Pontificia Universidad Javeriana



Andres Eduardo Meneses Rincon

Dary Esmeralda Palacios

Sebastian David Rincon Polo

Samuel Beltran Martinez

Proyecto Final

Arquitectura Del Computador

Bogotá, Colombia

19 de noviembre 2024

Sustentación oral donde se expongan los resultados obtenidos en las diferentes simulaciones.

 El código debe:

Imprimir el menú:  
Seleccione una opción:

1. Leer

2. Escribir

0. Salir

El programa debe esperar una entrada desde el teclado. El usuario puede ingresar un número (1, 2, 0):

1. **Si se ingresa 1**:
   * Debe permitir al usuario **ingresar texto** (hasta 32 caracteres) que se almacenará en el buffer input\_buffer.
   * La entrada debe terminar cuando el usuario presiona la tecla **Enter**.
   * Si el usuario no ingresa nada y presiona Enter, se debe regresar al menú.
2. **Si se ingresa 2**:
   * Debe mostrar el texto que previamente se almacenó en el buffer input\_buffer.
   * Si no hay texto almacenado, puede mostrar un mensaje como "No hay texto guardado".
3. **Si se ingresa 0**:

Debe salir del programa mostrando un mensaje de despedida, por ejemplo: Chao 🙂  
**Si se ingresa cualquier otra tecla**:

* + El programa debe imprimir un mensaje de error, como:  
    Opción inválida. Inténtelo de nuevo.

Código Documentado

org 0x7c00

bits 16

### **Punto de entrada**

start:

jmp boot ; Salta a la rutina principal

1. **jmp boot**: Salta inmediatamente a la etiqueta boot, que contiene la lógica principal del programa.

**Rutina principal**

boot:

cli ; Desactiva las interrupciones

cld ; Limpia el indicador de dirección

mov si, menu ; Carga la dirección del menú en SI

call print\_string ; Llama a `print\_string` para imprimir el menú

1. **cli**: Desactiva las interrupciones del procesador para evitar que el flujo del programa sea interrumpido.
2. **cld**: Limpia el indicador de dirección (Direction Flag), asegurando que las operaciones de cadenas avancen hacia adelante.
3. **mov si, menu**: Carga la dirección de la cadena menu en el registro SI (Source Index), que se usará para acceder a los caracteres.
4. **call print\_string**: Llama a la subrutina print\_string para imprimir el texto almacenado en menu.

### **Lectura y manejo de opciones**

call read\_input

cmp byte [buffer], '1' ; Compara la entrada con '1'

je .read\_option

cmp byte [buffer], '2' ; Compara la entrada con '2'

je .write\_option

jmp boot ; Si no coincide, regresa al menú

1. **call read\_input**: Llama a la subrutina read\_input para leer un carácter ingresado por el usuario.
2. **cmp byte [buffer], '1'**: Compara el contenido de buffer con el carácter '1'.
3. **je .read\_option**: Salta a .read\_option si la comparación es verdadera (el usuario ingresó '1').
4. **cmp byte [buffer], '2'**: Compara el contenido de buffer con el carácter '2'.
5. **je .write\_option**: Salta a .write\_option si la comparación es verdadera (el usuario ingresó '2').
6. **jmp boot**: Si ninguna opción es válida, vuelve al inicio del menú.

### **Opción: Leer texto**

.read\_option:

call read\_text ; Llama a la función `read\_text`

jmp boot ; Vuelve al menú

1. **call read\_text**: Llama a la subrutina read\_text, que permite al usuario escribir hasta 32 caracteres y los guarda en input\_buffer.
2. **jmp boot**: Regresa al menú principal.

**Opción: Mostrar texto**

.write\_option:

call print\_text ; Llama a la función `print\_text`

jmp boot ; Vuelve al menú

1. **call print\_text**: Llama a la subrutina print\_text, que imprime el contenido de input\_buffer.
2. **jmp boot**: Regresa al menú principal.

**Cadenas de texto**

menu db "Seleccione una opción:", 0ah, 0dh

menu\_option1 db "1. Leer", 0ah, 0dh

menu\_option2 db "2. Escribir", 0ah, 0dh

menu\_exit db "0. Salir", 0ah, 0dh, 0

1. Estas son cadenas de texto mostradas al usuario.
   * db: Define una cadena de bytes.
   * 0ah y 0dh: Representan un salto de línea y retorno de carro (equivalentes a \n y \r).
   * El 0 al final marca el fin de la cadena.

**Buffer**

buffer db 0 ; Almacena la opción seleccionada

input\_buffer db 32, 0 ; Espacio para almacenar texto (máximo 32 caracteres)

1. **buffer**: Guarda la opción seleccionada por el usuario (un carácter).
2. **input\_buffer**: Espacio para guardar el texto ingresado por el usuario (máximo 32 caracteres más un terminador 0).

**Subrutina: Leer opción del usuario**

read\_input:

mov ah, 0x00 ; Preparar la función BIOS para leer un carácter

int 0x16 ; Llama a la interrupción 0x16 (teclado)

mov [buffer], al ; Guarda el carácter leído en `buffer`

ret ; Retorna al llamador

1. **mov ah, 0x00**: Configura el registro AH para la función de leer un carácter desde el teclado.
2. **int 0x16**: Llama a la interrupción del BIOS para leer el carácter ingresado por el usuario.
3. **mov [buffer], al**: Guarda el carácter leído en buffer.
4. **ret**: Vuelve al punto desde donde se llamó la subrutina.

**Subrutina: Imprimir una cadena**

print\_string:

mov ax, 0xB800 ; Direccion de la memoria de video

mov es, ax ; ES apunta a la memoria de video

.next\_char:

lodsb ; Carga el siguiente carácter en AL

or al, al ; Verifica si el carácter es 0 (fin de la cadena)

jz .done\_print

mov ah, 0x07 ; Atributo de color (blanco sobre negro)

mov [es:di], ax ; Escribe el carácter en la memoria de video

add di, 2 ; Avanza dos bytes (carácter y atributo)

jmp .next\_char

.done\_print:

ret

1. **mov ax, 0xB800**: Configura AX para apuntar al segmento de memoria de video (modo texto).
2. **mov es, ax**: Configura el segmento ES para acceder a la memoria de video.
3. **lodsb**: Carga el siguiente carácter de SI en AL.
4. **or al, al**: Verifica si el carácter es 0 (NULL).
5. **jz .done\_print**: Si el carácter es 0, termina.
6. **mov ah, 0x07**: Configura el color del texto (blanco sobre negro).
7. **mov [es:di], ax**: Escribe el carácter y el atributo en la memoria de video.
8. **add di, 2**: Avanza al siguiente carácter en pantalla.
9. **ret**: Regresa al llamador.

**Relleno y firma**

times 510 - ($ - $$) db 0 ; Rellena con ceros hasta 510 bytes

dw 0xAA55 ; Firma requerida del sector de arranque

1. **times**: Llena el resto del espacio del sector con ceros para que el tamaño total sea de 512 bytes.
2. **dw 0xAA55**: Agrega la firma estándar del sector de arranque. El BIOS solo ejecutará el código si encuentra esta firma.

**Informe de pruebas:**

**Introducción**

En este informe detallamos el proceso que seguimos para desarrollar un bootloader básico utilizando lenguaje ensamblador, este proyecto tiene como objetivo familiarizarnos con los conceptos clave del arranque del sistema y el uso del modo real de 16 bits. El bootloader que implementamos permite al usuario interactuar con un menú, ingresar texto y visualizarlo en pantalla.

**Desarrollo del Código**

**1. Estructura Básica del Bootloader**

La primera tarea fue establecer la base de nuestro bootloader. Sabemos que este debe comenzar en la dirección de memoria 0x7C00, donde la BIOS carga el primer sector del disco. Definimos esta dirección usando la directiva org 0x7C00. Especificamos que el código debe ejecutarse en modo real de 16 bits con la directiva bits 16.



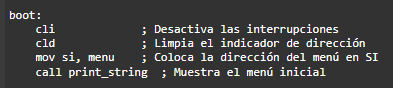
##### **2. Punto de Entrada**

El código comienza con un salto (jmp) a la etiqueta boot, asegurándonos de inicializar correctamente nuestro entorno de ejecución.



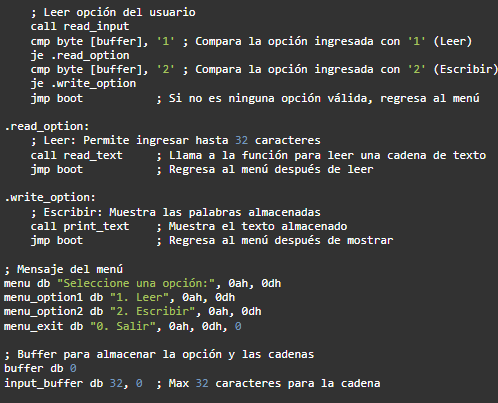
##### **3. Configuración Inicial**

En la rutina boot, desactivamos las interrupciones (cli) y limpiamos el indicador de dirección (cld). Posteriormente, llamamos a la subrutina print\_string para mostrar un menú al usuario.



##### **4. Lectura de Opciones**

Implementamos la subrutina read\_input para leer la opción seleccionada por el usuario. Esto se hace mediante la interrupción int 0x16, que captura la entrada del teclado y almacena el carácter en el buffer buffer.



#### **5. Diseño del Menú de Opciones del Usuario**

### Para comenzar, diseñamos un menú que se muestra al usuario tras la carga del bootloader. Este menú presenta tres opciones principales:

### **Leer:** Permite ingresar texto.

### **Escribir:** Muestra el texto ingresado previamente.

### **Salir:** Aunque se menciona en el menú, esta opción aún no tiene lógica implementada para cerrar el programa.

### El menú está configurado para capturar una opción ingresada por el usuario mediante la subrutina read\_input. Una vez capturada, esta opción es comparada con valores específicos ('1' y '2') para determinar qué acción ejecutar.

#### **6. Opciones del Menú**

El menú ejecuta diferentes subrutinas según la opción seleccionada:

**a) Opción 1: Leer** La subrutina asociada a la opción **Leer** permite al usuario ingresar una cadena de texto de hasta 32 caracteres. Para ello, utilizamos la función read\_text, la cual almacena el texto ingresado en el buffer input\_buffer.

Una vez finalizada la lectura, el control retorna al menú principal, asegurando que el sistema esté listo para recibir nuevas instrucciones.

**b) Opción 2: Escribir** Por otro lado, la subrutina asociada a la opción **Escribir** muestra en pantalla el texto almacenado previamente en el buffer. Para esto, utilizamos la función print\_text.

Esta opción es particularmente útil para verificar la integridad del texto ingresado previamente.

#### **7. Mensajes del Menú**

Los mensajes que se presentan al usuario fueron cuidadosamente diseñados para ser claros y directos. Estos incluyen el texto del menú principal y las descripciones de las opciones disponibles

Con esta estructura, aseguramos que el usuario comprenda las opciones disponibles desde el primer momento.

#### **8. Buffer para Almacenar Datos**

El manejo de la entrada del usuario y la información almacenada se realiza mediante dos buffers:

* **buffer**: Almacena la opción seleccionada por el usuario.
* **input\_buffer**: Almacena el texto ingresado por el usuario, con una longitud máxima de 32 caracteres.



**10. Relleno y Firma del Sector**

Finalmente, rellenamos el archivo binario hasta alcanzar 512 bytes, agregando la firma 0xAA55, requerida para que la BIOS lo reconozca como un sector de arranque válido

#### **9. Implementación y Pruebas**

Durante el desarrollo, ejecutamos nuestro código en un entorno emulado utilizando **QEMU** para garantizar su correcto funcionamiento en modo real. Gracias a estas pruebas, logramos identificar y corregir errores, asegurando que las funcionalidades de lectura y escritura operaran según lo esperado. A pesar de que la opción de "Salir" aún no está completamente implementada, el sistema en su conjunto cumplió con nuestras expectativas iniciales.



