_row
3      MOV block_start_row, BX
4      ADD BX, 12
5      MOV block_finish_row, BX
6  .Dibujar:
7      MOV BX, play_ground_start_col
8      MOV block_start_col, BX
9      ADD BX, 12
10     MOV block_finish_col, BX
11     .Dibujar2:
12         CALL Dibujar_Borde_Bloque_Unico
13         ADD block_start_col, 12
14         ADD block_finish_col, 12
15         MOV BX, play_ground_finish_col
16         CMP BX, block_start_col
17         JNZ .Dibujar2
18     ADD block_start_row, 12
```

```
    ADD block_finish_row, 12
    MOV BX, play_ground_finish_row
    CMP BX, block_start_row
    JNZ .Dibujar
    RET
DibujarBorde ENDP
```

Listing 2: Funcion para dibujar los bordes

Como resultado de esto podemos obervar en pantalla lo siguiente

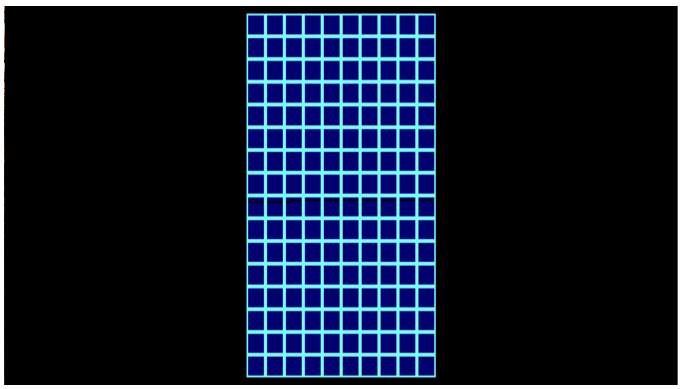

Figura 2: Tablero inicial

4. Puntaje

Luego dibujamos nuestro sistema de puntaje que llevamos para ello lo que se va a hacer es simplemente en una posision marcada poner el puntaje , y luego podemos usar nuestra funcion para actualizar el score, que lo que hace es una conversion binaria a ASCII para actualizar y mostrar nuestro puntaje, ambas funciones se implementaron de la siguiente forma y se ve de la siguiente forma

```
DisplayScore PROC
    MOV AH, 02h
    MOV BH, 00h
    MOV DH, 04h
    MOV DL, 01h
    INT 10h
    MOV AH, 09h
    LEA DX, msg_score
    INT 21h
    RET
DisplayScore ENDP

UpdateScore PROC
    XOR AX, AX
    MOV SI, 9
    MOV AX, score
    MOV BX, 10
.label:
```

```
    CMP SI, 5
    JE .exit_label
    XOR DX, DX
    DIV BX
    ADD DX, 30h
    MOV [msg_score+si], DL
    DEC SI
    JMP .label
.exit_label:
    CALL DisplayScore
    RET
UpdateScore ENDP
```

Listing 3: Funciones para mostrar y actualizar el puntaje

Se ve en nuestro entorno asi

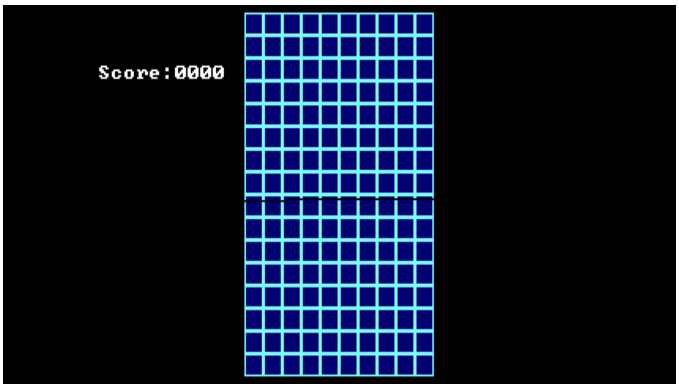

Figura 3: Score dibujado

5. Siguiente pieza

Para terminar con la parte de nuestro entorno inicial necesitamos saber la cola de las piezas siguientes, para ello lo explicare a grandes rasgos lo que hago

1. Generar dos numeros aleatorios entre 0 a 4 y dibujarlos
2. Sacar el primer numero generado para que sea la primer pieza a caer y dibujarla en el tablero
3. Generar aleatoriamente otra pieza (numero de 0 a 4)
4. Actualizar las piezas siguientes

Para esto ya se implementan funciones para dibujar las piezas en nuestro tablero, se ocupan funciones que generen numeros aleatorios. Ahora surgue un problema que pasa si tenemos un estado donde ya no podemos seguir metiendo piezas, es decir, hemos perdido, pues lo que se hace es poner un validador de que la pieza a dibujar no tiene algo que le estorbe, de ser asi se dibuja en otra posicion y si ya no tiene mas espacio para futuro el juego termina. Asi que esto ultimo solo requiere la nueva funcion que valide si la pieza siguiente puede ser dibujada,

y si es asi con instrucciones de JE y JMP se eligue la figura para dibujar, el resultado queda de la siguiente manera

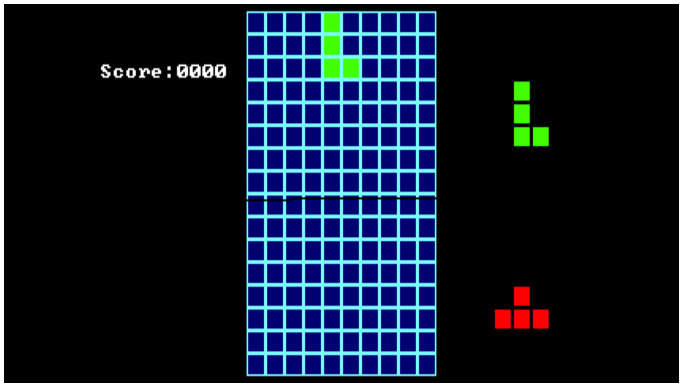

Figura 4: Tablero Final Dibujado

6. Teclado

Ahora por como sirve tetris la pieza empezara a caer cada cierto intervalo de tiempo, por lo tanto ya podemos a empezar a jugar, por lo que necesitamos saber como decirle al programa que queremos hacer, si movernos a la izquierda, a la derecha , bajar un poco, rotar la pieza o tirar la pieza

Lo que hacemos es leer una tecla, si es alguna de las que podemos usar para movernos hacemos la accion que corresponde

- W - Nos permite rotar la pieza
- A - Nos movemos a la izquierda
- S - Bajamos la pieza una casilla
- D - Nos movemos a la derecha
- F - Tirar la pieza

```
LlamarTeclado PROC
    MOV delay_counter, 1
delay_loop3:
    MOV CX, 0FFFFH
    INC delay_counter
delay_loop4:
    CALL Acciones
    LOOP delay_loop4
    CMP delay_counter, 5
    JNZ delay_loop3
    RET
LlamarTeclado ENDP

Acciones PROC
    MOV AH, 01h
    INT 16h
```

```
    JZ arrow_exit

    MOV AH, 00h
    INT 16h

    CMP AL, 'd'
    JE shape_shift_right_label
    CMP AL, 'a'
    JE shape_shift_left_label
    CMP AL, 's'
    JE shape_shift_down_label
    CMP AL, 'w'
    JE shape_rotate_label
    CMP AL, 'f'
    JE fast_shape_shift_down_label

    JMP arrow_exit

shape_shift_right_label:
    CALL shape_shift_right
    JMP arrow_exit

shape_shift_left_label:
    CALL shape_shift_left
    JMP arrow_exit

shape_shift_down_label:
    CALL shape_shift_down
    JMP arrow_exit

shape_rotate_label:
    CALL shape_rotate
    JMP arrow_exit

fast_shape_shift_down_label:
fast_loop:
    CALL shape_shift_down
    CMP successful_magic_shift, 0H
    JE arrow_exit
    JMP fast_loop

arrow_exit:
    RET
Acciones ENDP
```

Listing 4: Implementacion del teclado

7. Puntos

Ya que podemos mover las piezas, saber cual pieza seguir e ir armando nuestro plan de juego debemos saber que hacer cuando una fila se llena y como en el juego original, se borra y se convierten en puntos haciendo que se actualize

nuestro puntaje y que todas las piezas que no esten activas, es decir la que no estamos manipulando colapsen para darnos mas espacio para jugar.

Para poder lograr esto lo que hacemos es checar todas las filas que esten llenas, y checamos eso checando que sean de un color distinto al fondo , si esta llena la borra , aumentamos nuestro score y todo lo que este por encima lo colapsa. es un algoritmo sencillo

8. Conclusiones

Y asi tenemos nuestro jego de tetris, omite las implementaciones de como se dibujan las piezas y como se rotan, ya que el codigo es muy largo, pero se enviaran al profesor y se pueden encontrar en mi repositorio de github, pero con este proyecto se logro completar los objetivos de implementar algo a bajo nivel usando todo lo que en un lenugjae de alto nivel seria mas sencillo pero esto me permitio entender mejor lo que pasa a nivel de hardware cuando programo algo