UNIVERSIDAD AUTÓNOMA DE BAJA CALIFORNIA
FACULTAD DE INGENIERÍA CAMPUS MEXICALI
INGENIERÍA EN COMPUTACIÓN

Reporte: Proyecto Final
Breakout

Organización y Arquitectura de Computadoras
Omar Muñoz Urias

Moya Monreal Erick Anselmo - 01110604
Jimenez Barrera Astrid Yamilet - 01182660

Fecha de entrega: 04 de Diciembre del 2025

Ciclo escolar 2025-2
4 de diciembre de 2025

Índice

1. Introducción

El presente proyecto fue desarrollado como parte final del curso de Organización y Arquitectura de Computadoras. Consiste en la implementación de un videojuego completo tipo "Breakout", el cual está escrito utilizando una combinación de lenguaje C con la librería SDL3 para la gestión gráfica, y ensamblador MASM x86 inline para la programación de la lógica interna del juego.

La arquitectura del proyecto se dividió en dos capas principales:

- **Capa de Alto Nivel (C):** Responsable de inicializar la ventana de juego, renderizar los gráficos en pantalla, capturar eventos del teclado y gestionar el bucle principal del juego.
- **Capa de Bajo Nivel (Ensamblador):** Responsable de la lógica central del juego. Aquí se programó el cálculo del movimiento de la pelota, la detección de colisiones, y el manejo directo de estructuras de datos en memoria para gestionar los puntajes.

El objetivo general de la práctica fue demostrar la viabilidad de integrar lenguajes de programación de alto y bajo nivel para crear un sistema eficiente y funcional. Adicionalmente, se buscó aplicar de manera práctica los conocimientos adquiridos durante el semestre, tales como:

- El uso de saltos condicionales (`JMP`, `JE`, `JNE`, `JG`) y ciclos para controlar el flujo de ejecución directamente a nivel de procesador.
- El manejo directo de la memoria RAM mediante direccionamiento indirecto para controlar el arreglo de ladrillos y sus propiedades sin abstracciones del lenguaje de alto nivel.
- La implementación del algoritmo de ordenamiento burbuja completamente en ensamblador para gestionar la tabla de mejores puntuaciones.
- La manipulación de registros del procesador (EAX, EBX, ECX, EDX, ESI, EDI) para realizar operaciones aritméticas y lógicas de manera eficiente.

Finalmente, este proyecto permite visualizar de manera práctica cómo las instrucciones básicas del procesador (operaciones aritméticas, lógicas y de control de flujo) que pueden ser implementadas en ensamblador, interactúan para producir un entorno gráfico interactivo en tiempo real.

2. Problemática y Justificación

La problemática central que motiva este proyecto radica en el cumplimiento de los requerimientos estrictos establecidos para el proyecto final del curso de Organización y Arquitectura de Computadoras. El desafío principal consistió en idear y desarrollar una aplicación que lograra la convergencia entre un lenguaje de alto nivel y el lenguaje ensamblador. Para la

capa de alto nivel, se tomó la decisión de utilizar el lenguaje C, debido a la experiencia previa y práctica que se tiene con su sintaxis.

Sin embargo, dado que las especificaciones del proyecto prohibían explícitamente el uso de la terminal para la interfaz de usuario y exigían un entorno gráfico, fue necesario implementar la biblioteca externa SDL3. La integración de esta herramienta representó un reto técnico considerable durante la etapa de configuración inicial; una vez superado este obstáculo, la problemática se trasladó a la correcta codificación de la lógica interna para asegurar que ambos lenguajes operaran en conjunto.

La justificación para resolver esta problemática es primordialmente académica: completar el proyecto es el requisito indispensable para acreditar la materia. Más allá de la calificación, la realización de este videojuego sirve como el medio para demostrar tangiblemente la adquisición de los conocimientos impartidos durante el semestre, probando que se tiene la capacidad de manipular manualmente los registros del procesador, controlar el flujo del programa y gestionar la memoria de manera eficiente en un entorno real.

3. Objetivos de la Práctica

3.1. Objetivo General

Diseñar e implementar una aplicación de software híbrida que integre un lenguaje de alto nivel con rutinas de bajo nivel, con el propósito de demostrar la aplicación práctica de los conocimientos adquiridos en el curso de Organización y Arquitectura de Computadoras. El proyecto busca cumplir con los requerimientos de evaluación mediante la creación de un entorno gráfico interactivo donde la lógica crítica sea gestionada directamente a través de instrucciones de ensamblador.

3.2. Objetivos Específicos

- **Integración de lenguajes:** Establecer una comunicación eficiente entre el lenguaje C (encargado de la gestión de recursos y la interfaz gráfica mediante SDL3) y el lenguaje ensamblador x86 (encargado del procesamiento lógico), demostrando la interoperabilidad entre ambos niveles de abstracción.

- **Aplicación de algoritmos de ordenamiento:** Implementar manualmente el algoritmo de ordenamiento burbuja (*Bubble Sort*) utilizando instrucciones de ensamblador para gestionar y organizar estructuras de datos en memoria, cumpliendo con el requisito de manipulación de algoritmos clásicos a bajo nivel.

- **Manipular memoria y estructuras de datos:** Utilizar direccionamiento indirecto y cálculo de desplazamientos (*offsets*) en ensamblador para acceder y modificar arreglos de estructuras (*structs*) en la memoria RAM, específicamente para la carga de niveles y propiedades de los ladrillos.

- **Controlar el flujo de ejecución:** Aplicar instrucciones de salto condicional (`JE`, `JNE`, `JMP`) y ciclos para implementar algoritmos lógicos complejos, como el ordenamiento de puntajes (Método de Burbuja) y la selección de dificultad.

4. Marco Teórico

Para comprender la implementación técnica de este proyecto, es necesario definir los conceptos fundamentales de la arquitectura de computadoras y las herramientas de software utilizadas. A continuación, se presentan las bases teóricas sobre la programación híbrida, el manejo de memoria y los algoritmos empleados.

4.1. Librería SDL (Simple DirectMedia Layer)

SDL es una biblioteca de desarrollo multiplataforma diseñada para proporcionar acceso de bajo nivel al hardware de audio, teclado, ratón, joystick y gráficos a través de OpenGL y Direct3D.

- **Función principal:** Actúa como una capa de abstracción que permite al programador escribir código en C para manejar ventanas y renderizado sin interactuar directamente con el driver de la tarjeta gráfica [2].
- **Manejo de Eventos:** SDL utiliza una cola de eventos (*event queue*) para capturar interacciones del usuario. Funciones como `SDL_PollEvent` permiten extraer estos eventos (teclas presionadas, clics) para ser procesados dentro de un bucle infinito conocido como *Game Loop*.

4.2. Ensamblador Inline (`__asm`)

El ensamblador en línea (*Inline Assembly*) es una característica de los compiladores de C/C++ que permite incrustar instrucciones de lenguaje ensamblador directamente dentro del código fuente de alto nivel.

- **Ventaja:** Según Irvine, esta técnica elimina la necesidad de enlazar archivos objeto externos y simplifica el paso de parámetros, ya que el bloque `__asm` puede acceder a las variables declaradas en C por su nombre, sin necesidad de gestionar manualmente la pila de llamadas [1].
- **Uso:** Se utiliza para optimizar secciones críticas de código donde se requiere acceso directo a los registros del CPU o a banderas de estado específicas que no son accesibles desde C.

4.3. Control de Flujo en Arquitectura x86

A nivel de hardware, el procesador ejecuta instrucciones de manera secuencial a menos que se modifique el registro Puntero de Instrucción (EIP).

- **Saltos Condicionales:** Son instrucciones que transfieren el control a una nueva dirección de memoria solo si se cumplen ciertas condiciones en el registro de banderas (*EFLAGS*). Ejemplos comunes incluyen:
  - `JE` (Jump if Equal): Salta si la bandera Zero Flag (ZF) es 1.

- `JG` (Jump if Greater): Salta si el resultado de una comparación anterior indica que el primer operando es mayor que el segundo (basado en banderas de signo y desbordamiento) [3].

- **Ciclos:** En ensamblador, los bucles se construyen combinando etiquetas, comparaciones y saltos condicionales, o utilizando la instrucción `LOOP` que utiliza el registro `ECX` como contador automático.

4.4. Direccionamiento de Memoria y Arreglos

La memoria RAM se organiza linealmente. Para acceder a elementos dentro de una estructura de datos o un arreglo, se utiliza el **direccionamiento indirecto base más desplazamiento**.

- **Principio:** Si un registro (como `ESI`) contiene la dirección base de un arreglo, se puede acceder al siguiente elemento sumando el tamaño del dato (offset). Por ejemplo, en un arreglo de enteros de 4 bytes, el segundo elemento se encuentra en `[ESI + 4]` [4].

4.5. Algoritmo de Ordenamiento de Burbuja

Es un algoritmo de ordenamiento sencillo que funciona iterando repetidamente a través de la lista que se desea ordenar.

- **Funcionamiento:** Compara elementos adyacentes y los intercambia (*swap*) si están en el orden incorrecto. Este proceso se repite hasta que no se requieren más intercambios.

- **Complejidad:** Aunque no es el más eficiente para grandes volúmenes de datos ($O(n^2)$), su implementación a nivel de ensamblador es ideal para fines educativos porque ilustra claramente el uso de bucles anidados, comparaciones y movimiento de datos en memoria [5].

4.6. Unidad de Punto Flotante (FPU x87)

La FPU es un coprocesador dedicado a realizar operaciones con números reales (decimales). A diferencia de los registros generales (EAX, EBX), la FPU utiliza una **pila de registros** de 80 bits (`ST0` a `ST7`).

- **Instrucciones clave:**

  - `FLD`: Carga un valor de memoria y lo empuja al tope de la pila (`ST0`).
  - `FSTP`: Saca el valor del tope de la pila y lo guarda en memoria.
  - `FCOMIP`: Compara el tope de la pila con otro valor y actualiza las banderas del CPU para permitir saltos condicionales [1].

5. Procedimiento

El desarrollo del proyecto se llevó a cabo siguiendo una metodología estructurada de ingeniería de software, dividida en fases de configuración, implementación, integración y pruebas. A continuación, se detallan los recursos utilizados y los pasos ejecutados.

5.1. Materiales y Equipos

Para la realización de esta práctica se utilizaron los siguientes recursos:

- **Hardware:** Computadora personal con arquitectura de procesador x64 (Intel/AMD).
- **Entorno de Desarrollo Integrado (IDE):** Microsoft Visual Studio 2022.
- **Lenguajes de Programación:** Lenguaje C para la estructura general y Ensamblador MASM x86 para la lógica de bajo nivel.
- **Librerías Externas:** SDL3 (Simple DirectMedia Layer) y SDL3_ttf para la gestión gráfica.

5.2. Condiciones Experimentales y Configuración

Aunque el equipo de cómputo opera bajo una arquitectura de 64 bits, se estableció una condición de compilación específica debido a las restricciones del compilador de Microsoft:

- **Modo de Depuración x86:** Dado que Visual Studio no soporta bloques de ensamblador en línea (`__asm`) para la plataforma x64, fue obligatorio configurar el proyecto para compilar y depurar en modo **x86 (32 bits)**. Esto permitió el acceso directo a los registros extendidos de 32 bits (`EAX`, `EBX`, etc.) y la correcta integración del código máquina con el código C.
- **Vinculación (Linking):** Se configuraron las dependencias del proyecto para enlazar estrictamente con las versiones x86 de las librerías SDL3, asegurando la compatibilidad binaria.

5.3. Descripción Detallada del Procedimiento

1. **Configuración del Entorno Gráfico:** Se inició creando la estructura base en C. Se inicializó la librería SDL3 y se creó una ventana de resolución lógica de 1400x900 píxeles. Se implementó el bucle principal del juego (*Game Loop*) encargado de limpiar la pantalla, procesar eventos y renderizar los cuadros por segundo.
2. **Definición de Estructuras de Datos:** Se diseñaron las estructuras (`struct`) en C para representar los objetos del juego: `Jugador` (para el nombre y puntaje) y `Ladrillo` (para posición, resistencia y estado). Esto definió el mapa de memoria que posteriormente sería manipulado por el ensamblador.

3. **Implementación de Lógica en Ensamblador:** Se procedió a sustituir funciones críticas de C por bloques de ensamblador *inline*:
    - **Carga de Nivel:** Se escribió la rutina para recorrer la matriz de patrones y calcular la posición en memoria de cada ladrillo usando direccionamiento base más desplazamiento.
    - **Cálculo de Velocidad:** Se implementó la lógica matemática utilizando la FPU, cargando variables `float` en la pila `ST(0)`, operando sobre ellas y devolviendo el resultado a la variable de C.
    - **Ordenamiento:** Se codificó el algoritmo de burbuja para ordenar el arreglo de estructuras de jugadores, manipulando punteros de memoria directamente.
4. **Integración y Pruebas:** Se integró la detección de colisiones y la física del juego. Se realizaron pruebas de ejecución paso a paso (debugging) inspeccionando los registros del CPU en tiempo real para asegurar que los punteros no accedieran a zonas de memoria inválidas y que la pila de la FPU se limpiara correctamente después de cada cálculo.

5.4. Precauciones de Seguridad y Manejo de Recursos

A nivel de software, se tomaron precauciones críticas para evitar errores de ejecución:

- **Integridad de la Pila FPU:** Fue necesario asegurar que cada instrucción de carga (`FLD`) tuviera su correspondiente descarga (`FSTP`) para evitar el desbordamiento de la pila de registros flotantes.
- **Alineación de Memoria:** Se verificó que los desplazamientos (*offsets*) utilizados en ensamblador coincidieran exactamente con el tamaño de los tipos de datos en C para evitar la corrupción de la información en las estructuras.

6. Resultados con pruebas

- Presentación de los datos obtenidos durante el experimento.
- Tablas, gráficos, fotografías u otros recursos visuales que ayuden a interpretar los resultados.
- Análisis de los datos, incluyendo cálculos si son necesarios.

7. Conclusiones

- Recapitulación de los principales hallazgos del experimento.
- Interpretación de los resultados en relación con los objetivos planteados.
- Discusión de las implicaciones prácticas y teóricas de los resultados.
- Posibles limitaciones del experimento y áreas para futuras investigaciones.

Referencias

[1] Irvine, K. R. (2019). *Assembly Language for x86 Processors* (8th ed.). Pearson Education. Capítulos 6, 9 y 12.

[2] LibSDL.org. (2024). *SDL3 API Reference Documentation*. Recuperado de https://wiki.libsdl.org/SDL3/

[3] Stallings, W. (2016). *Computer Organization and Architecture: Designing for Performance* (10th ed.). Pearson.

[4] Tanenbaum, A. S. (2013). *Structured Computer Organization* (6th ed.). Pearson Prentice Hall.

[5] Patterson, D. A., & Hennessy, J. L. (2014). *Computer Organization and Design* (5th ed.). Morgan Kaufmann.