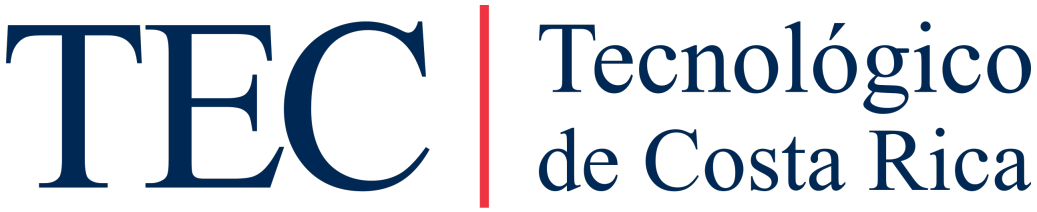
Tarea 1

Instituto Tecnológico de Costa Rica
Centro Académico de Alajuela

IC6600: Principios de Sistemas Operativos

March 19, 2025



Tarea 1
Estudiante:
Marco Alvarez Grijalba
Profesor:
MSc. Ing. Kevin Moraga García
Semestre 1

1 Introducción

Este programa se divide en dos bloques principales: la lectura de sectores por parte de un MBR y la ejecucion de un juego de deletreo, ambas tareas deben ser ejecutadas con un programa de ensamblador NASM.

Un MBR (Master Boot Record) es el primer sector que se lee en cualquier dispositivo de almacenamiento, normalmente tiene un tamaño de 512 bytes, y se puede usar para almacenar una tabla de particiones, para identificar un dispositivo, o, la opcion que se usara en este proyecto, cargar otros sectores de memoria. Cuando se enciende un ordenador la BIOS lee estos primeros 512 bytes y ejecuta en modo real las instrucciones ahi guardades. Para que la BIOS identifique el programa guardado como un MBR los ultimos 2 bytes deben corresponder a la firma de MBR, los cuales son 0x55AA

El modo real BIOS es un modo de 16 bits que se utiliza para iniciar los procesadores x86. Es un modo simplista aunque ralmente es el hardware, no la BIOS, es el que inicia el modo real La BIOS determina qué tipo de procesador son en realidad y cambia al modo apropiado Para poder leer sectores de memoria, el MBR debe tener configurada la direccion del sector que se desea leer y la cantidad de lectores que se desean cargar, una vez hecho esto se utiliza la interrupcion INT 0x13, para luego utilizar la instruccion jmp de ensamblador, para ejecutar el codigo que se requiere
Para la parte del juego, se tiene el siguiente flujo de programa a implementar:

1. El programa genera una serie de 5 caracteres aleatorios, por ejemplo "ajrte"

2. Se guarda e imprime esta cadena de caracteres

3. Se genera y se guarda el deletreo fonetico correcto de manera automatica, por ejemplo "AlphaJuliettRomeoTangoEcho"

4. El programa espera a que el usuario deletree la cadena usando el alfabeto fonetico, esto lo hace ingresando palabra por palabra, por ejemplo

Alpha
Juliett
Romeo
Tango
Echo
Romeo

4.1: cada vez que el usuario ingresa una palabra, el programa guarda esa palabra en un registro mas grande, por medio de un buffer

5. Una vez guardadas las 5 palabras ingresadas por el usuario, se comparar el deletreo original con el deletreo ingresado y se asigna una puntuación de acuerdo a las coincidencias encontradas

2 Ambiente de desarrollo:

VisualStudio Code: Se utilizó el editor de código , el cual facilita la comprensión y lectura del código a la hora de escribir. Se escogio este editor en especifico debido a su simpleza y compatibilidad con multiples sistema operativos.

NASM (Netwide Assembler): como ensamblador para la escritura del programa, este es un ensamblador y desensamblador gratuito para la plataforma Intel x86. Permite la escritura y compilacion de programas, conviertiendo el código fuente .asm en un archivo binario ejecutable .bin. Estos ejecutables pueden ser de arquitecturas de 64, 32 y 16 bits y, este ultimo es el utilizado por la BIOS

para ejecutar programas en modo real.

QEMU: Para la prueba del los diferentes programas se utilizó el emulador y virtualizador de máquina QEMU, el cual permite la simulación de la BIOS sin necesidad de reiniciar el ordenador.

3 Estructuras de datos usadas y funciones:

Las principales funciones para la ejecucion de este programa son las siguientes:

3.1 En mbr.asm:

El código configura primero la ubicación de memoria para cargar el programa, asignando el segmento ES a 0x7E0 y el offset BX a 0, lo que equivale a la dirección 0x7E00 en memoria. A continuación, se prepara la llamada a la BIOS mediante INT 13h para leer tres sectores del disco duro, comenzando en el cilindro 0, cabeza 0 y sector 2, utilizando DL=0x80 para seleccionar el disco. Los datos leídos se almacenan en la dirección previamente configurada; si ocurre algún error durante la lectura (detectado por el Carry Flag), se redirige a una rutina de error, pero si la operación es exitosa, se transfiere la ejecución al programa cargado mediante un salto a 0x7E0:0000.

3.2 En mrpv.asm:

copy_word: Esta rutina copia una palabra (una cadena de caracteres terminada en 0) desde un origen hasta un destino. Primero se asegura que la dirección de copia se incremente de forma normal, mediante el comando 'cld'. Luego, en un bucle, se carga un byte del origen apuntado por BX, y se compara con 0 para determinar si se llegó al final de la cadena; si no es el final, el byte se copia al destino, apuntado por DI, se incrementan ambos punteros y se repite el proceso hasta encontrar el carácter nulo, momento en el cual la rutina finaliza.

read_string: Esta rutina lee una cadena de caracteres desde el teclado y la almacena en un buffer cuyo inicio debe estar apuntado por DI. En cada iteración, la rutina espera a que se presione una tecla usando INT 16h; si la tecla presionada no es ENTER (0x0D), se muestra el carácter en pantalla mediante INT 10h y se guarda en el buffer, avanzando DI para la siguiente posición. El proceso se repite hasta que se presiona ENTER, momento en el cual se coloca un byte nulo al final del buffer para marcar la terminación de la cadena.

add_points, add_tens, add_cents y max_score: Estas rutinas gestionan el sistema de puntuación que utiliza tres dígitos representados en ASCII, se decidio utilizar este metodo debido a las complicaciones que se presentaron al utilizar numeros a la hora de leer, imprimir y sumar numeros. La rutina 'add_points' toma el dígito de unidades, lo convierte de ASCII a su valor numérico y, si es menor que 9, lo incrementa y lo vuelve a convertir a ASCII; si ya es 9, llama a 'add_tens' para reiniciar las unidades y aumentar el dígito de decenas. De forma similar, 'add_tens' reinicia las unidades y, si el dígito de decenas es 9, delega la operación a 'add_cents' para incrementar el dígito de centenas; si éste alcanza también el valor máximo (9), se considera que se ha llegado al puntaje máximo y se invoca 'max_score', que muestra un mensaje final usando 'print_string'.Para este caso, ase decidio utilazr una puntuacion maxima posible de 999, ya que se considero que durante la ejecucion del juego no es factible sobrepasar este numero.

get_random_word: Esta rutina genera una letra aleatoria tomando como base una semilla. Primeroinvoca a 'get_letter', que genera un valor pseudoaleatorio; a ese valor se le aplica una operación AND con 25 para limitarlo al rango de 0 a 25 (correspondiente a las 26 letras del alfabeto), y posteriormente se le suma el valor ASCII de 'a' para convertir el número en una letra minúscula (de 'a' a 'z'). Finalmente, la letra resultante se almacena en un buffer específico.

get_letter: Esta rutina implementa un generador de números aleatorios utilizando un método de congruencia lineal. Toma el valor actual de la semilla (almacenada en memoria en la etiqueta 'seed') y le suma el valor de 'points_rdn', almacenando el resultado en AX. Luego, multiplica AX por la constante 48271 y, tras limpiar DX, le suma 12345 para alterar el resultado; se utiliza la instrucción ADC para

incluir el acarreo. El nuevo valor resultante se guarda nuevamente en la variable ‘seed‘, de modo que para la próxima llamada se disponga de una semilla actualizada, y la rutina retorna dejando el resultado en AX.

compare_letters: Esta rutina compara letra por letra dos cadenas: una que representa el deletreo fonético original (apuntada por SI) y otra que ha sido ingresada por el usuario (apuntada por DI). En cada iteración, carga la letra actual de ambas cadenas y, si encuentra un carácter nulo en la cadena original, salta a la rutina para mostrar la puntuación, ya que indica el final de la palabra. Si aún hay letras, llama a ‘update_score‘ para ajustar el puntaje según si las letras coinciden o no, y luego incrementa ambos punteros (SI y DI) para pasar a la siguiente letra, repitiendo el proceso hasta terminar.

update_score: Esta rutina se encarga de actualizar la puntuación en función de la comparación de letras. Comienza cargando el valor actual de ‘points_rdn‘ y luego compara la letra del deletreo original (en AL) con la letra ingresada (en BL). Si ambas letras son iguales, salta a la etiqueta ‘.correct‘ (presumiblemente para incrementar la puntuación). Además, si la letra original es un asterisco (‘’*‘‘), salta a la etiqueta ‘.incorrect_star‘ para tratar ese caso especial. Si ninguna de estas condiciones se cumple, la rutina simplemente retorna sin modificar la puntuación.

4 Instrucciones para ejecutar el programa:

Compilar ambos programas:
nasm -f bin mbr.asm -o mbr.bin
nasm -f bin mrpv.asm -o mrpv.bin

Copiar programas a la usb:
sudo dd if=mbr.bin of=/dev/sdb bs=512 count=1
sudo dd if=mrpv.bin of=/dev/sdb bs=512 seek=1 count=5

5 Actividades realizadas por estudiante:

Fecha	Actividad	Horas	Descripción
05/03/2025	Investigación sobre bootloaders y carga de programas desde BIOS	3	Revisión de documentación y pruebas iniciales.
06/03/2025	Configuración del entorno (NASM, QEMU, USB booteable)	4	Instalación y pruebas de compilación de código.
07/03/2025	Desarrollo del bootloader básico	3	Escritura y depuración del código de arranque.
08/03/2025	Implementación de impresión en pantalla con BIOS	3	Uso de `int 10h` para mostrar texto en modo real.
09/03/2025	Lectura de entrada del teclado	3	Captura de teclas con `int 16h` y pruebas en QEMU.
10/03/2025	Diseño del juego	4	Planificación de la lógica del juego y flujo de ejecución.
11/03/2025	Codificación del juego	3	Implementación inicial de la lógica de juego en NASM.
12/03/2025	Optimización de código y depuración	4	Corrección de errores y mejora de flujo del programa.
13/03/2025	Implementación de generación aleatoria desde BIOS	5	Uso de `int 1Ah` para obtener valores pseudoaleatorios. Intento fallido
14/03/2025	Pruebas y ajustes del código del bootloader	4	Verificación de carga de sectores y ejecución del juego.

15/03/2025	Depuración de errores en lectura de disco	5	Diagnóstico y solución de errores en `int 13h`.
17/03/2025	Pruebas en hardware real	2	Evaluación en una USB booteable en diferentes equipos.
17-03-2025	Generación de caracteres aleatorios	3	Creacion de la funcion para generar caracteres aleatorios correctamente
17-03-2025	Almacenamiento de palabras en un buffer	2	Estos buffers serviran para el almacenamiento de lo ingresado por el usuario
17-03-2025	Implementación de comparación entre input y cadena esperada	3	Por medio de una funcion de comparacion creada
18-03-2025	Implementación de una rutina de puntuación	3	Creacion del proceos de puntuacion necesario
Total		**54**	

6 Autoevaluación:

En su version final, el proyecto cumple con todos los requerimientos solicitados en la especificacion, sin embargo se encontraron multiples problemas, algunos de los cuales no se les pudo dar una solucion adecuada:

1. Para la generacion de letras aleatorias, no fue posible crear una funcion que generara numeros totalmente aleatorios, debido a que no se encontro una implementacion adecuada de las formas ya existentes(utilizar el reloj interno o el contador de tecleo como semilla) debido a esto se imple,ento una rutina de generacion pseudo aleatoria, la cual se alimenta de la misma ejecucion del programa para generar numero de manera no tan predecible

2. Durante la implementación de la función de puntuación, no fue posible encontrar una manera de imprimir directamente los números almacenados dentro del programa, por lo que se opto por almacenar dicha puntuación como caracteres modificables. Este proceso de modificación también se implemento dentro del programa

3. En la fase final, debido a la implementación del juego en cuanto a la comparacion del deletreo original con el deletreo ingresado, si se ingresas palabras de un tamaño diferente a las guardadas, aunque las palabras siguientes esten correctas, es muy posible que estas no sumen puntos a la puntuacion final.

6.1 Rúbrica de la sección Autoevaluación.

Rubro	**Porcentaje total**	**Porcentaje obtenido**
Sector de arranque	30	30
Fonético	50	40
Documentación	20	20
Total de Porcentaje	100	**90**

6.2 Reporte de commits Asociados al programa.

7 Lecciones Aprendidas del proyecto:

Lo principal en este tipo de proyectos es buscar avances concretos, no atacar todo el problema en un conjunto si no utilizar el conocido divide y vencerás, siempre buscando tener avances significativos.

Para este proyecto considere que lo primero debía ser la creación del mbr.bin con el cual ejecutar el programa, después de eso pensé en desarrollar la función de generación de números aleatorios, sin embargo tuve muchos problemas con este proceso, por lo que decidí dejar esto para lo ultimo, ya que a mi parecer es mejor asegurarse de entregar algo útil que funcione aunque sea parcialmente, que no entregar nada por haber invertido demasiado tiempo en resolver un problema.

De igual manera, la investigación y comprensión de los temas necesarios para realizar los proyectos son un pilar fundamental. Personalmente se me dificulta la utilización de inteligencias artificiales, por que si utilizo un código generado totalmente por una inteligencia artificial es mas complicado la detección y corrección de errores, tomando en cuenta que esta IA no son infalibles en sus respuetas.

Es por esto que a mi parecer, es mejor investigar todo lo posible acerca del problema a resolver y en caso de ser necesario, utilizar las inteligencias artificiales como asistentes en tareas de menor importancia.

8 Bibliografía

Foro acerca de desarrollo de sistemas operativos:
https://wiki.osdev.org/Expanded_Main_Page

Pagina oficial de NASM:
https://www.nasm.us/

Pagina de referencias de registros de NASM:
https://www.cs.uaf.edu/2017/fall/cs301/reference/x86_64.html

Tutotrial para principiantes de NASM
https://cs.lmu.edu/ ray/notes/nasmtutorial/