UNIVERSIDAD DEL VALLE DE GUATEMALA

CC3054 - Organización de Computadoras y Assembler

Sección 21

Ing. Roger Díaz

Proyecto 04

Acertijo del Pastor - Assembler

Diego Leiva	21752
Pablo Orellana	21970

GUATEMALA, 02 de junio de 2023

Acertijo del Pastor - Assembler

El presente informe detalla la creación de un programa en MASM x86 que aborda el clásico acertijo del pastor y su desafío para cruzar un lobo, una oveja y una lechuga al otro lado de un río. Este proyecto tiene como objetivo principal desarrollar una solución al acertijo utilizando el lenguaje de ensamblador MASM x86.

El acertijo del pastor plantea una situación en la que un pastor se encuentra en una orilla del río junto con un lobo, una oveja y una lechuga. El desafío consiste en trasladar a todos los elementos al otro lado del río, siguiendo ciertas restricciones. Estas restricciones establecen que el pastor solo puede llevar consigo a uno de los elementos a la vez y que no puede dejar al lobo junto a la oveja sin su supervisión, ya que el lobo se comería a la oveja. Del mismo modo, no puede dejar a la oveja junto a la lechuga, ya que la oveja se comería la lechuga.

Para abordar este acertijo, se ha implementado un programa en lenguaje de ensamblador MASM x86. El lenguaje de ensamblador es conocido por su cercanía a la arquitectura de la computadora y su capacidad para realizar operaciones de bajo nivel de forma eficiente. Este proyecto busca aprovechar las características del lenguaje de ensamblador para desarrollar una solución precisa y eficaz al acertijo del pastor.

El informe se estructura en diferentes secciones que describen el diseño y la implementación del programa en MASM x86. Además, se presentan las decisiones clave tomadas durante el proceso de desarrollo y se analizan los resultados obtenidos. También se discuten las posibles mejoras y ampliaciones que podrían realizarse en el futuro para perfeccionar el programa.

Código MASM x86

El código no contiene instrucciones de registro específicas. Sin embargo, podemos decir que se implementaron algunos patrones comunes de uso de registros basándonos en el código ensamblador proporcionado.

En el código, se utilizan principalmente los siguientes registros:
eax: Se utiliza para almacenar valores temporales y resultados de operaciones aritméticas.
esp: Se utiliza para administrar la pila. Se ajusta mediante instrucciones como sub esp, <valor> y add esp, <valor>.
ebp: Se utiliza como puntero de base para acceder a las variables locales y parámetros de la función.
ecx: Se utiliza en algunas instrucciones como contador o para almacenar valores temporales.
edx: Se utiliza en algunas instrucciones para almacenar valores temporales.
Es importante tener en cuenta que el uso de registros puede variar dependiendo del contexto y las optimizaciones del compilador. Por lo tanto, la información proporcionada aquí se basa en el código fuente proporcionado y puede no ser exhaustiva. Para obtener información más precisa sobre el uso de registros en el código, sería necesario examinar el código ensamblador generado por el compilador.

El programa proporcionado es un código en ensamblador que implementa un juego de lógica en el que el jugador debe mover objetos de un lado a otro siguiendo ciertas reglas. A continuación, se detallan las especificaciones de uso de los registros en cada subrutina del programa:

loser:

No utiliza registros específicos.
winer:

No utiliza registros específicos.
inRangeStart:

Utiliza el registro EAX para almacenar el valor de entrada.
Utiliza el registro ESP para gestionar la pila.
actualstate:

No utiliza registros específicos.
wherePastor:

Utiliza el registro EAX para comparaciones.
Utiliza el registro ESP para gestionar la pila.
leftOptions:

No utiliza registros específicos.
rightOptions:

No utiliza registros específicos.
questionleft:

Utiliza el registro EAX para almacenar el valor de entrada.
Utiliza el registro ESP para gestionar la pila.
questionright:

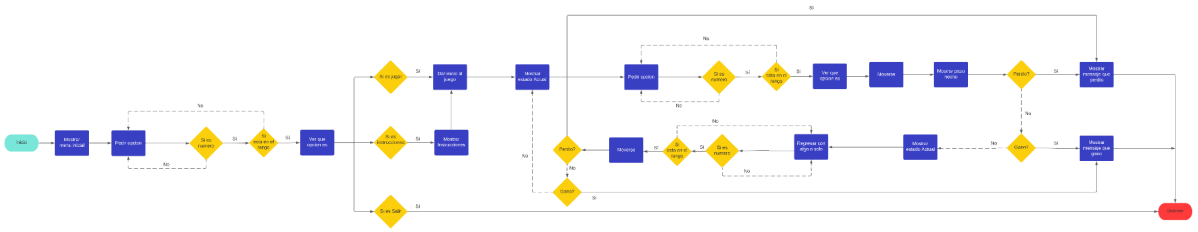
Utiliza el registro EAX para almacenar el valor de entrada.
Utiliza el registro ESP para gestionar la pila.
winlose:

No utiliza registros específicos.
movidasleft:

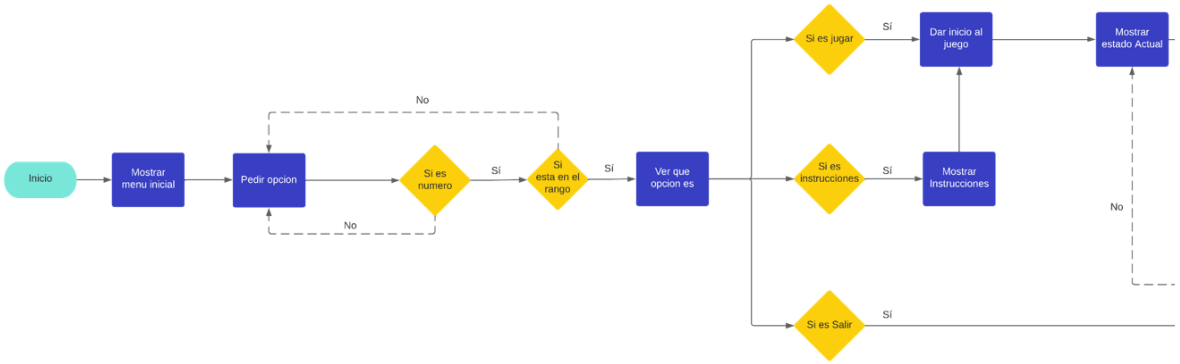
No utiliza registros específicos.
movidasright:

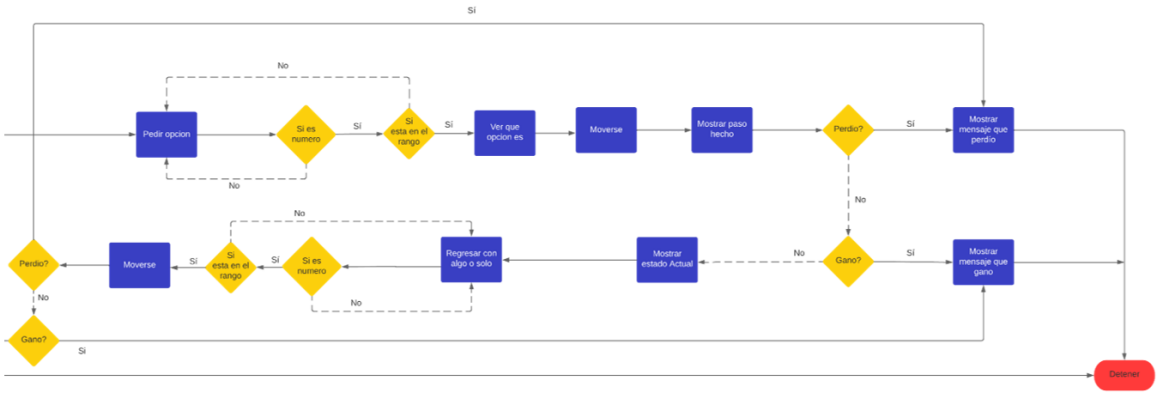
No utiliza registros específicos.
Es importante tener en cuenta que las especificaciones de uso de registros pueden variar dependiendo del compilador y la plataforma específicos utilizados. Las especificaciones proporcionadas se basan en la información disponible en el código fuente proporcionado.

Diagrama de Flujo



Zoom del Diagrama





Conclusiones

- El código utiliza la convención de llamada stdcall y el modelo de memoria plano (.model flat) para las funciones.
- Se incluyen varias bibliotecas utilizando la directiva includelib, incluyendo bibliotecas estándar de C como libcmt.lib, libvcruntime.lib, libucrt.lib, y legacy_stdio_definitions.lib.
- El código define varias cadenas de texto almacenadas en la sección .data, que se utilizan para imprimir mensajes en la consola.
- El código utiliza instrucciones de E/S para leer y escribir datos en la consola, utilizando las funciones printf y scanf.
- El código utiliza variables locales para almacenar información como el estado de los objetos (lobo, oveja, lechuga, pastor) y el estado del juego.
- El código implementa un menú principal con opciones para iniciar el juego, ver las instrucciones y salir.
- El código utiliza estructuras de control como bucles y condiciones (IF) para controlar el flujo del programa.
- El código hace uso de instrucciones de manipulación de pila (push y pop) para pasar parámetros y almacenar valores temporales.
- El código hace uso de registros como eax, esp, ebp, ecx y edx para realizar operaciones y almacenar valores temporales.
- En resumen, el código parece ser un programa de consola que implementa un juego del lobo, la oveja y la lechuga utilizando lenguaje ensamblador x86.