
Instituto Tecnológico de Costa Rica
Campus Tecnológico Central Cartago
Escuela de Ingeniería en Computación

Curso: IC-3101 Arquitectura de Computadoras

# Actividad: Tarea # 4 - Trivia NASM

Estudiantes:
Juan Andrés Bastidas López - 2025066242
Steadman Murillo Parrales - 2025097768

Fecha de entrega: 10 de Noviembre de 2025, segundo periodo

Profesor: M.Sc. Esteban Arias Méndez

As the submission of the first project in the course, this project presents the development of a trivia game implemented entirely in Assembly language. The program allows users to answer multiple-choice questions stored in a data structure, each associated with a specific score value. When the player selects an answer, the system validates it, provides feedback, and updates the total score accordingly.

The project demonstrates the use of fundamental Assembly concepts such as memory management, file handling, loops, conditionals, and user input/output while maintaining efficiency and understanding of computer architecture and processor-level operations.

## Índice

## I. Introducción

El presente trabajo es la entrega del primer proyecto del curso de Arquitectura de Computadores, por el profesor M.Sc Esteban Arias Mendez. Este proyecto consiste en el desarrollo de una trivia utilizando el ensamblador NASM de 32 bits, en la arquitectura i382, como lenguaje principal de este proyecto.

El juego de trivia consiste de más de 30 preguntas de cultura general, combinando dos tipos de preguntas, verdadero y falso, y 4 opciones. Cada una de estas preguntas posee un valor arraigado de manera arbitraria a estas, de 1 a 3 puntos.

Al iniciar el programa, se iniciará una ronda de 10 preguntas aleatorias de la estructura de datos del programa, así asegurando una experiencia distinta en cada ronda. Tras responderlas, al usuario se le dará el puntaje y la cantidad de preguntas respondidas correctamente. Posterior a ello, el usuario podrá repetir la ronda; los puntajes y cantidad de respuestas correctas serán acumuladas hasta que el usuario decida terminar el juego.

Para la resolución se usó la guía oficial de NASM, además de material de apoyo e instrucciones del libro de Sivarama además de fuentes en línea. Por otro lado, para la resolución de problemas, se utilizaron dos equipos con sistemas operativos Linux, pero con distintas versiones (Mint y Arch), ayudándonos de la aplicación de notas gedit [2] y el programa Visual Studio Code [3]. En cuanto al código, se priorizó el uso de múltiples arreglos e índices para el almacenamiento de todos los datos que fueren necesarios del archivo de preguntas, tales como las preguntas, las opciones y las respuestas.

## II. Marco Teórico de Referencia

### 1. ¿Qué es el lenguaje ensamblador?

El lenguaje ensamblador constituye el nivel más bajo de programación legible por humanos antes del código máquina. Cada instrucción en ensamblador se traduce directamente en una instrucción del procesador, permitiendo un control preciso sobre el hardware. En la arquitectura x86, ampliamente utilizada desde los microprocesadores Intel 386, el modo de 32 bits introdujo registros extendidos (como EAX, EBX, ECX, EDX) y la capacidad de direccionar mayores espacios de memoria. [4]

El lenguaje ensamblador se relaciona estrechamente con el procesador, debido a que es traducido de forma 1 a 1 al lenguaje máquina. Por otro lado, el lenguaje ensamblador es dependiente del procesador, es por ello, que hay diferentes lenguajes a lo largo del mundo. Por otro lado, el lenguaje ensamblador posee ventajas sobre los de alto nivel, como puede ser su velocidad de procesamiento y eficiencia a la hora del espacio que ocupa un programa. [4]

### 2. Fundamentos de funcionalidades del proyecto

Este apartado se hace para explicar las referencias y explicaciones de aquellos códigos que fueron empleados de fuentes externas dentro del proyecto. En este proyecto fueron usados solamente 4 códigos. Tres de ellos fueron empleados de la fuente [4] para el manejo de archivos. El último de la fuente [5] fue para el desarrollo de un número pseudoaleatorio.

**Manejo de Archivos:**

En lenguaje Ensamblador, se requiere el acceso a archivos además de la Entrada y Salida de Datos. Se considera pertinente evaluar los códigos empleados que estaban (previo al proyecto e investigación) desconocidos para su correcta aplicación y uso.

**System call 5** — Open a file

| | | |
|---|---|---|
| Inputs: | EAX | = 5 |
| | EBX | = file name |
| | ECX | = file access mode |
| | EDX | = file permissions |
| Returns: | EAX | = file descriptor |
| Error: | EAX | = error code |

*Fig. 1 Adaptado de fuente [4]*

De este código se aprendió la apertura de un archivo, de esta forma se accedió a la estructura de datos necesaria para el programa.

**System call 3** — Read from a file

| | | |
|---|---|---|
| Inputs: | EAX | = 3 |
| | EBX | = file descriptor |
| | ECX | = pointer to input buffer |
| | EDX | = buffer size<br>(maximum number of bytes to read) |
| Returns: | EAX | = number of bytes read |
| Error: | EAX | = error code |

*Fig. 2 Adaptado de fuente [4]*

De este código se aprendió la lectura de un archivo, de esta forma se accedió a la estructura de datos necesaria para el programa, y se interpretaron los datos, además de recorrer el archivo de una forma adecuada.

**System call 6** — Close a file

| | | |
|---|---|---|
| Inputs: | EAX | = 6 |
| | EBX | = file descriptor |
| Returns: | EAX | = — |
| Error: | EAX | = error code |

*Fig. 3 Adaptado de fuente [4]*

Mediante este código, se aprendió sobre el cierre de un archivo, útil para una correcta finalización del programa.

**Numeros PseudoAleatorios:**
Para el programa se requiere acceder a las preguntas de forma aleatoria, por ello, se creó un proceso que daba un índice aleatorio, por el cual, se podía acceder a preguntas de forma aleatoria. Para su uso en la trivia.

```
rdtsc    ; counter → EDX:EAX
mov bx, 6
div  bx  ; DX ← AX % 6
inc  dx  ; number can be 0, so sart from 1

Now DX contans number from 1 to 6.
```

*Fig. 4 Adaptado de fuente [5]*

De este código surgió la idea del uso del reloj interno del procesador, para luego obtener un valor numérico en base al residuo de una división, para luego tomar este valor como un índice y traducirlo a una respuesta.

## III. Planteamiento del Problema

Según los requerimientos del enunciado para el presente proyecto. Se debe poder acceder a un archivo donde se posean las características de las preguntas, sus opciones, enunciados, respuesta y valor. Por otro lado, también se deben poder preguntar al usuario, deben ser aleatorias y finalmente se debe llevar un conteo del puntaje del usuario.

Esto implica acceso e interpretación del archivo, además de una interacción mediante el terminal por el usuario. Es sabiendo esto, que se decidió primero hacer la interpretación de la información del archivo y tras de ello, guardarlo en diferentes arreglos, para su posterior acceso y uso para la interacción con el usuario.

El programa entero se puede dividir en dos partes, la primera que es la interpretación del archivo que consta de los procedimientos necesarios para guardar todos los aspectos de las preguntas en sus respectivos arreglos, esto fue logrado mediante una estructura por pregunta en bloques y un conteo de caracteres 'enter'. Tras guardar las preguntas en sus respectivos arreglos, sigue la segunda parte del programa, que es la interacción con el usuario, que accede a todo lo guardado en los arreglos y lo emplea para una experiencia adecuada.

## IV. Desarrollo de contenidos

Para completar este ejercicio, tenemos gran cantidad de elementos que se deben tomar en cuenta. Principalmente, se tiene que tomar un enfoque para desarrollar un programa amigable con el usuario, y esto se logra por medio de un buen orden durante la programación.

Inicialmente, y como parte de nuestro patrón de diseño, nos dedicamos a asociar medidas importantes a palabras claves a través de *EQUs*. Esto no solo es una práctica que facilita la programación al usar palabras clave, sino que permite hacer modificaciones de manera sencilla y escalar nuestra implementación, si así se quisiera.

```
bufsize EQU 300   :  Máxima cantidad de bytes por pregunta
tamannio_pregunta EQU 80  ; Máximo tamaño que puede tener una pregunta
tamannio_respuesta EQU 2  ; Máximo tamaño que puede tener una respuesta
tamannio_opcion EQU 45  ; Maximo tamaño que puede tener una opción
tamannio_totalidad_pregunta EQU 7  ; Cantidad de lineas que tiene cada pregunta en el archivo (pregunta + 4 opciones + respuesta + puntaje)

bloque_opciones EQU tamannio_opcion*4  ; Máximo de tamaño que puede tener un bloque de 4 opciones
cantidad_preguntas EQU 10  ; Cantida de preguntas que se le muestran al usuario en una "run"
```

*Fig. 5. Se establecen EQUs para mejorar la compresión del código*

Posteriormente, pasamos al bloque de `.DATA`. Debemos poner en esta sección todos los mensajes que se le muestran al usuario en los distintos casos a los que se puede llegar. Esto involucra indicaciones de uso del programa, mensajes sobre el estado de su partida e incluso mensajes de error a la hora de leer el archivo de preguntas

```
.DATA
filename db "PreguntasTrivia.txt",0 ; nombre del archivo
msg_bienvenida db "¡Bienvenido a nuestro juego de Trivia! Se te asignarán 10 preguntas aleatorias de nuestra base de datos, y deberás responder con la letra que contenga la opción correcta. ¡Buena suerte!", 0
; Mensajes de comprobación y estado
msg_correcto db "¡CORRECTO!", 0  ; Mensaje para respuestas correctas
msg_incorrecto db "¡INCORRECTO! La respuesta correcta era la opción ", 0  ; Mensaje para respuestas incorrectas
; Mensajes para mostrar el score


msg_indicar_respuesta db "Su respuesta a esta pregunta es: ", 0
msg_puntaje_1 db "¡Has acumulado un total de ", 0
msg_puntaje_2 db " punto(s)!", 0
msg_error db "Error leyendo archivo: intente nuevamente", 0
msg_volver_jugar db "¿Desea volver a jugar? (S/N): ", 0
msg_valor_pregunta_1 db "Esta pregunta vale por ", 0
msg_valor_pregunta_2 db " punto(s)", 0
```

*Fig. 6. Inicialización de mensajes en .DATA*

Para los datos no inicializados, se tomó una de las decisiones más importantes de diseño: crear arreglos para cada tipo de información extraída del archivo, entre ellas las preguntas,

las opciones y las respuestas asignadas. Esta construcción está estrechamente conectada a las definiciones de medidas establecidas en un inicio del código, contribuyendo también a que este sea escalable. Se reservan además espacios con un consumo menos significativo, como puede ser el índice de la pregunta y el contador de ENTERS para moverse en el archivo leído.

```
.UDATA
contador_enter resd 1  ; Cuenta la cantidad de Enters para así ver las preguntas correctamente
buffer resb bufsize ; Buffer como mediador para guardar todo a sus arreglos
buftrash resb 1    ; Buffer para guardar un valor hasta saltar x lineas

arreglo_preguntas resb tamannio_pregunta*cantidad_preguntas ;reserva 10 preguntas
arreglo_opciones resb bloque_opciones*cantidad_preguntas ;las 4 opciones de las 10 preguntas
arreglo_respuestas resb tamannio_respuesta*cantidad_preguntas ;las respuestas de las 10 preguntas
arreglo_puntaje resb tamannio_respuesta*cantidad_preguntas  ; Arreglo que guarda el puntaje de cada pregunta

n_pregunta resd 1  ; Pregunta actual (índice dentro de casi TODOS los arreglos, exceptuando arreglo_num_elegidos)

arreglo_num_elegidos resd cantidad_preguntas  ; Arreglo que guarda los números elegidos para ser presentados
n_repetidos resd 1  ;  Cantidad que guarda la cantidad de numeros en el arreglo (se guarda en un DWORD por comodidad para trabajar con ESI y EDI posteriormente)
```

*Fig. 7. Reserva de datos para arreglos y variables en .UDATA*

Nuestro primer paso dentro del código es la lectura del archivo. Para leerlo, utilizamos un system call para la apertura de archivos en modo lectura, tomada directamente del libro de Silvarama [4]. Con una configuración específica de los registros generales y el llamado a una interrupción es que podemos abrir un archivo de forma exitosa

```
abrir_archivo:  ; Proceso para abrir nuestro archivo de preguntas
    mov eax, 5  ; sys_open
    mov ebx, filename  ; Nombre del archivo
    mov ecx, 0  ; Modo: ReadOnly
    int 0x80  ; Se llama a la interrupción
    cmp eax, 0  ; El resultado fue guardado en EAX: si este dió 0, hubo un error
    js  error  ; Si error, salir
    mov ebx, eax  ; Descriptor del archivo es guardado en EBX
```

*Fig. 8. Apertura del archivo al inciar el código*

Una vez con el archivo abierto, se procede a elegir qué preguntas del archivo vamos a tomar. El proceso se hace a través de una etiqueta `randint`, la cual genera números pseudoaleatorios que van desde el 0 (representando la pregunta 1) al 34 (siendo la pregunta 35). Esto se logra gracias a la instrucción `rdtsc` (que toma el contador de ciclos de reloj del procesador desde el arranque del sistema), la cual es acompañada por rotaciones e instrucciones XOR para generar aún más desorden.

El número posterior a ello es dividido por 35, para que así su residuo sea el usado para el número aleatorio, pues el residuo va de 0 a 34.

```
randint:  ; Rutina para volver a llamar randint (numero entre 0 y 29) e iniciar todo otra vez
    mov DWORD [contador_enter], 0
    push ebx   ; Push necesario para mantener el descriptor en EBX

    rdtsc  ; Se toma el contador de ciclos de reloj del procesador desde el arranque del sistema
    xor edx, eax  ; Desorden
    rol edx, 13  ; Desorden
    xor eax, edx  ; Desorden
    mov ebx, 35  ; Número maximo
    xor edx, edx  ; Desorden
    div ebx  ; Divide y el número generado obtenido es pseudo-aleatorio (residuo se encuentra en EDX)

    pop ebx  ; Se recupera nuestro registro EBX
```

*Fig. 9. Etiqueta randint para elegir números aleatorios*

Al generar un número, se revisa si no está en un arreglo de preguntas selectas que se maneja: si lo está, mandamos a elegir otro número. En el caso de que el número no pertenezca al arreglo, se procede a añadir al arreglo.

```
    mov esi, 0  ; Indice para recorrer arreglo_repetidos

verificar_repetido:  ; Etiqueta para verificar si el número obtenido del randint está repetido
    cmp esi, DWORD [n_repetidos]  ; Comparamos con DWORD[n_repetidos] para ver si llegamos al siguiente espacio vacio (DWORD[n_repetidos] es un número)
    je guardar_numero  ; Si llegamos al final, no está repetido

    mov eax, [arreglo_num_elegidos + esi*4] ; Mientras no se haya llegado al final de los números obtenidos, se lee el siguiente número guardado
    cmp eax, edx  ; Se compara con el obtenido por nuestro randint
    je repetir_randint  ; Si coincide, significa que el número ya está y debemos repetir el randint
    inc esi  ; Si no son iguales, procedemos a incrementar esi y revisar el siguiente dígito
    jmp verificar_repetido  ; Saltamos a la misma etiqueta en la que nos encontramos

guardar_numero:  ; Etiqueta a la que llegamos si el número que obtuvimos no está repetido:
    mov eax, DWORD [n_repetidos]  ; Ponemos en EAX la cantidad de números que habiamos contabilizado antes
    mov [arreglo_num_elegidos + eax*4], edx ; A la siguiente casilla libre se le asigna EDX (resultado de randint)
    inc DWORD [n_repetidos]  ; Incrementamos la cantidad de números guardados para apuntar al siguiente espacio vacio
    ; Se continua con el flujo normal (saltar_loop, etc.)
    jmp no_repetido

repetir_randint:  ; Etiqueta a la que llegamos cuando el número obtenido estaba repetido
    jmp randint  ; Se repite el randint

no_repetido:  ; Etiqueta a la que llegamos cuando nuestro número no fue repetido
    imul edx, tamannio_totalidad_pregunta  ; Se multiplica nuestro resultado de randint por el tamaño de una pregunta
    mov DWORD [contador_enter], edx  ;  Se guarda el valor de los enters a saltar en el contador
```

*Fig. 10. Secuencia para seleccionar una pregunta no repetida*

Ya con nuestro número único, y como se puede ver abajo de la Fig. 10. con la etiqueta `no_repetido`, procedemos a contar cuántos ENTERS debemos pasar para llegar a la pregunta que seleccionamos. Es necesario entonces que hagamos una secuencia de recorrer el archivo, contando cuantos saltos de línea llevamos.

```
; === Leer hasta x enters, pues es el final de la pregunta ===
saltar_loop:  ; Etiqueta encargada de posicionarnos dentro del archivo, haciendo que apuntemos a la pregunta que necesitamos
    mov eax, 3  ; sys_read
    mov ecx, buftrash  ; Dirección del buffer temporal de 1 byte
    mov edx, 1  ; Leemos 1 byte
    int 0x80  ; Procedemos con la interrupción
    cmp eax, 0  ; El resultado fue guardado en EAX: si este dió 0, hubo un error
    jle error  ; EOF o error

    mov al, [buftrash]  ; Cargar el byte leído
    cmp al, 0x0A  ; ¿El byte es un enter?
    jne saltar_loop  ; Si no es un enter, procedemos a leer siguiente

    ; si es enter
    dec DWORD [contador_enter]  ; Decrementamos el contador de enters recorridos
    cmp DWORD [contador_enter], 0  ; Si este llega a su fin, procedemos a leer el archivo
    je leer_archivo  ; Etiqueta para leer el archivo
    jmp saltar_loop  ; Si no se ha llegado al final, se repite otra vez el loop
```

*Fig. 11. Secuencia para ubicarse en la pregunta elegida*

Ya apuntando a nuestra pregunta, procedemos a leerla. Aprovechando el gran espacio de buffer que reservamos, la leemos en su totalidad y luego procedemos a ir fragmentando su contenido

```
;== Leer archivo ==
;Tras saltar la cantidad de enters que dice el archivo ahora si leemos, iniciando en la pregunta que obtubimos

leer_archivo:  ; Etiqueta para la lectura del archivo
    mov eax, 3  ; sys_read
    mov ecx, buffer  ; Buffer donde se guarda la lectura
    mov edx, bufsize  ; Cantidad máxima a leer
    int 0x80  ; Se hace la interrupción
    cmp eax, 0  ; El resultado fue guardado en EAX: si este dió 0, hubo un error
    jle error  ; EOF o error

    ; En este momento de la ejecución el buffer tiene la pregunta que queriamos, entonces procedemos a pasarla a su respectivo espacio
    ; Preparativos para copiar pregunta
    mov ecx, DWORD [n_pregunta]  ; Se pone en ECX el número de pregunta en el que estamos actualmente (indice del arreglo de preguntas)
    lea ESI, buffer  ; Con lea, cargamos la dirección del buffer en ESI
    mov EAX, ECX  ; Ponemos en EAX el número de pregunta en el que estamos
    imul EAX, tamannio_pregunta  ; Multiplicamos el número de pregunta por el tamaño que tiene una pregunta (esta operación nos termina dando el offset del siguiente espacio vacio disponible dentro del arreglo de preguntas)
    lea EDI, [arreglo_preguntas + EAX]  ; Ponemos a EDI a apuntar a este espacio vacio
```

*Fig. 12. Lectura de la totalidad de la pregunta, guardada en un buffer*

Recorrer nuestro espacio de buffer nos permite sacar todos los elementos que construyen una pregunta, ordenados de la siguiente manera:

- Pregunta
- Opciones (4 en total, desde la A a la D)
- Respuesta correcta
- Puntaje sumado al responder correctamente

El proceso requiere de mucho orden, tanto al codificar como se ejecuta la lectura como al redactar el contenido el el archivo de preguntas.

```
"¿Cuántos huesos tiene el cuerpo humano?"
"A. 184"
"B. 206"
"C. 231"
"D. 191"
B
1
```

*Fig. 13. Ejemplo de preguntas dentro del .txt que las reagrupa*

Empezamos entonces por sacar la pregunta. Sabiendo que estamos ubicados al inicio del buffer, solo debemos leer y copiar en nuestro arreglo de preguntas hasta toparnos un cambio de línea. Esto se hace por medio de los registros ESI y EDI, que apuntan a espacios de memoria diferentes (al buffer que almacena nuestra lectura y al arreglo revisado, respectivamente)

```
; EDI = Apuntando al inicio del espacio vacío de arreglo_preguntas
; ESI = Apuntando al inicio del buffer que tiene el texto que queremos
guardar_pregunta:  ; Etiqueta para guardar las cadenas necesarias
    mov al,[ESI]  ; Mueve el byte revisado del buffer a AL
    mov [EDI], al  ; Lo copia en el arreglo
    inc ESI  ; Siguiente del buffer
    inc EDI  ; Siguiente del arreglo
    cmp al, 0x0A  ; Se compara con enter (indica el final de la pregunta)

    je guardar_opciones  ; Si se llegó al final, pasamos a guardar las opciones
    jmp guardar_pregunta  ; Si no ha llegado al enter, se sigue leyendo y copiando del buffer
```

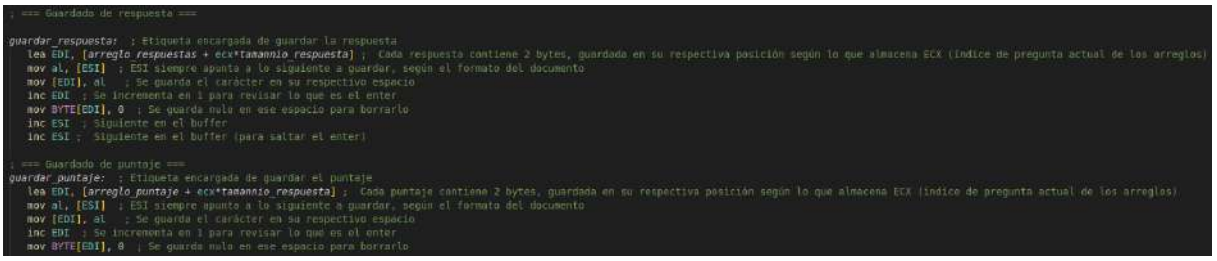
*Fig. 14. Secuencia para copiar la pregunta en su arreglo respectivo*

La misma secuencia es empleada para obtener las opciones, solo que debemos sumar el componente extra de los saltos para registrar cada una de ellas. Esto nos obliga mantener una secuencia de copiado correcta y una comprobación de 4 ENTERS al final para poder pasar al guardado de la respuesta.

```
;=== Guardar opciones de la respectiva pregunta ===

guardar_opciones:  ; Preparativos para guardar las opciones de la pregunta
   mov DWORD[EDI-1], 0  ; Se almacena un nulo al final de la pregunta (sirve para eliminar el carácter del enter)
   mov BYTE[contador_enter], 0 ; Se reinicia el índice de opción a 0

   ; Calcular dirección inicial de la primera opción:
   mov eax, ecx  ; Se pone en EAX el contenido de ECX (recordando que ECX tiene el número de pregunta en el que estamos actualmente)
   imul eax, bloque_opciones  ; 180 = 45*4 (tamaño bloque por pregunta)
   lea edi, [arreglo_opciones + eax]  ; Así como se hizo con las preguntas, ponemos a EDI a apuntar al siguiente espacio vacío donde escribiremos las opciones que obtuvimos


ciclo_opciones:  ; Etiqueta del ciclo de lectura de opciones
   mov al,[ESI]  ; Mueve el byte revisado del buffer a AL
   mov [EDI], al  ; Lo copia en el arreglo
   inc ESI  ; Siguiente del buffer
   inc EDI  ; Siguiente del arreglo

   cmp al, 0x0A  ; Si hay un enter, debemos recalcular y seguir con la siguiente opción de pregunta
   je siguiente_opcion  ; Se llegó a un enter, entonces se pasa a la siguiente opción
   jmp ciclo_opciones  ; Si no, seguimos leyendo normal


siguiente_opcion:  ; Etiqueta para el paso entre opciones

; === Recálculo de posición y ajuste de opción ===
   mov BYTE[EDI-1], 0  ; Se reemplaza el enter con un 0
   inc DWORD [contador_enter]  ; Siguiente opcion (ajuste de opción al incrementar el contador de enters que llevamos)

   ; Cálculo de nueva posición para la siguiente opción
    mov eax, ecx  ; Se pone en EAX el contenido de ECX (número de pregunta en el que estamos actualmente)
    imul eax, bloque_opciones  ; 180 = 45*4 (tamaño bloque por pregunta)
    mov ebx, DWORD [contador_enter]  ; Ponemos en EBX el contador de enters que llevamos (número entre 0 y 3)
    imul ebx, tamannio_opcion  ; Este número guardado en EBX es multiplicado por el tamaño que consume cada opción, asegurando que el espacio que registramos es el indicado
    add eax, ebx  ; Le sumamos EBX a nuestro offset, almacenado en EAX
    lea edi, [arreglo_opciones + eax]  ; Ponemos a EDI a apuntar a ese espacio recién calculado

; === Se compara con 4: si es el caso, se terminó de guardar ===
   cmp DWORD [contador_enter], 4  ; Si el contador de enters llegó a 4, significa que leímos todas las opciones y pasamos la guardado de respuesta
   je guardar_respuesta  ; Paso al guardado de respuesta
   jmp ciclo_opciones  ; Si no se llegó a los 4 enters, se siguen guardando opciones en su respectivo arreglo
```

*Fig. 15. Secuencia para el copiado de las opciones de una pregunta*

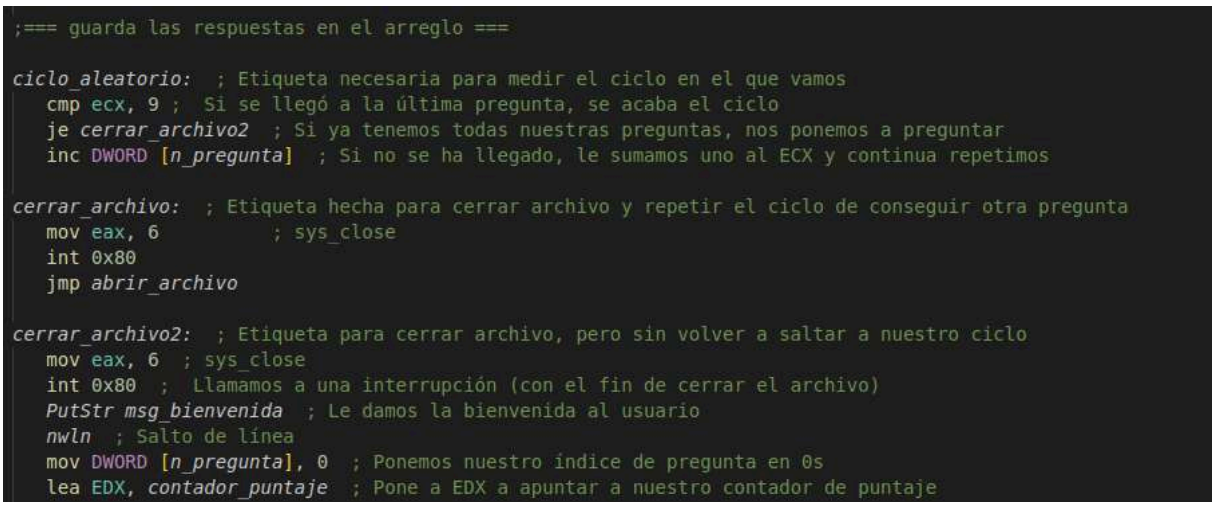
Tanto el guardado de la respuesta como la del puntaje que suma al acertar son sumamente sencillos, y están prácticamente copiados, salvo por algunos incrementos del ESI para continuar con la lectura. Mantenemos el hecho de copiar cada componente a su arreglo respectivo, calculando bien dónde ubicamos cada uno.



```
; === Guardado de respuesta ===

guardar_respuesta:  ; Etiqueta encargada de guardar la respuesta
   lea EDI, [arreglo_respuestas + ecx*tamannio_respuesta] ;  Cada respuesta contiene 2 bytes, guardada en su respectiva posición según lo que almacena ECX (índice de pregunta actual de los arreglos)
   mov al, [ESI]  ; ESI siempre apunta a lo siguiente a guardar, según el formato del documento
   mov [EDI], al   ; Se guarda el carácter en su respectivo espacio
   inc EDI  ; Se incrementa en 1 para revisar lo que es el enter
   mov BYTE[EDI], 0  ; Se guarda nulo en ese espacio para borrarlo
   inc ESI  ; Siguiente en el buffer
   inc ESI ;  Siguiente en el buffer (para saltar el enter)

; === Guardado de puntaje ===
guardar_puntaje:  ; Etiqueta encargada de guardar el puntaje
   lea EDI, [arreglo_puntaje + ecx*tamannio_respuesta] ;  Cada puntaje contiene 2 bytes, guardada en su respectiva posición según lo que almacena ECX (índice de pregunta actual de los arreglos)
   mov al, [ESI]  ; ESI siempre apunta a lo siguiente a guardar, según el formato del documento
   mov [EDI], al   ; Se guarda el carácter en su respectivo espacio
   inc EDI  ; Se incrementa en 1 para revisar lo que es el enter
   mov BYTE[EDI], 0  ; Se guarda nulo en ese espacio para borrarlo
```

*Fig. 16. Guardado de respuesta y de puntaje: comparten prácticamente la misma estructura*

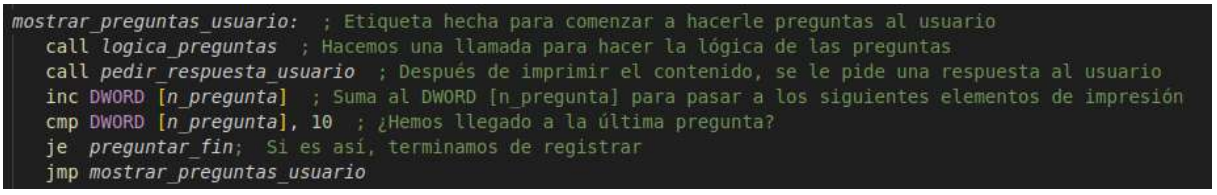
Todo este proceso de leer y sacar los elementos de una pregunta se hace 9 veces más, consiguiendo todos los elementos necesarios para preguntarle al usuario. En cada iteración, cerramos el archivo y lo volvemos a abrir, exceptuando la última.



```
;=== guarda las respuestas en el arreglo ===

ciclo_aleatorio:  ; Etiqueta necesaria para medir el ciclo en el que vamos
   cmp ecx, 9 ;  Si se llegó a la última pregunta, se acaba el ciclo
   je cerrar_archivo2  ; Si ya tenemos todas nuestras preguntas, nos ponemos a preguntar
   inc DWORD [n_pregunta]  ; Si no se ha llegado, le sumamos uno al ECX y continua repetimos

cerrar_archivo:  ; Etiqueta hecha para cerrar archivo y repetir el ciclo de conseguir otra pregunta
   mov eax, 6          ; sys_close
   int 0x80
   jmp abrir_archivo

cerrar_archivo2:  ; Etiqueta para cerrar archivo, pero sin volver a saltar a nuestro ciclo
   mov eax, 6  ; sys_close
   int 0x80  ;  Llamamos a una interrupción (con el fin de cerrar el archivo)
   PutStr msg_bienvenida  ; Le damos la bienvenida al usuario
   nwln  ; Salto de línea
   mov DWORD [n_pregunta], 0  ; Ponemos nuestro índice de pregunta en 0s
   lea EDX, contador_puntaje  ; Pone a EDX a apuntar a nuestro contador de puntaje
```

*Fig. 17. Repetición del ciclo en caso de no haber terminado de conseguir preguntas*

Una vez con todas las preguntas y sus datos en los arreglos, pasamos a la secuencia de hacerle preguntas al usuario. Para tener métodos más reutilizables, priorizamos el uso de llamadas `call` y `ret`: esto nos permitió simplificar montones el hecho de repetir la secuencia por pregunta, además de darnos una mayor dirección hacía como debíamos manejar el hecho de mostrar la información.



```
mostrar_preguntas_usuario:  ; Etiqueta hecha para comenzar a hacerle preguntas al usuario
   call logica_preguntas  ; Hacemos una llamada para hacer la lógica de las preguntas
   call pedir_respuesta_usuario  ; Después de imprimir el contenido, se le pide una respuesta al usuario
   inc DWORD [n_pregunta]  ; Suma al DWORD [n_pregunta] para pasar a los siguientes elementos de impresión
   cmp DWORD [n_pregunta], 10  ; ¿Hemos llegado a la última pregunta?
   je  preguntar_fin;  Si es así, terminamos de registrar
   jmp mostrar_preguntas_usuario
```

*Fig. 18. Etiqueta principal para mostrarle preguntas al usuario y verificar por cual va*

El hecho de ordenar todo en arreglos nos da una flexibilidad enorme a la hora de recorrerlos. Con un solo índice, podemos acceder a todos los espacios deseados e imprimirlos. Podemos apreciarlo mejor en la Fig. 19. dónde se observa que calculamos un índice en EAX, apuntamos a esa posición con ESI y nos podemos imprimir con el método `imprimir_letras`.

```
logica_preguntas:
; IMPRESIÓN DE PREGUNTA
   mov EAX, DWORD [n_pregunta]  ;  Ponemos en EAX el índice del arreglo que llevamos
   imul EAX, tamannio_pregunta  ; Lo multiplicamos por el tamaño de una pregunta
   lea ESI, [arreglo_preguntas + EAX]  ; Ponemos a ESI a apuntar a ese espacio
   call imprimir_letras  ; Se imprime la pregunta
   nwln  ; Salto de línea

; IMPRESIÓN DE OPCIONES
   xor EBX, EBX  ;  Se deja el EBX en 0s (limpiamos registro)

loop_impresion_opciones:  ; Bucle para la impresión de las 4 opciones
   mov EAX, DWORD [n_pregunta]  ;  Ponemos en EAX el índice del arreglo que llevamos
   mov ECX, EBX  ; Ponemos en ECX por la opción de pregunta que vamos revisando
   imul EAX, bloque_opciones  ; Lo multiplicamos por el tamaño del bloque de opciones
   imul ECX, tamannio_opcion  ; Lo multiplicamos por el tamaño de una sola opción
   add EAX, ECX  ; Se suman EAX y ECX para obtener el offset indicado
   lea ESI, [arreglo_opciones + EAX]  ; Ponemos a ESI a apuntar al espacio indicado del arreglo_opciones
   call imprimir_letras;  ; Imprimimos
   nwln  ; Salto de línea
   inc EBX  ; Se incrementa EBX porque se revisó ya una opción
   cmp EBX, 4  ; Si EBX es igual a 4, significa que revisamos ya todas las opciones
   je return  ; Si ya se imprimieron las opciones, se vuelve al mostrar_preguntas_usuario
   jmp loop_impresion_opciones


imprimir_letras:  ; Impresión letra por letra
   cmp WORD [ESI], 0  ; Se compara la letra en [ESI] con 0
   je return  ; Si es 0, se llegó al final de la pregunta y se acaba la ejecución
   mov AL, [ESI]  ; Si no es 0, se utiliza PutCh para imprimir en pantalla hasta que se llegue al nulo
   PutCh AL  ; Ponemos el byte en pantalla
   inc ESI  ; Incrementamos ESI para revisar el siguiente espacio
   jmp imprimir_letras  ; Repetimos el ciclo de impresión
```

*Fig. 18. Secuencia para imprimir pregunta y sus opciones: ambas comparten el método de imprimir letras*

Cuando la pregunta está en la pantalla del usuario, queda pedirle una respuesta. La etiqueta `pedir_respuesta_usuario` nos permite hacer exactamente esto. Le mostramos al jugador el valor que se va a sumar a su puntaje si acierta la pregunta, además de indicarle a dónde debe escribir su opción elegida. Aquí, es tan sencillo como comparar el carácter introducido con el que se encuentra en el arreglo de respuestas para determinar si tenemos una respuesta correcta o una incorrecta.

```
pedir_respuesta_usuario:  ; Lógica para pedirle respuesta al usuario
   mov ECX, DWORD [n_pregunta]  ;  Ponemos en ECX el índice del arreglo que llevamos
   imul ECX, tamannio_respuesta  ; Se calcula el offset dentro de arreglo_respuestas
   lea ESI, [arreglo_puntaje + ECX]  ; ESI apunta a la posición correcta del arreglo de puntajes
   PutStr msg_valor_pregunta_1  ; Mensaje de puntaje 1 se muestra en pantalla
   PutCh [ESI]  ; Se muestra el puntaje de la pregunta actual
   PutStr msg_valor_pregunta_2  ; Mensaje de puntaje 2 se muestra en pantalla
   nwln  ; Salto de línea
   PutStr msg_indicar_respuesta  ; Se pone en pantalla el mensaje de respuesta
   GetCh  AL  ; Se almacena la respuesta del usuario en AL
   nwln  ; Salto de línea
   lea ESI, [arreglo_respuestas + ECX]  ; ESI apunta a la posición correcta del arreglo de respuestas
   call procesar_char  ; Pasamos el contenido de AL a mayúsculas
   cmp AL, [ESI]  ; Comparamos la respuesta correcta con la respuesta introducida
   je respuesta_correcta  ; Si ambos espacios son iguales, significa que la respuesta es correcta
   jmp respuesta_incorrecta  ; Si son distintos, la  respuesta es incorrecta
```

*Fig. 19. Etiqueta para dar información al usuario y procesar su respuesta*

Como mejora de calidad de vida, tomamos el código para transformar a mayúsculas del libro de Silvarama [4]: esto para que introducir una letra minúscula pero correcta sea también contado como un acierto.

```
; ═══ Paso de respuestas a mayúscula═══
procesar_char:  ; Etiqueta para pasar de minúsculas a mayúsculas
        cmp AL, 'a'  ; Si el carácter es inferior a la a minúscula
        jl no_operacion  ; No es un carácter en minúscula
        cmp AL, 'z' ; Si el carácter es superior a la z minúscula
        jg no_operacion;  No es un carácter minúscula

transformar_mayuscula:  ; Etiqueta a la que se llega si la letra es minúscula
        add AL, 'A'-'a'  ; Se tranforma con una suma la letra en mayúscula
        ret  ; Se devuelve al call inicial

no_operacion:  ; Etiqueta a la que se llega el carácter no es minúscula (una mayúscula o cualquier carácter aparte)
        ret  ; Se devuelve al call inicial, no hay transformación real
```

*Fig. 20. Procedimiento de [4] adaptado para transformar mayúsculas*

He aquí el procedimiento llamado en la *fig.20* para la mejora de la calidad de vida.

```
respuesta_correcta:  ; Etiqueta cuando el usuario acierta la pregunta
   mov EAX, DWORD [arreglo_puntaje + ECX]  ; Guarda el contenido de la posición del puntaje en AL
   sub EAX, '0'  ; Convierte el carácter a su valor numérico
   add DWORD [EDX], EAX  ; Incrementa el puntaje
   PutStr msg_correcto  ; Mensaje de éxito
   nwln  ; Salto de línea
   PutStr msg_puntaje_1  ; Mensaje de puntaje 1 se muestra en pantalla
   PutInt [EDX]  ; Se muestra el puntaje
   PutStr msg_puntaje_2  ; Mensaje de puntaje 2 se muestra en pantalla
   nwln  ; Salto de línea
   nwln  ; Salto de línea
   jmp return  ; Vuelta a mostrar_preguntas_usuario
```

*Fig. 21. Etiqueta para el procesamiento de una pregunta cuya respuesta fue acertada.*

Al final del proceso de la figura 19, se poseen dos jumps. Uno de ellos va a la etiqueta `respuesta_correcta` en esta etiqueta, se mueve el valor de la respuesta guardado en el *arreglo_puntaje* y luego se transforma dicho carácter a su valor numérico y se suma el puntaje guardado en EDX. Finalmente retorna hasta `mostrar_respuesta_usuario` para así continuar hasta que se hayan recorrido las 10 preguntas. Esto se puede ver mejor en la figura 18, donde se compara si se ha llegado a 10. Antes de este salto, imprime mensajes necesarios para el usuario en pantalla.

```
respuesta_incorrecta:  ; Etiqueta cuando el usuario falla la pregunta
   PutStr msg_incorrecto  ; Mensaje de fallo
   PutCh [ESI]  ; Se muestra la respuesta correcta (almacenada en ESI)
   nwln  ; Salto de línea
   PutStr msg_puntaje_1  ; Mensaje de puntaje 1 se muestra en pantalla
   PutInt [EDX]  ; Se muestra el puntaje
   PutStr msg_puntaje_2  ; Mensaje de puntaje 2 se muestra en pantalla
   nwln  ; Salto de línea
   nwln  ; Salto de línea
   jmp return  ; Vuelta a mostrar_preguntas_usuario
```

*Fig. 21. Etiqueta para el procesamiento de una pregunta cuya respuesta fue incorrecta.*

Si la pregunta no es correcta, salta a este procedimiento, el cual, solamente muestra la respuesta correcta en pantalla, no incrementa el puntaje, e imprime en pantalla información útil para el usuario, antes de retornar a `mostrar_respuesta_usuario`.

```
return:  ; Etiqueta para hacer rets condicionales
   ret  ; Return para cuando hay un call
```

*Fig. 22. Return de figuras 20 y 21.*

Este es el return usado por las dos figuras anteriores.

```
error:  ; Etiqueta de error
        mov EAX, 6
        int 0x80
        nwln  ; Salto de línea
        PutStr msg_error  ; Mensaje de error
        nwln  ; Salto de línea
    jmp fin  ; Salto al final del programa
```

*Fig. 23. Etiqueta de error por si sucede un error al leer el archivo.*

Solamente por medidas de precaución se realizó esta etiqueta por si sucede un error durante el procesamiento del archivo, donde finaliza el programa tras encontrarlo, además se cierra el archivo por si acaso. El código que llama a esta etiqueta está en la figura 8.

```
preguntar_fin:  ; Etiqueta para preguntar si desea volver a jugar
   PutStr msg_volver_jugar  ; Se pregunta al usuario si desea volver a jugar
   GetCh AL  ; Se obtiene la respuesta del usuario
   nwln  ; Salto de línea
   call procesar_char  ; Pasamos el contenido de AL a mayúsculas
   cmp AL, 'S'  ; Se compara si la respuesta es Sí
   je inicio_reinicio  ; Si es así, se reinicia el juego
                       ;Si no acaba el programa.


fin:
    .EXIT
```

*Fig. 24. Etiquetas del proceso de finalización del programa.*

Tras pasar las 10 preguntas, desde el código de la figura 18, se salta a esta etiqueta. La cual solo se imprime en pantalla los enunciados y realizan las validaciones de, si el usuario quiere repetir el juego. Si no salta a fin, que solo cierra el código.

## V. Análisis de Resultados

Tras el desarrollo del programa y su explicación se procede a ejecutarlo 3 veces. Después de ello, se hará un análisis de resultados, para ver si cumple con todos los requerimientos.

Debido a que la ejecución sería muy larga no se va a enseñar dos juegos seguidos. No obstante si se va a mostrar un fragmento que muestre que si se puede.

**Primera ejecución:**

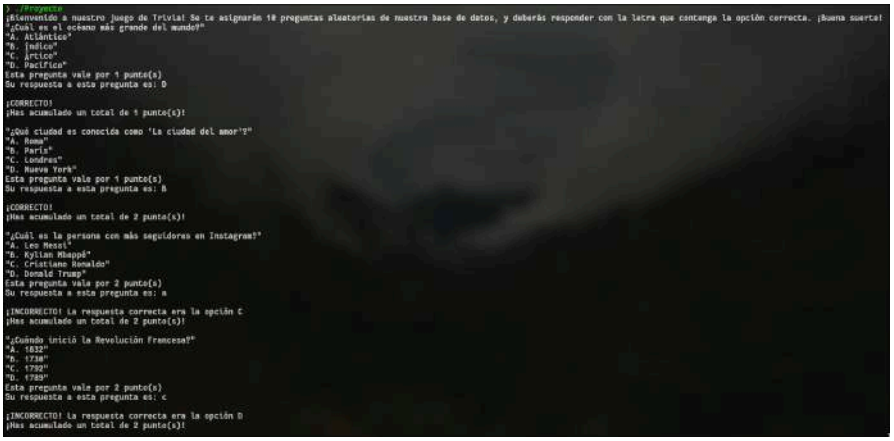



*Fig. 25. Ejecución del programa.*

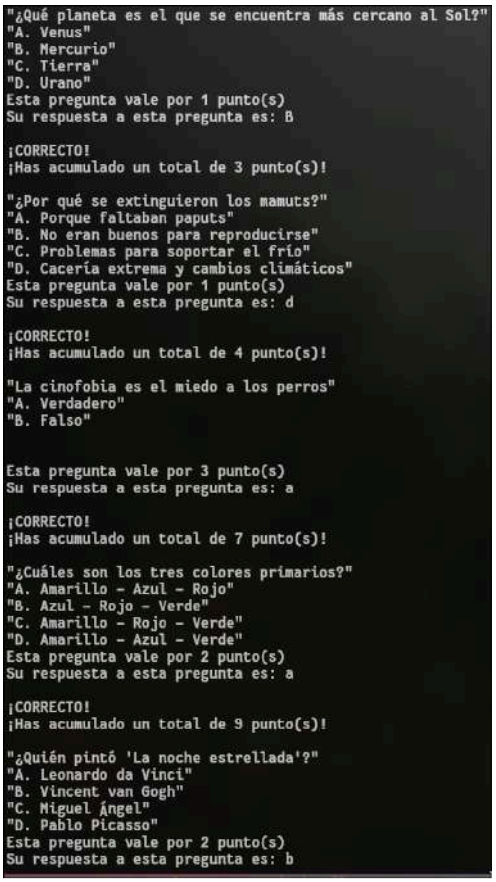



*Fig. 26. Ejecución del programa.*

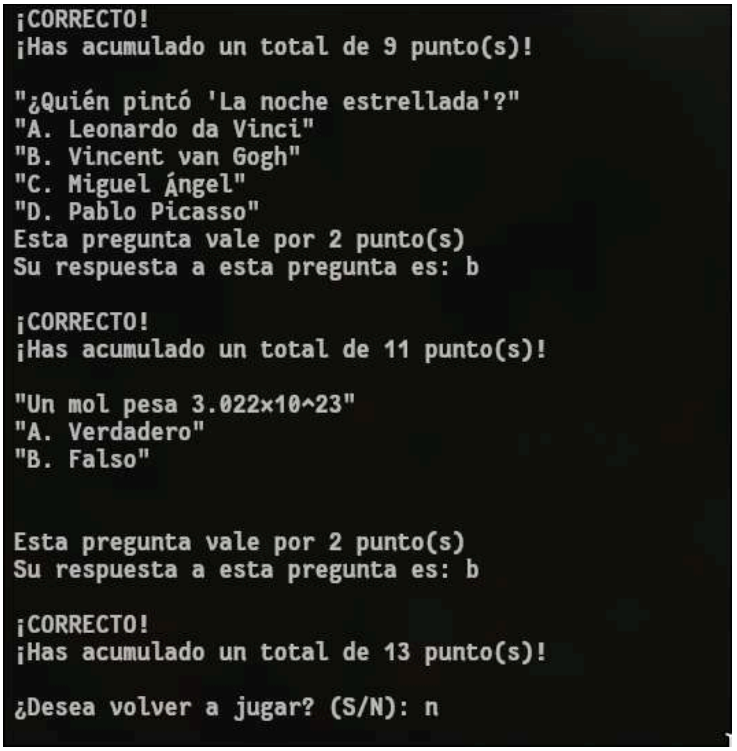



*Fig. 27. Ejecución del programa.*

**Segunda Ejecución:**

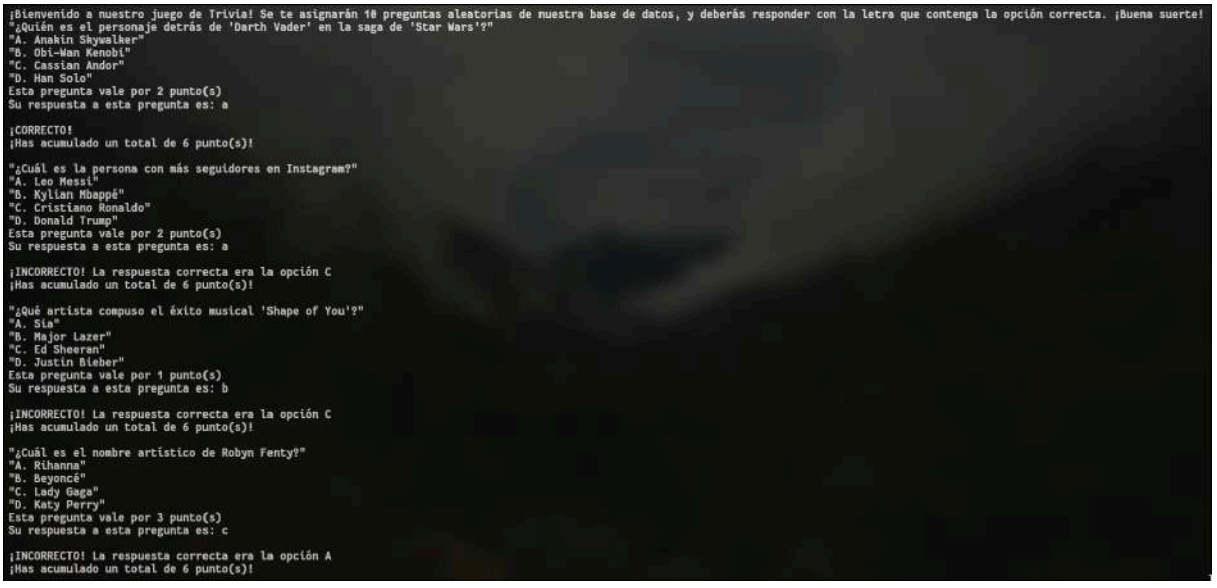



*Fig. 28. Ejecución del programa.*

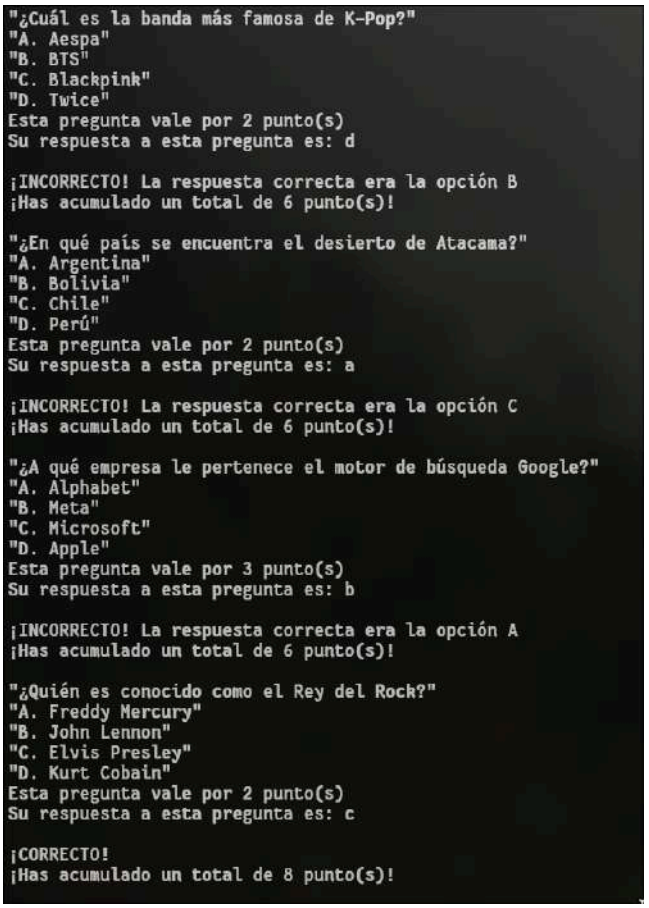



*Fig. 29. Ejecución del programa.*

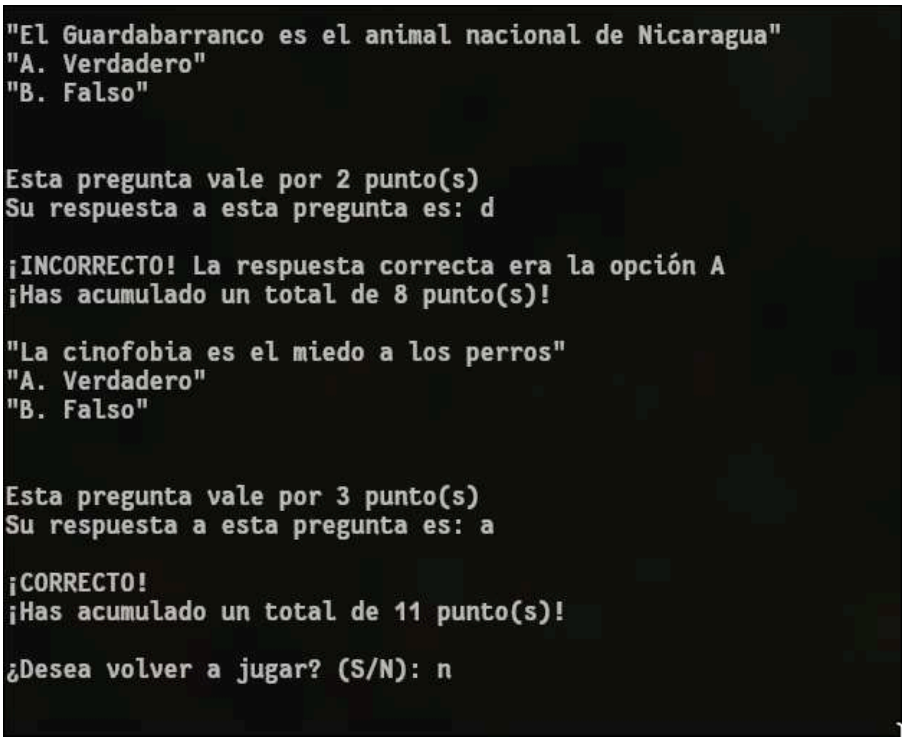



*Fig. 30. Ejecución del programa.*

**Tercera Ejecución:**

```
¡Bienvenido a nuestro juego de Trivia! Se te asignarán 10 preguntas aleatorias de nuestra base
 de datos, y deberás responder con la letra que contenga la opción correcta. ¡Buena suerte!
"Las manzanas pertenecen a la familia de las rosas"
"A. Verdadero"
"B. Falso"


Esta pregunta vale por 3 punto(s)
Su respuesta a esta pregunta es: a

¡CORRECTO!
¡Has acumulado un total de 3 punto(s)!

"¿Cuál es la persona con más seguidores en Instagram?"
"A. Leo Messi"
"B. Kylian Mbappé"
"C. Cristiano Ronaldo"
"D. Donald Trump"
Esta pregunta vale por 2 punto(s)
Su respuesta a esta pregunta es: b

¡INCORRECTO! La respuesta correcta era la opción C
¡Has acumulado un total de 3 punto(s)!

"¿Cuál es el océano más grande del mundo?"
"A. Atlántico"
"B. Índico"
"C. Ártico"
"D. Pacífico"
Esta pregunta vale por 1 punto(s)
Su respuesta a esta pregunta es: c

¡INCORRECTO! La respuesta correcta era la opción D
¡Has acumulado un total de 3 punto(s)!

"¿Cuál es la capital de Canadá?"
"A. Ottawa"
"B. Toronto"
"C. Quebec"
"D. Vancouver"
Esta pregunta vale por 2 punto(s)
Su respuesta a esta pregunta es: d

¡INCORRECTO! La respuesta correcta era la opción A
¡Has acumulado un total de 3 punto(s)!
```

*Fig. 31. Ejecución del programa.*

```
"¿Quién es el personaje detrás de 'Darth Vader' en la saga de 'Star Wars'?"
"A. Anakin Skywalker"
"B. Obi-Wan Kenobi"
"C. Cassian Andor"
"D. Han Solo"
Esta pregunta vale por 2 punto(s)
Su respuesta a esta pregunta es: a

¡CORRECTO!
¡Has acumulado un total de 5 punto(s)!

"¿Qué planeta es el que se encuentra más cercano al Sol?"
"A. Venus"
"B. Mercurio"
"C. Tierra"
"D. Urano"
Esta pregunta vale por 1 punto(s)
Su respuesta a esta pregunta es: b

¡CORRECTO!
¡Has acumulado un total de 6 punto(s)!

"¿Cuáles son los tres colores primarios?"
"A. Amarillo - Azul - Rojo"
"B. Azul - Rojo - Verde"
"C. Amarillo - Rojo - Verde"
"D. Amarillo - Azul - Verde"
Esta pregunta vale por 2 punto(s)
Su respuesta a esta pregunta es: c

¡INCORRECTO! La respuesta correcta era la opción A
¡Has acumulado un total de 6 punto(s)!

"¿Cuántos anillos hay en la bandera olímpica?"
"A. 4"
"B. 5"
"C. 6"
"D. 3"
Esta pregunta vale por 2 punto(s)
Su respuesta a esta pregunta es: d

¡INCORRECTO! La respuesta correcta era la opción B
¡Has acumulado un total de 6 punto(s)!
```

*Fig. 32. Ejecución del programa.*

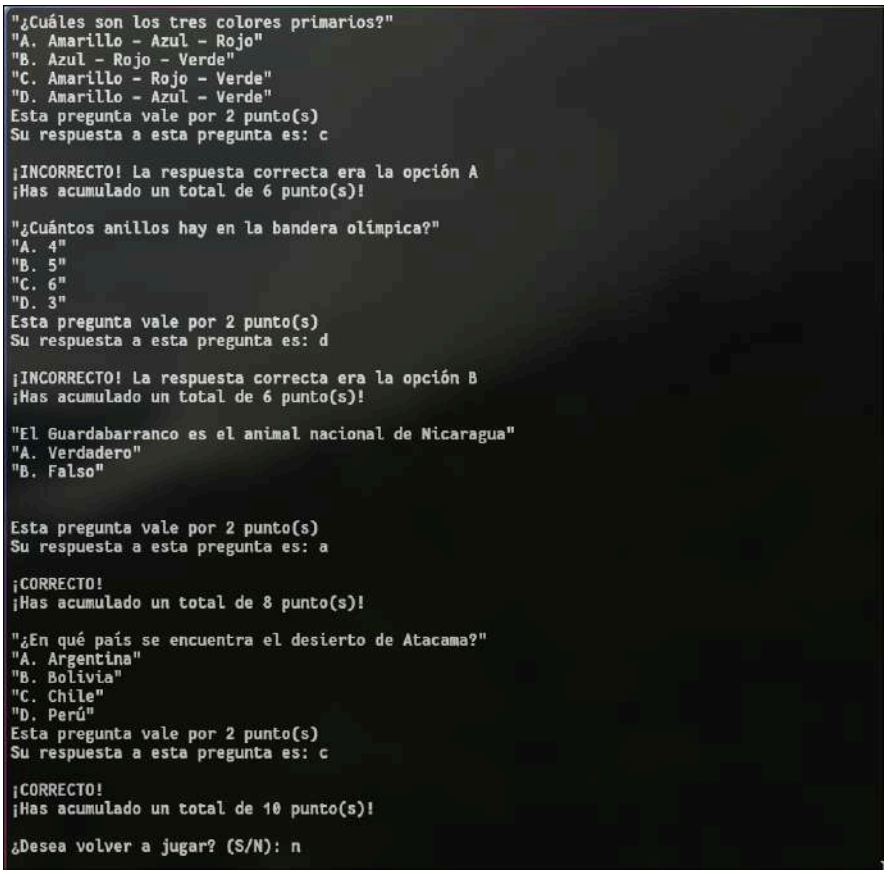



*Fig. 33. Ejecución del programa.*

**Prueba de repetición y consistencia:**

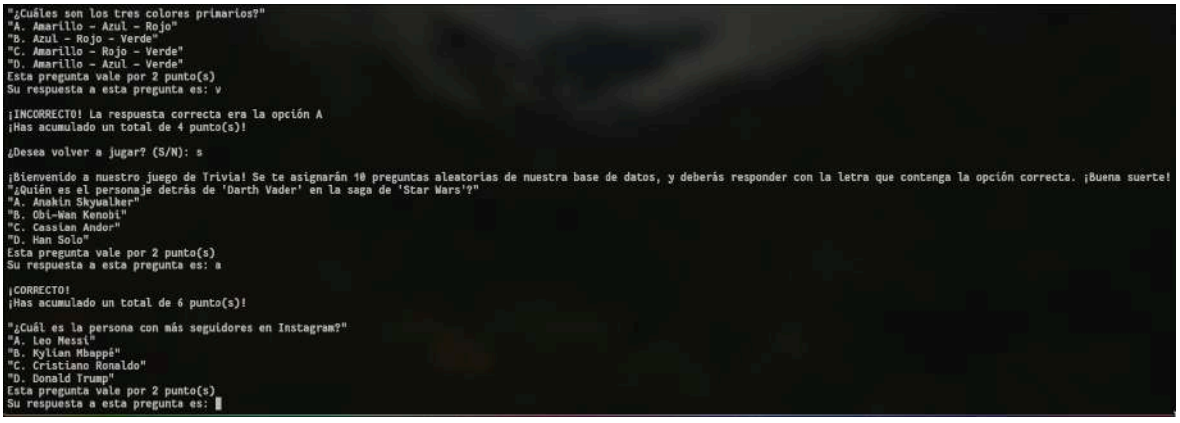



*Fig. 34. Ejecución del programa, prueba que la repetición de partida es exitosa.*

**Análisis:**

Cómo se logra apreciar el proyecto cumple con todas los requerimientos.

Del archivo accede a las preguntas y las imprime correctamente. Hace las validaciones necesarias para que el usuario entienda el siguiente paso a continuación y verifica la respuesta. Las preguntas son aleatorias, y poseen puntaje diferente si son respondidas correctamente y finalmente se puede repetir indefinidamente mientras aún suma el puntaje.

## VI. Observaciones

El enfoque de guardar la información del archivo fue exitoso y adecuado para este proyecto, debido a que facilitó su acceso y posterior verificación mediante el uso de arreglo de caracteres.

El uso de un ciclo de lectura y un buffer basura para llegar a la pregunta adecuada mediante el conteo de caracteres ‘enter’ fue vital para el funcionamiento del programa. Esto se debía hacer de esta forma, debido a que cada pregunta posee una cantidad diferente de bytes, no obstante si una cantidad fija de caracteres ‘enter’.

Una estructura de datos clara y organizada, también fue vital para la resolución del problema.

La mayoría del código fue hecho en conjunto antes del enunciado del proyecto, por ende, el github solo posee cambios menores y la documentación.

Finalmente, el uso de diversas herramientas y adecuada comunicación entre compañeros fue el enfoque y habilidad necesaria para la realización del proyecto. Demostrando el uso de organización, habilidades blandas, y conocimiento.

No se cumplieron con requerimientos extra.

## VII. Conclusiones

En conclusión, mediante una comunicación y confianza mutua clara, el uso de fuentes confiables externas, la utilización de un enfoque de almacenamiento de datos por funcionalidad de una estructura de datos adecuadamente organizada, y su empleo en la interacción con el usuario. Permitieron una documentación adecuada al igual que un entregable que cumple con los requerimientos deseados.

No se realizaron requerimientos extra. Por otro lado, debido a que el proyecto fue realizado previo al enunciado, el apartado de github solo posee cambios menores y la versión final de la documentación.

## VIII. Apéndice

```
%include "io.mac"

; ProyectoTrivia.asm
; Lee un archivo hasta EOF y genera un número aleatorio entre 1 y 30
; Y almacena a sus respectivos arreglos los valores

;====================================
;Definicion de nombres
;====================================

bufsize EQU 300       ;  Máxima cantidad de bytes por pregunta
tamannio_pregunta EQU 80  ; Máximo tamaño que puede tener una pregunta
tamannio_respuesta EQU 2  ; Máximo tamaño que puede tener una respuesta
tamannio_opcion EQU 45  ; Máximo tamaño que puede tener una opción
tamannio_totalidad_pregunta  EQU  7   ; Cantidad de líneas que tiene cada pregunta en el
archivo (pregunta + 4 opciones + respuesta + puntaje)
```

```
bloque_opciones EQU tamannio_opcion*4   ; Máximo de tamaño que puede tener un bloque de 4 opciones
cantidad_preguntas EQU 10   ; Cantida de preguntas que se le muestran al usuario en una "run"


.DATA
filename db "PreguntasTrivia.txt",0 ; nombre del archivo
msg_bienvenida db "¡Bienvenido a nuestro juego de Trivia! Se te asignarán 10 preguntas aleatorias de nuestra base de datos, y deberás responder con la letra que contenga la opción correcta. ¡Buena suerte!", 0
; Mensajes de comprobación y estado
msg_correcto db "¡CORRECTO!", 0  ; Mensaje para respuestas correctas
msg_incorrecto db "¡INCORRECTO! La respuesta correcta era la opción ", 0  ; Mensaje para respuestas incorrectas
; Mensajes para mostrar el score


msg_indicar_respuesta db "Su respuesta a esta pregunta es: ", 0
msg_puntaje_1 db "¡Has acumulado un total de ", 0
msg_puntaje_2 db " punto(s)!", 0
msg_error db "Error leyendo archivo: intente nuevamente", 0
msg_volver_jugar db "¿Desea volver a jugar? (S/N): ", 0
msg_valor_pregunta_1 db "Esta pregunta vale por ", 0
msg_valor_pregunta_2 db " punto(s)", 0

.UDATA
contador_puntaje resd 1  ; Contador de puntaje total
contador_enter resd 1    ; Cuenta la cantidad de Enters para asi ver las preguntas correctamente
buffer resb bufsize ; Buffer como mediador para guardar todo a sus arreglos
buftrash resb 1        ; Buffer para guardar un valor hasta saltar x lineas

arreglo_preguntas resb tamannio_pregunta*cantidad_preguntas ;reserva 10 preguntas
arreglo_opciones resb bloque_opciones*cantidad_preguntas ;las 4 opciones de las 10 preguntas
arreglo_respuestas resb tamannio_respuesta*cantidad_preguntas ;las respuestas de las 10 preguntas
arreglo_puntaje resb tamannio_respuesta*cantidad_preguntas  ; Arreglo que guarda el puntaje de cada pregunta

n_pregunta resd 1  ; Pregunta actual (índice dentro de casi TODOS los arreglos, exceputando arreglo_num_elegidos)

arreglo_num_elegidos resd cantidad_preguntas   ; Arreglo que guarda los números elegidos para ser presentados
n_repetidos resd 1  ;  Cantidad que guarda la cantidad de numeros en el arreglo (se guarda en un DWORD por comodidad para trabajar con ESI y EDI posteriormente)

;=================================================================
; En resumen cada pregunta posee 84 bytes como maximo
; Cada pregunta posee 4 opciones de respuesta de 45 bytes maximo
; Y cada respuesta correcta posee 2 bytes (A y nulo; por ejemplo)
;=================================================================

.CODE
        .STARTUP
inicio:  ; Incio de nuestro código
    mov DWORD [n_pregunta], 0  ; Se limpia n_pregunta
    mov DWORD [contador_puntaje], 0  ; Se limpia el contador de puntaje
    jmp abrir_archivo;  ; Saltamos a la apertura del archivo

inicio_reinicio:  ; Etiqueta para reiniciar el juego
   ; LIMPIEZA DE TODOS LOS ARREGLOS Y VARIABLES
   xor EAX, EAX  ; Ponemos AL en 0 (contenido que vamos a copiar con stosb)
   lea EDI, [arreglo_preguntas]  ; Dirección de inicio del arreglo de preguntas
    mov ECX, tamannio_pregunta * cantidad_preguntas  ; Movemos a ECX la cantidad total de bytes a limpiar (tamaño de arreglo_preguntas)
   rep stosb  ; rep stosb copia el contenido de AL (0) en [EDI] ECX veces

   lea EDI, [arreglo_opciones]  ; Dirección de inicio del arreglo de opciones
    mov ECX, bloque_opciones * cantidad_preguntas  ; Movemos a ECX la cantidad total de bytes a limpiar (tamaño de arreglo_opciones)
   rep stosb  ; rep stosb copia el contenido de AL (0) en [EDI] ECX veces
```

```
   lea EDI, [arreglo_respuestas]  ; Dirección de inicio del arreglo de respuestas
    mov ECX, tamannio_respuesta * cantidad_preguntas  ; Movemos a ECX la cantidad total de
bytes a limpiar (tamaño de arreglo_respuestas)
   rep stosb  ; rep stosb copia el contenido de AL (0) en [EDI] ECX veces

   lea EDI, [arreglo_puntaje]  ; Dirección de inicio del arreglo de puntajes
    mov ECX, tamannio_respuesta * cantidad_preguntas  ; Movemos a ECX la cantidad total de
bytes a limpiar (tamaño de arreglo_puntaje)
   rep stosb  ; rep stosb copia el contenido de AL (0) en [EDI] ECX veces

   lea EDI, [arreglo_num_elegidos]  ; Dirección de inicio del arreglo de números elegidos
   mov ECX, cantidad_preguntas
   rep stosd  ; rep stosd copia el contenido de EAX (0) en [EDI] ECX veces

   lea EDI, [n_repetidos]  ; Dirección de inicio de n_repetidos
   mov ECX, 1
   rep stosd  ; rep stosd copia el contenido de EAX (0) en [EDI] ECX veces

   mov DWORD [contador_enter], 0  ; Se limpia el contador enters
   mov DWORD [n_pregunta], 0  ; Se limpia n_pregunta

abrir_archivo:  ; Proceso para abrir nuestro archivo de preguntas
    mov eax, 5  ; sys_open
    mov ebx, filename  ; Nombre del archivo
    mov ecx, 0  ; Modo: ReadOnly
    int 0x80  ; Se llama a la interrupción
    cmp eax, 0  ; El resultado fue guardado en EAX: si este dió 0, hubo un error
    js  error  ; Si error, salir
    mov ebx, eax  ; Descriptor del archivo es guardado en EBX


randint:  ; Rutina para volver a llamar randint (numero entre 0 y 29) e iniciar todo otra
vez
    mov DWORD [contador_enter], 0
    push ebx  ; Push necesario para mantener el descriptor en EBX

     rdtsc  ; Se toma el contador de ciclos de reloj del procesador desde el arranque del
sistema
    xor edx, eax  ; Desorden
    rol edx, 13  ; Desorden
    xor eax, edx  ; Desorden
    mov ebx, 35  ; Número maximo
    xor edx, edx  ; Desorden
      div ebx  ; Divide y el número generado obtenido es pseudo-aleatorio (residuo se
encuentra en EDX)

    pop ebx  ; Se recupera nuestro registro EBX

;=================================================
; COMPROBAR SI ESTA REPETIDO EL NUMERO
;=================================================

    mov esi, 0  ; Índice para recorrer arreglo_repetidos

verificar_repetido:  ; Etiqueta para verificar si el número obtenido del randint está
repetido
     cmp esi, DWORD [n_repetidos]  ; Comparamos con DWORD[n_repetidos] para ver si llegamos
al siguiente espacio vacío (DWORD[n_repetidos] es un número)
    je guardar_numero  ; Si llegamos al final, no está repetido

     mov eax, [arreglo_num_elegidos + esi*4] ; Mientras no se haya llegado al final de los
números obtenidos, se lee el siguiente número guardado
    cmp eax, edx  ; Se compara con el obtenido por nuestro randint
    je repetir_randint  ; Si coincide, significa que el número ya está y debemos repetir el
randint
      inc esi  ; Si no son iguales, procedemos a incrementar esi y revisar el siguiente
dígito
    jmp verificar_repetido  ; Saltamos a la misma etiqueta en la que nos encontramos

guardar_numero:  ; Etiqueta a la que llegamos si el número que obtuvimos no está repetido:
     mov eax, DWORD [n_repetidos]  ; Ponemos en EAX la cantidad de números que habíamos
contabilizado antes
    mov [arreglo_num_elegidos + eax*4], edx ; A la siguiente casilla libre se le asigna EDX
(resultado de randint)
     inc DWORD [n_repetidos]  ; Incrementamos la cantidad de números guardados para apuntar
```

```
al siguiente espacio vacío
    ; Se continua con el flujo normal (saltar_loop, etc.)
    jmp no_repetido

repetir_randint:  ; Etiqueta a la que llegamos cuando el número obtenido estaba repetido
    jmp randint  ; Se repite el randint

no_repetido:  ; Etiqueta a la que llegamos cuando nuestro número no fue repetido
     imul edx, tamannio_totalidad_pregunta  ; Se multiplica nuestro resultado de randint por
6
     mov DWORD [contador_enter], edx  ;  Se guarda el valor de los enters a saltar en el
contador

;================================================================================
; Explicacion:
; En el documento no existen nulos solo enters donde se puede leer todo, asi pues
;***************
;* Pregunta1\n *
;* Opcion1\n   *
;* Opcion2\n   *
;* Opcion3\n   *
;* Opcion4\n   *
;* Respuesta\n *
;* Puntaje\n   *
;* Pregunta2\n *
;***************
;
; Para llegar de pregunta1 a pregunta2 se pasan por 6 enters, por ende sacando
; el numero aleatorio de randint y luego multiplicandolo por 6, logramos saltar
; n enters hasta la pregunta deseada
;================================================================================


; === Leer hasta x enters, pues es el final de la pregunta ===
saltar_loop:   ; Etiqueta encargada de posicionarnos dentro del archivo, haciendo que
apuntemos a la pregunta que necesitamos
    mov eax, 3  ; sys_read
    mov ecx, buftrash  ; Dirección del buffer temporal de 1 byte
    mov edx, 1  ; Leemos 1 byte
    int 0x80  ; Procedemos con la interrupción
    cmp eax, 0  ; El resultado fue guardado en EAX: si este dió 0, hubo un error
    jle error  ; EOF o error

    mov al, [buftrash]  ; Cargar el byte leído
    cmp al, 0x0A  ; ¿El byte es un enter?
    jne saltar_loop  ; Si no es un enter, procedemos a leer siguiente

    ; si es enter
    dec DWORD [contador_enter]  ; Decrementamos el contador de enters recorridos
    cmp DWORD [contador_enter], 0  ; Si este llega a su fin, procedemos a leer el archivo
    je leer_archivo  ; Etiqueta para leer el archivo
    jmp saltar_loop  ; Si no se ha llegado al final, se repite otra vez el loop

;=== Leer archivo ===
;Tras saltar la cantidad de enters que dice el archivo ahora si leemos, iniciando en la
pregunta que obtubimos

leer_archivo:  ; Etiqueta para la lectura del archivo
    mov eax, 3  ; sys_read
    mov ecx, buffer  ; Buffer donde se guarda la lectura
    mov edx, bufsize  ; Cantidad máxima a leer
    int 0x80  ; Se hace la interrupción
    cmp eax, 0  ; El resultado fue guardado en EAX: si este dió 0, hubo un error
    jle error  ; EOF o error

     ; En este momento de la ejecución el buffer tiene la pregunta que queríamos, entonces
procedemos a pasarla a su respectivo espacio
    ; Preparativos para copiar pregunta
     mov ecx, DWORD [n_pregunta]  ; Se pone en ECX el número de pregunta en el que estamos
actualmente (índice del arreglo de preguntas)
    lea ESI, buffer  ; Con lea, cargamos la dirección del buffer en ESI
    mov EAX, ECX  ; Ponemos en EAX el número de pregunta en el que estamos
     imul EAX, tamannio_pregunta  ; Mutiplicamos el número de pregunta por el tamaño que
tiene una pregunta (esta operación nos termina dando el offset del siguiente espacio vacío
disponible dentro del arreglo de preguntas)
```

```
    lea EDI, [arreglo_preguntas + EAX]  ; Ponemos a EDI a apuntar a este espacio vacío

;=== Guardar la pregunta al arreglo correspondiente ===

; EDI = Apuntando al inicio del espacio vacío de arreglo_preguntas
; ESI = Apuntando al inicio del buffer que tiene el texto que queremos
guardar_pregunta:  ; Etiqueta para guardar las cadenas necesarias
    mov al,[ESI]  ; Mueve el byte revisado del buffer a AL
    mov [EDI], al  ; Lo copia en el arreglo
    inc ESI  ; Siguiente del buffer
    inc EDI  ; Siguiente del arreglo
    cmp al, 0x0A  ; Se compara con enter (indica el final de la pregunta)

    je guardar_opciones  ; Si se llegó al final, pasamos a guardar las opciones
     jmp guardar_pregunta  ; Si no ha llegado al enter, se sigue leyendo y copiando del
buffer

;=== Guardar opciones de la respectiva pregunta ===

guardar_opciones:  ; Preparativos para guardar las opciones de la pregunta
   mov DWORD[EDI-1], 0  ; Se almacena un nulo al final de la pregunta (sirve para eliminar
el carácter del enter)
   mov BYTE[contador_enter], 0 ; Se reinicia el índice de opción a 0

   ; Calcular dirección inicial de la primera opción:
   mov eax, ecx  ; Se pone en EAX el contenido de ECX (recordando que ECX tiene el número
de pregunta en el que estamos actualmente)
   imul eax, bloque_opciones  ; 180 = 45*4 (tamaño bloque por pregunta)
   lea edi, [arreglo_opciones + eax]  ; Así como se hizo con las preguntas, ponemos a EDI a
apuntar al siguiente espacio vacío donde escribiremos las opciones que obtuvimos


ciclo_opciones:  ; Etiqueta del ciclo de lectura de opciones
   mov al,[ESI]  ; Mueve el byte revisado del buffer a AL
   mov [EDI], al  ; Lo copia en el arreglo
   inc ESI  ; Siguiente del buffer
   inc EDI  ; Siguiente del arreglo

   cmp al, 0x0A  ; Si hay un enter, debemos recalcular y seguir con la siguiente opción de
pregunta
   je siguiente_opcion  ; Se llegó a un enter, entonces se pasa a la siguiente opción
   jmp ciclo_opciones  ; Si no, seguimos leyendo normal


siguiente_opcion:  ; Etiqueta para el paso entre opciones

; === Recálculo de posición y ajuste de opción ===
   mov BYTE[EDI-1], 0  ; Se reemplaza el enter con un 0
    inc DWORD [contador_enter]  ; Siguiente opcion (ajuste de opción al incrementar el
contador de enters que llevamos)

   ; Cálculo de nueva posición para la siguiente opción
     mov eax, ecx  ; Se pone en EAX el contenido de ECX (número de pregunta en el que
estamos actualmente)
    imul eax, bloque_opciones  ; 180 = 45*4 (tamaño bloque por pregunta)
     mov ebx, DWORD [contador_enter]  ; Ponemos en EBX el contador de enters que llevamos
(número entre 0 y 3)
    imul ebx, tamannio_opcion  ; Este número guardado en EBX es multiplicado por el tamaño
que consume cada opción, asegurando que el espacio que registramos es el indicado
    add eax, ebx  ; Le sumamos EBX a nuestro offset, almacenado en EAX
     lea edi, [arreglo_opciones + eax]  ; Ponemos a EDI a apuntar a ese espacio recién
calculado

; === Se compara con 4: si es el caso, se terminó de guardar ===
    cmp DWORD [contador_enter], 4  ; Si el contador de enters llegó a 4, significa que
leímos todas las opciones y pasamos la guardado de respuesta
   je guardar_respuesta  ; Paso al guardado de respuesta
   jmp ciclo_opciones  ; Si no se llegó a los 4 enters, se siguen guardando opciones en su
respectivo arreglo

; === Guardado de respuesta ===

guardar_respuesta:  ; Etiqueta encargada de guardar la respuesta
    lea EDI, [arreglo_respuestas + ecx*tamannio_respuesta] ;  Cada respuesta contiene 2
bytes, guardada en su respectiva posición según lo que almacena ECX (índice de pregunta
```

```
actual de los arreglos)
    mov al, [ESI]  ; ESI siempre apunta a lo siguiente a guardar, según el formato del
documento
   mov [EDI], al   ; Se guarda el carácter en su respectivo espacio
   inc EDI  ; Se incrementa en 1 para revisar lo que es el enter
   mov BYTE[EDI], 0  ; Se guarda nulo en ese espacio para borrarlo
   inc ESI  ; Siguiente en el buffer
   inc ESI ;  Siguiente en el buffer (para saltar el enter)

; === Guardado de puntaje ===
guardar_puntaje:  ; Etiqueta encargada de guardar el puntaje
    lea EDI, [arreglo_puntaje + ecx*tamannio_respuesta] ;  Cada puntaje contiene 2 bytes,
guardada en su respectiva posición según lo que almacena ECX (índice de pregunta actual de
los arreglos)
    mov al, [ESI]  ; ESI siempre apunta a lo siguiente a guardar, según el formato del
documento
   mov [EDI], al   ; Se guarda el carácter en su respectivo espacio
   inc EDI  ; Se incrementa en 1 para revisar lo que es el enter
   mov BYTE[EDI], 0  ; Se guarda nulo en ese espacio para borrarlo

;=== guarda las respuestas en el arreglo ===

ciclo_aleatorio:  ; Etiqueta necesaria para medir el ciclo en el que vamos
   cmp ecx, 9 ;  Si se llegó a la última pregunta, se acaba el ciclo
   je cerrar_archivo2  ; Si ya tenemos todas nuestras preguntas, nos ponemos a preguntar
    inc DWORD [n_pregunta]  ; Si no se ha llegado, le sumamos uno al ECX y continua
repetimos

cerrar_archivo:  ; Etiqueta hecha para cerrar archivo y repetir el ciclo de conseguir otra
pregunta
   mov eax, 6          ; sys_close
   int 0x80
   jmp abrir_archivo

cerrar_archivo2:  ; Etiqueta para cerrar archivo, pero sin volver a saltar a nuestro ciclo
   mov eax, 6  ; sys_close
   int 0x80  ;  Llamamos a una interrupción (con el fin de cerrar el archivo)
   PutStr msg_bienvenida  ; Le damos la bienvenida al usuario
   nwln  ; Salto de línea
   mov DWORD [n_pregunta], 0  ; Ponemos nuestro índice de pregunta en 0s
   lea EDX, contador_puntaje  ; Pone a EDX a apuntar a nuestro contador de puntaje

mostrar_preguntas_usuario:  ; Etiqueta hecha para comenzar a hacerle preguntas al usuario
   call logica_preguntas  ; Hacemos una llamada para hacer la lógica de las preguntas
    call pedir_respuesta_usuario  ; Después de imprimir el contenido, se le pide una
respuesta al usuario
    inc DWORD [n_pregunta]  ; Suma al DWORD [n_pregunta] para pasar a los siguientes
elementos de impresión
   cmp DWORD [n_pregunta], 10  ; ¿Hemos llegado a la última pregunta?
   je  preguntar_fin;  Si es así, terminamos de registrar
   jmp mostrar_preguntas_usuario

logica_preguntas:
; IMPRESIÓN DE PREGUNTA
   mov EAX, DWORD [n_pregunta]  ;  Ponemos en EAX el índice del arreglo que llevamos
   imul EAX, tamannio_pregunta  ; Lo multiplicamos por el tamaño de una pregunta
   lea ESI, [arreglo_preguntas + EAX]  ; Ponemos a ESI a apuntar a ese espacio
   call imprimir_letras  ; Se imprime la pregunta
   nwln  ; Salto de línea

; IMPRESIÓN DE OPCIONES
   xor EBX, EBX  ;  Se deja el EBX en 0s (limpiamos registro)

loop_impresion_opciones:  ; Bucle para la impresión de las 4 opciones
   mov EAX, DWORD [n_pregunta]  ;  Ponemos en EAX el índice del arreglo que llevamos
   mov ECX, EBX  ; Ponemos en ECX por la opción de pregunta que vamos revisando
   imul EAX, bloque_opciones  ; Lo multiplicamos por el tamaño del bloque de opciones
   imul ECX, tamannio_opcion  ; Lo multiplicamos por el tamaño de una sola opción
   add EAX, ECX  ; Se suman EAX y ECX para obtener el offset indicado
    lea ESI, [arreglo_opciones + EAX]  ; Ponemos a ESI a apuntar al espacio indicado del
arreglo_opciones
   call imprimir_letras;  ; Imprimimos
   nwln  ; Salto de línea
   inc EBX  ; Se incrementa EBX porque se revisó ya una opción
   cmp EBX, 4  ; Si EBX es igual a 4, significa que revisamos ya todas las opciones
```

```
   je return  ; Si ya se imprimieron las opciones, se vuelve al mostrar_preguntas_usuario
   jmp loop_impresion_opciones


imprimir_letras:  ; Impresión letra por letra
   cmp WORD [ESI], 0  ; Se compara la letra en [ESI] con 0
   je return  ; Si es 0, se llegó al final de la pregunta y se acaba la ejecución
    mov AL, [ESI]   ; Si no es 0, se utiliza PutCh para imprimir en pantalla hasta que se
llegue al nulo
   PutCh AL  ; Ponemos el byte en pantalla
   inc ESI  ; Incrementamos ESI para revisar el siguiente espacio
   jmp imprimir_letras  ; Repetimos el ciclo de impresión


pedir_respuesta_usuario:  ; Lógica para pedirle respuesta al usuario
   mov ECX, DWORD [n_pregunta]  ;  Ponemos en ECX el índice del arreglo que llevamos
   imul ECX, tamannio_respuesta  ; Se calcula el offset dentro de arreglo_respuestas
    lea ESI, [arreglo_puntaje + ECX]   ; ESI apunta a la posición correcta del arreglo de
puntajes
   PutStr msg_valor_pregunta_1  ; Mensaje de puntaje 1 se muestra en pantalla
   PutCh [ESI]  ; Se muestra el puntaje de la pregunta actual
   PutStr msg_valor_pregunta_2  ; Mensaje de puntaje 2 se muestra en pantalla
   nwln  ; Salto de línea
   PutStr msg_indicar_respuesta  ; Se pone en pantalla el mensaje de respuesta
   GetCh  AL  ; Se almacena la respuesta del usuario en AL
   nwln  ; Salto de línea
    lea ESI, [arreglo_respuestas + ECX]  ; ESI apunta a la posición correcta del arreglo de
respuestas
   call procesar_char  ; Pasamos el contenido de AL a mayúsculas
   cmp AL, [ESI]  ; Comparamos la respuesta correcta con la respuesta introducida
    je respuesta_correcta   ; Si ambos espacios son iguales, significa que la respuesta es
correcta
   jmp respuesta_incorrecta  ; Si son distintos, la  respuesta es incorrecta

; === Paso de respuestas a mayúscula===
procesar_char:  ; Etiqueta para pasar de minúsculas a mayúsculas
       cmp AL, 'a'  ; Si el carácter es inferior a la a minúscula
       jl no_operacion  ; No es un carácter en minúscula
       cmp AL, 'z' ; Si el carácter es superior a la z minúscula
       jg no_operacion;  No es un carácter minúscula

transformar_mayuscula:  ; Etiqueta a la que se llega si la letra es minúscula
       add AL, 'A'-'a'  ; Se tranforma con una suma la letra en mayúscula
       ret  ; Se devuelve al call inicial

no_operacion:   ; Etiqueta a la que se llega el carácter no es minúscula (una mayúscula o
cualquier carácter aparte)
       ret  ; Se devuelve al call inicial, no hay transformación real

respuesta_correcta:  ; Etiqueta cuando el usuario acierta la pregunta
   mov EAX, DWORD [arreglo_puntaje + ECX]  ; Guarda el contenido de la posición del puntaje
en AL
   sub EAX, '0'  ; Convierte el carácter a su valor numérico
   add DWORD [EDX], EAX  ; Incrementa el puntaje
   PutStr msg_correcto  ; Mensaje de éxito
   nwln  ; Salto de línea
   PutStr msg_puntaje_1  ; Mensaje de puntaje 1 se muestra en pantalla
   PutInt [EDX]  ; Se muestra el puntaje
   PutStr msg_puntaje_2  ; Mensaje de puntaje 2 se muestra en pantalla
   nwln  ; Salto de línea
   nwln  ; Salto de línea
   jmp return  ; Vuelta a mostrar_preguntas_usuario

respuesta_incorrecta:  ; Etiqueta cuando el usuario falla la pregunta
   PutStr msg_incorrecto  ; Mensaje de fallo
   PutCh [ESI]  ; Se muestra la respuesta correcta (almacenada en ESI)
   nwln  ; Salto de línea
   PutStr msg_puntaje_1  ; Mensaje de puntaje 1 se muestra en pantalla
   PutInt [EDX]  ; Se muestra el puntaje
   PutStr msg_puntaje_2  ; Mensaje de puntaje 2 se muestra en pantalla
   nwln  ; Salto de línea
   nwln  ; Salto de línea
   jmp return  ; Vuelta a mostrar_preguntas_usuario
```

```
return:  ; Etiqueta para hacer rets condicionales
   ret  ; Return para cuando hay un call

error:  ; Etiqueta de error
        mov EAX, 6
        int 0x80
        nwln  ; Salto de línea
        PutStr msg_error  ; Mensaje de error
        nwln  ; Salto de línea
   jmp fin  ; Salto al final del programa

preguntar_fin:  ; Etiqueta para preguntar si desea volver a jugar
   PutStr msg_volver_jugar  ; Se pregunta al usuario si desea volver a jugar
   GetCh AL  ; Se obtiene la respuesta del usuario
   nwln  ; Salto de línea
   call procesar_char  ; Pasamos el contenido de AL a mayúsculas
   cmp AL, 'S'  ; Se compara si la respuesta es Sí
   je inicio_reinicio  ; Si es así, se reinicia el juego
                     ;Si no acaba el programa.


fin:
    .EXIT


;TODO: mensaje de error y revisar repetidos

;================================================
;Informacion útil
;================================================


;Por pregunta son 81 bytes de offset
;Por bloque de opciones son 172 bytes y por opcion son 43
;Por respuesta son 2 bytes

;Si deseo llegar a la segunda opcion de la pregunta 8 debo
;En arreglo_pregunta es:
;(n_pregunta-1)*tamannio_pregunta
;En arreglo_opciones es:
;[n_pregunta * tamannio_bloque_opciones + tamannio_opcion * n_opcion]
;En arreglo_respuestas:
;(n_pregunta-1)*tamannio_respuesta
```

**Link del Github:**
https://github.com/kokoju/Trivia-NASM.git

## IX. Bibliografía

[1] NASM Project, NASM – The Netwide Assembler (version 3.01rc4) — Documentación, [En línea]. Disponible en: https://www.nasm.us/xdoc/3.01rc4/html/nasm00.html

[2] GNOME Project, *Gedit Manual*, GNOME Help, [En línea]. Disponible en: https://help.gnome.org/users/gedit/stable/

[3] Microsoft. “Visual Studio Code - The open source AI code editor”. [En línea]. Disponible en: https://code.visualstudio.com/

[4] Silvarama, “Guide to Assembly Language Programming in Linux”. Springer Science+Business Media, Inc., 233 Spring Street, New York, NY 10013, USA, 2005, pp.402-421

[5] Ruistorm, “Random in Assembly/ASM (tasm),” Jun. 14, 2003. https://www.experts-exchange.com/questions/20648317/Random-in-Assembly-ASM-tasm.html